

Algoritmos Vivos

Metodologia para aulas:

práticas: Sempre o aluno precisa trabalhar os conceitos vistos na aula.

exigentes: O exercício de cada aula vale cerca de 5% da nota bimestral.

individuais: O exercício que o aluno tem que fazer é único. Ninguém mais na sala (nem no universo) tem aquele exercício.

divertidas: Não sei se exagero ao chamar uma aula de divertida, mas a falação do professor está limitada a 30 minutos, mais ou menos. Depois o aluno tem que por mãos à obra.

respeitosas: Cada exercício feito pelo aluno é corrigido e a ele devolvido no prazo de uma semana. A coleção de exercícios corrigidos sobra como memória dos assuntos estudados em sala de aula

flexíveis: A lista de exercícios prontos a serem usados nesta metodologia é de cerca de 800.

P. Kantek
Curitiba
1993-2024

Metodologia de Trabalho

*Tudo o que precisamos aprender,
aprendemos fazendo... Aristóteles, Ética*

Esta metodologia de trabalho em sala de aula, denominada ALGORITMOS VIVOS, resulta da convicção de que não se aprende ouvindo, vendo, prestando atenção ou apenas sendo estimulado. A única maneira de aprender é fazendo.

Resta ao professor ser criativo para propor aos alunos coisas que eles possam fazer e que sendo minimamente interessantes ajudem a consolidar os conceitos estudados.

Este *insight* surgiu na minha cabeça em 1993, quando professor de estrutura de dados da PUC/Pr tentava ensinar o funcionamento do algoritmo de compressão de Huffman. Tarefa inglória, já que esse algoritmo é tudo de bom que um algoritmo pode ser: abstrato, conciso, elegante, preciso, algo complicado mas sobretudo brilhante. Me lembro de ter pensado, como seria bom se o computador (que na época começava a se tornar disponível e farto) pudesse gerar um exercício diferente para cada aluno.

Com isso, o desafio seria maior: cada aluno tinha o seu exercício, não haveria possibilidade de compartilhamento de resultados (conhecido como cola), se bem que haveria e muito a possibilidade de colaboração e trabalho em grupo: todos os exercícios têm a mesma definição, e o resultado seria facilmente corrigível e com resultado não sujeito à discussão.

Diante do sucesso da primeira iniciativa (o exercício código 451 na lista a seguir) começou uma grande diversão que tem sido a invenção e programação de novos exercícios.

O que começou em algoritmos, esvaiu-se para programação, inteligência artificial, compiladores, lógica matemática, métodos numéricos, linguagens, etc etc.

Curitiba, 1 de janeiro de 2026
Pedro Luis Kantek Garcia Navarro
pkantek@gmail.com

Materializando abstrações: O uso de Algoritmos Vivos em sala de aula

Pedro Luis Kantek Garcia Navarro

pkantek@gmail.com

CELEPAR / Universidade Positivo Al. Dr. Carlos de Carvalho 250, ap52 80410-180 Curitiba PR
AV e algoritmos vivos ©1992 por Pedro Kantek

O currículo LATTES do autor pode ser visualizado em <http://lattes.cnpq.br/7330616928412664>

Resumo

Comenta-se a dificuldade de ensinar algoritmos e programação. Dificuldade esta, crescente pela universalização de acesso aos cursos universitários de informática. Aumentando em número, tais cursos agregam mais e mais interessados, porém o desempenho médio desses alunos parece diminuir. Vem também a constatação de que o uso do computador na empreitada de ensinar programação é pobre: limita-se a ser compilador e executor de programas. Apresenta-se a idéia de algoritmos vivos, como ferramenta para materializar um algoritmo: o abstrato torna-se concreto. Princípios de seu uso e funcionamento são a seguir apresentados. Um exemplo de algoritmo vivo usado para transmitir as idéias básicas da computação evolutiva é apresentado. A lista dos mais de 200 algoritmos vivos existentes é mostrada e comentada. Conclui-se pela pertinência de seu uso e melhora dos resultados obtidos em situações reais de aprendizado de programação de computadores.

Palavras chave

programação, algoritmos, prática de programação, pedagogia da informática, ensino de algoritmos, ensino de programação.

Antecedentes

Poucas tarefas são tão difíceis como ensinar a projetar, escrever, implantar, depurar e manter algoritmos. Seja porque são meras abstrações, ou porque trata-se de matemática aplicada ou ainda porque ao adquirir "vida" os algoritmos tornam-se mais e mais complexos de serem apreendidos, o caso é que apenas uma pequena parcela dos alunos é que pode ser apresentada aos algoritmos e ato contínuo, sair os criando e os usando. Tom de Marco descreve em seu livro *Peopeware*, que é normal encontrar diferenças na produtividade de programadores de 10:1 [1]. Trazendo esta diferença de desempenho para dentro da sala de aula, percebe-se que há que se usar de criatividade para permitir algum aprendizado a todas as pessoas, independente de sua localização no espectro de capacidade e produtividade acima citado.

Para os que formam a maioria e que se localizam nas faixas menores de produtividade, o aprendizado é difícil e demorado. Muitos exercícios há que fazer e muitas horas passar diante de um computador para que este, pacientemente – embora com teimosia – vá mostrando como são as entranhas de um algoritmo, o que o move e como os resultados (in) esperados são produzidos.

Há que estimular os alunos, instigá-los e indicar os caminhos do aprendizado. A simples colocação de conceitos, algoritmos, casos exemplo e princípios de funcionamento na expectativa de que esses conceitos e princípios sejam absorvidos pelos alunos, tem se mostrado expectativa quase irrealizável. Muitos dos alunos parecem não saber "por onde começar".

Este estado de coisas tem sido agudizado nos últimos anos pela explosão de novos cursos de informática. Considerando a crescente demanda por mão de obra, e o conseqüente interesse crescente da comunidade de entrantes no terceiro grau, as universidades e faculdades tem expandido largamente a oferta de tais cursos.

Na cidade de Curitiba, que tem um porte médio, existem hoje mais de 30 cursos superiores diretamente relacionados à informática. Há 10 anos, este número era inferior a 10.

A nova legislação do ensino superior (Decreto Lei n. 2306 de 19/08/97) [2], facilitou a criação de novas universidades e centros universitários e propiciou a entrada de vastos contingentes humanos no terceiro grau. Esta situação tem o mérito indiscutível de alavancar a qualidade de vida das pessoas através da educação. Há indiscutível correlação entre os índices de desenvolvimento de uma sociedade e o número de pessoas portando diploma superior.

Um subproduto menor da situação acima descrito traz implicações para os que trabalham no ensino da informática. E, do nosso ponto de vista, implicações negativas diga-se. Trata-se da diminuição da relação candidato / vaga na entrada dos cursos universitários de informática. Embora eu desconheça estudo conclusivo, a prática mostra que quanto menor é esta relação, menos qualificados são os alunos entrantes. Maiores, portanto, as dificuldades na apresentação e posterior apreensão do conceito de algoritmos.

Em nossa instituição, a Universidade Positivo (UP), há 10 anos, a relação candidato / vaga para o curso de informática era de 4 candidatos por vaga para os cursos noturnos e de um pouco mais de dois para um nos cursos diurnos. Esta relação se manteve mais ou menos constante por diversos anos, ainda que em um deles, tenhamos tido uma procura atípica que elevou os índices para 7:1 e para 4,5:1 respectivamente. Entretanto, nos últimos 3 anos, em nenhum vestibular, qualquer das relações foi superior a 1:1. Isto significa que nos últimos 3 anos, sempre houve mais vagas do que candidatos.

O impacto desse estado de coisas para o ensino de programação é significativo. Subitamente vimos frente a grupos de alunos que entraram no curso sem outra credencial além da habilidade de não zerar nenhuma prova do vestibular. Seu desempenho em disciplinas do segundo grau, particularmente a matemática, irmã gêmea da programação, deixava a desejar.

Ainda introduzindo o assunto, cabe uma constatação: as práticas de ensino de informática tem usado muito pouco o potencial representado pelo computador na sala de aula. Afora as indefectíveis aulas de laboratório onde – em geral – um professor escreve algoritmos no quadro e os alunos copiam-no e tentam executá-lo, pouco esforço parece ter sido dispendido na busca de maneiras de pôr o computador a auxiliar o aluno e o professor nesta tarefa. Parece ser mais uma manifestação do adágio "casa de ferreiro, espeto de pau". Há falta de softwares, metodologias, iniciativas e relatos de experiências que usem a máquina para realmente ajudar a programar, afora o uso óbvio que é pela compilação e execução de exercícios.

Comecei a ensinar informática em 1976, e algoritmos e programação a partir de 1989. A partir de 1993 passei a lecionar também estruturas de dados, que nada mais são do que "algoritmos e programação" mais sofisticados. Conhecendo um pouco da capacidade e habilidade dessas máquinas, desde que programadas, comecei a pensar em como elas poderiam ajudar alunos e professor em atingirem seus alvos comuns.

Nasceu assim, a partir de 1992, a idéia dos algoritmos vivos. Ainda não tinham este nome, nem a estrutura conceitual que será a seguir apresentada, mas não importa: aquele exercício que pedia aos alunos para criar e usar uma árvore de Huffman para codificar e comprimir uma pequena frase era a semente de algo que iria crescer e frutificar.

O que é um algoritmo vivo

Na visão de hoje, os algoritmos vivos são uma ferramenta de ajuda ao ensino e aprendizado da programação. Do ponto de vista do aluno, trata-se de uma folha de papel contendo três coisas:

- A teoria subjacente a um determinado tópico, incluindo-se os algoritmos necessários escritos em pseudo-código.
- Um exemplo completo do uso desses algoritmos, especificando-se os dados de entrada, o tratamento sofrido por esses dados e o resultados que eles produzem. Este exemplo é o mesmo para todos os alunos.
- Uma nova instância de execução, na qual apenas os dados de entrada são apresentados, pedindo-se ao aluno que, em seguida, produza o resultado que o algoritmo produziria se executado fosse em um computador. Esta nova instância tem dados diferentes para cada aluno. As respostas sempre são um conjunto pequeno de dados.

Cada aluno recebe uma folha, personalizada, com seu nome e demais informações que auxiliarão na correção automatizada do exercício. Obviamente, se cada aluno tem a sua folha, os dados que a compõe podem ser distintos. Relembre-se que esta folha é emitida por um programa de computador, que portanto, pode e gera dados distintos para cada aluno.

Do ponto de vista do professor, trata-se de ferramenta importante, já que permite que cada aluno se defronte com um problema diferente, impedindo aquele comportamento muitas vezes usual: um ou dois fazem os exercícios, enquanto os demais divagam, sendo o resultado obtido ao final devidamente socializado entre todos.

Ao liberar o professor de conduzir a iniciativa comum de solucionar a questão, o uso desta técnica permite que ele atenda os alunos, atacando as dificuldades de cada um de maneira individual. Permite também que cada aluno siga o seu ritmo e trabalhe na velocidade que lhe é peculiar, liberando-se a turma da necessidade de todos irem atrelados a um determinado ritmo e que em geral é determinado pelos alunos que maior dificuldade apresentam.

Certamente isto está de acordo com a moderna pedagogia que apregoa a necessidade de limitar aulas meramente expositivas. Tenho apresentado esta questão a meus alunos de maneira simples e direta: quem tem que trabalhar é o aluno. O professor, supostamente, já sabe isso tudo. A ele cabe ajudar os que tem dificuldade e não atrapalhar, saindo da frente dos que vão sozinhos. Essa idéia não é de minha autoria. Outros, mais abalizados já o disseram, como por exemplo, Lauro de Oliveira Lima: "o problema da aula expositiva é que, ou é intragável e suporífera ou é brilhante, cativante e dramática. No primeiro caso, simplesmente não existe. No segundo, o próprio brilhantismo do orador e o interesse em acompanhar a exposição, impedem qualquer reflexão, ..., de modo que terminada a aula, ficam boiando na consciência, apenas alguns pontos pitorescos, quase sempre os menos relevantes". [4] Coube-me meditar sobre o assunto, admitir a veracidade do enunciado e trabalhar para acabar com as aulas só expositivas.

A possibilidade de realizar muitos trabalhos individuais ao longo do bimestre, tipicamente mais de 10, permite que sejam atribuídos graus a tais trabalhos, minimizando a importância, inclusive quantitativa, da prova bimestral. Assim, o conceito numérico do aluno vai sendo construído dia-a-dia, dando chance aos que apresentam maior dificuldade, orientando o estudo, diluindo-o ao longo do tempo e não apenas apresentando uma prova que dará origem a um grau absoluto e irrecorrível.

Na Universidade Positivo, onde os algoritmos vivos estão sendo usados em sua última encarnação, eles têm contribuído com 60 % da nota bimestral, reservando-se os 40 % restantes para a prova, que até mesmo por ter tido sua importância relativa diminuída, pode ainda continuar sendo de uma exigência formal e conceitual absoluta. Se toda a nota do bimestre corresse por conta apenas do exame, os critérios de correção teriam que ser relaxados, sob pena de médias baixas e conseqüente insucesso de grande parte dos alunos.

O programa que gera os algoritmos vivos, gera também as respostas dos mesmos. Afinal, se cada aluno tem um exercício diferente, há que se dar ao professor alguma facilidade na correção. Isto ocorre de tal maneira que é possível até, transferir a tarefa de correção a monitores ou outros auxiliares que não precisam conhecer o conteúdo ou o tema de estudo. Tal resposta é gerada em papel e também em mídia magnética, permitindo uma correção vis-a-vis com a resposta dada, ou alternativamente a digitação dos dados gerados pelo aluno. Neste caso, depois de corrigí-los, o programa gerador dos algoritmos vivos produz um arquivo que contém a identificação do aluno, a identificação do evento (neste caso um exercício e qual deles) e o grau obtido. Um outro software de minha autoria denominado PANA, que gerencia alunos, turmas, conteúdos, datas, provas, exercícios e notas captura tal arquivo e ao final do bimestre, calcula médias e as envia - por meio eletrônico - para a secretaria da instituição. Afinal, como se disse acima, quem tem que trabalhar é o computador.

Finalmente, como a emissão de uma coleção de exercícios individualizados é obtida apenas "pressionando-se alguns botões", o uso de algoritmos vivos permite variações interessantes na sua aplicação. Assim por exemplo, quando um resultado ruim é alcançado, o mesmo exercício pode ser refeito, várias vezes até. Não há problemas, já que a cada emissão, embora o enunciado permaneça o mesmo, os dados de entrada são distintos. Outra abordagem interessante é a criação de pequenos grupos, cada um com um exercício, para que ocorra o trabalho conjunto. Isto é usado em algoritmos vivos difíceis e/ou demorados.

Particularmente, eles tem sido usados nos últimos 2 anos, em uma modalidade que aparentou ser a melhor, depois de diversas tentativas, acertos e erros: Uma aplicação de um determinado algo-

ritmo é feita, normalmente em sala de aula. Nesse dia, o resultado não é levado em consideração para a nota bimestral, e conseqüentemente os resultados esperados estão à disposição dos alunos que podem assim acompanhar seu desempenho e discutir com o professor acertos e erros. Esta primeira execução serve também para apresentar o exercício aos alunos. Em uma próxima aula, com alguns dias de intervalo (o que permite que interessados implementem e experimentem o algoritmo rodando em computador), uma nova emissão é apresentada. Nesta segunda oportunidade o exercício vale nota e o resultado não está disponível aos alunos. Estes o receberão apenas no futuro, já com o exercício devidamente corrigido e valorado.

Para encerrar esta apresentação, vale citar que a partir de 2001, o arcabouço conceitual dos algoritmos vivos foi usado para uma variante do software que gera provas. Aqui, a ênfase está em perguntas e respostas esperadas. A vantagem é a de permitir que cada aluno tenha uma prova diferente, sempre com questões de complexidade e temática similares, todas retiradas de um banco de questões especialmente preparado. Os resultados desta última iniciativa ainda não estão disponíveis, devido ao pouco tempo de uso.

Anatomia de um algoritmo vivo

Os algoritmos vivos estão 100 % programados em APL [3]. Embora tal linguagem não seja de uso disseminado, nem conte com muitos adeptos, ela definitivamente deixa qualquer outra atrás quando se trata de escrever programas exploratórios, complexos, com alto grau de manuseio nos dados de entrada. Perde, sem nenhuma dúvida, em desempenho, mas isto aqui não é importante. Sua interface é ruim, ela usa um alfabeto especial e é orientada a caracter, mas ultrapassado este desconforto, abre-se um paraíso ao programador. A linguagem parece ter sido construída ad hoc para os algoritmos vivos. Com ela é possível produzir um novo exercício em poucas horas. Desconfia-se que ao usar qualquer outra linguagem, seriam semanas e não horas.

Já estão disponíveis todas as funções que implementam as partes comuns dos algoritmos vivos. Para cada novo exercício é necessário criar apenas as seguintes funções específicas daquele exercício:

- Uma função - usualmente denominada GERACASO - que gera um caso novo, aleatório, mas que permite solução em tempo hábil (estimado em cerca de 1 hora de trabalho para o aluno).
- Um conjunto de funções - dependentes do problema - que recebem o caso acima gerado e o resolvem gerando o gabarito para o professor.

Catálogo

Para a construção dos algoritmos vivos criei um índice de catalogação, parecido ao em uso nas bibliotecas que permite numerar os algoritmos vivos e inserir cada um em um contexto amplo da ciência da computação. Os grandes temas desta classificação são:

- 1 generalidades
- 2 alocação encadeada e apontadores
- 3 listas, pilhas e filas
- 4 árvores
- 5 ordenação, estruturas tipo hashing e cadeias
- 6 grafos
- 7 tópicos avançados
- 8 outros
- 9 não categorizados anteriormente

Conclusões

1. As aulas de estruturas de dados e de tópicos avançados em informática tem sido mais participativas, produtivas e agradáveis. Alunos têm manifestado o interesse que a abordagem lhes desperta. Algoritmos que assustam pela complexidade quando são apresentados, revelam sua singeleza quando postos em execução. Ou, ao contrário, os detalhes dinâmicos de um algoritmo aparentemente simples, quando vivenciados pelos alunos, mostram sua complexidade.

2. O entendimento e a depuração de programas baseados nos algoritmos estudados fica facilitado, relatam os alunos. De posse de um algoritmo vivo preenchido e entendido é menos difícil projetar e construir um programa de computador que funcione.
3. As médias bimestrais das turmas onde os algoritmos vivos são usados têm crescido. A disciplina de estrutura de dados e arquivos, que há vários anos, tinha o duvidoso título de "a que mais reprova alunos", perdeu-o, e passou a ter índices similares, e em certos casos, melhores do que as demais disciplinas do curso.
4. A parte expositiva da matéria, ainda e sempre necessária, foi reduzida, já que algoritmos vivos consomem muito tempo em sala de aula. Ela precisou ser apurada, concentrada, despida de trivialidades. Desconfia-se que o que permaneceu, ficou enriquecido e fortalecido.
5. A perda de alunos ao longo do ano também diminuiu. Era comum a desistência de grandes contingentes de participantes. Alunos que aparentavam estarem "perdidos" iam desistindo, sobretudo no início do curso. Hoje, o número de desistentes é menor e é compatível com as demais disciplinas do curso.
6. Finalmente, os algoritmos vivos tem ajudado a planejar e dar versões mais enriquecidas das disciplinas de estruturas de dados e de tópicos avançados. Embora representem uma carga adicional de trabalho para o professor na preparação e correção, esta carga tem sido contrabalançada com os resultados positivos obtidos.

Bibliografia

1. De MARCO, Tom e LISTER, Timothy. Peopleware. Como Gerenciar Equipes e Projetos Tornando-os mais produtivos. Ed McGraw Hill. 1990.
2. DECRETO LEI n. 2306, de 19 de agosto de 1997, que regulamenta as Instituições de Ensino Superior no Brasil. Disponível na íntegra em www.mec.gov.br/Sesu/ftp/Decreto2306.doc.
3. KANTEK, Pedro. Uma Introdução ao APL. CELEPAR. 1982.
4. LIMA, Lauro de Oliveira. Treinamento em Dinâmica de Grupo. Vozes. 1973.

Apresentação

Esta metodologia foi apresentada em um congresso internacional de Pedagogia na área de informática e publicado nos anais, como se pode ver em NAVARRO, P. L. K. G. . *A Toolbox To Teaching And Learning Data Structures: Live Algorithms*. In: *International Conference on Education and Information Systems: Technologies and Applications (EISTA '03), 2003, Orlando, 2003.*.

A toolbox to teaching and learning data structures: "Live Algorithms"

Pedro Luis Kantek Garcia Navarro.
Centro Universitário Positivo, UNICENP.
Curitiba, Paraná, 80410-180, Brazil.
E-mail: kantek@pr.gov.br

ABSTRACT

This article proposes a toolbox to help teachers and students in a hard task of teaching and learning algorithms and data structures. The name of that toolbox is "Live Algorithms". The text starts with issues of abstracting manipulations. Next, Live Algorithms is explained, where I intend to show how it could help us. The toolbox is described, its components are commented and the advantages of its use are listed. Finally, we compare the results obtained in two regular classrooms: the first worked with Live Algorithms and the other, without it, in a traditional approach.

Key-words: data structures learning, algorithms learning, toolbox algorithms teaching.

1. INTRODUCTION

Many authors, as Niklaus Wirth (1986) for instance, has pointed how the duty of teaching and learning algorithms as well as programming is very hard. In fact, to develop the capacity of building software, mainly those big size and complexity is very arduous challenge to people and organizations.

In the past, some kind of approaches has been taken to make easier this learning. One, and very important, was the utilization of a new algorithmic language; nevertheless, seems like any programming language, already known. To justify the approach, authors may say "When we separate *what* and *how* will be made something, we introduce an important task in the project of someone program: the write, test and expurgation of the algorithm. Not as a piece of programming code, but as an application of one idea.

When someone combines the project algorithm task with it's implementation in a real computer using available program language, probably introduces unnecessary complexity. As an example, when someone studies the Quad Tree algorithm, all the emphasis must be placed in the structures idea, functionality, and its fundamental characteristics. The organization used in the implementation (language, kind of data, primitive or derived structures, statements, and/or available facilities of the development and environment) must come right after its understanding, projecting, testing,

expurgating and approving as an algorithm. As soon as someone could be sure about its robustness and proper functionality, the work in a real computer could be initiated.

An important comment here, is that the only pre-request to begin the algorithm study, and also the program construct, is only a project, paper, pencil, and a lot of human understanding. No computer is needed

2. PREVIOUS PROPOSES

A very frequent one, in scientific literature, is when someone tries to study the actions in a production of animated sequences that go together with the algorithm. This approach has as the main advantage of materializing function. One example of this approach was the sounded and colored 30 min movie *Sorting Out Sorting* made in 1981. [Baecker, 1981]. It compares several kind of methods. Another examples are Balsa method by Mark Brown (1984) and John Stasko TANGO (1990). At Internet there are JELIOT software [Haajanenn et alli, 1997]. The start message of TANGO is emblematic: *Welcome to the algorithm theater!*

One problem here is the passive role lived by the student. He is invited to admire, to see the algorithm spatial development. How much learning each student can acquire and how much real utility by increasing knowledge this method brings to people is an open question. See [Byrne at Alli 1996]. More recently, wider proposals that appear will ask to the student to predict the algorithm behavior.

Another important question is the low generality from the proposals of the method. The majority concentrates a unique aspect of the computer's science. So, the movie above shows and compares only few sort methods. The Hill work [2002] deals only with tree algorithms. It seems there is very difficult to be generalist in animated matter.

3. THIS PROPOSAL

The Live Algorithm begins showing to the student one precise algorithm. So, he (or she) is invited to answer some kind of questions about the algorithm showed. It can be a variable value in some algorithms' point. Or can be an

algorithm full answers for some input data. What or how many statements will be performed.

The exercise doesn't ask to development of a new algorithm. This task comes later, when the outstanding algorithms knowledge stand. Wirth [1986], by example, defends this strategy.

4. LIVE ALGORITHMS OBJECTIVES

- The student is demanded to make a task which has a knowledge of functionality algorithm already presented as pre-requested
- Help the student to understand the algorithms' dynamic aspect and its structures of outstanding data;
- Have wide spectrum including the main themes studied in data structure and algorithms' project;
- Be independent of having computers in classroom;
- Have facility and agility for correcting papers; as a result, the pupil will have fast feed back;
- Make distinct postulation for each student in a way that, everyone should produce his own work, not allowing treachery.
- Allow matching duties in order to obtain best grade's average in which the theory test has less weight.
- Permit the students to make papers out of classroom as a contribution end effort to raise their grades
- Show practicing how algorithms, which seems simple can be very complex behavior.

5. LIVE ALGORITHMS DESCRIPTION

Many programming computers (presently 108) generate papers containing individual exercises. All exercises have the same aspect, because they share an expressive part of the produced code.

If the code is not shared then it has: introductory text about the studied problem; a case generator for a set of input data for this problem; a solver of the above case.

Each exercise paper has a complete example of a resolved exercise with comments and one new the student is asked to solve. To teacher, Live Algorithms produce a list with the correct answers questions. Each of generators was code in APL one of the bests programming language for this kind of application. The APL program generates a LATEX file, so, before the student's use, this file needed to be compiled. Now, the visual aspect of Live Algorithms is very nice.

The generator case is very hard to produce. How the generation of data is random this part of software has the responsibility to find an exercise that would be interesting to the student. It must be non trivial, and it must have an acceptable complexity. People (well, almost all people) must solve the exercise in 45 minutes. Particularly cases, which not deal with algorithm main idea, must be rejected.

How is produced an exercise by computer program if each student has a different case about the same algorithm? So,

the effort demanded is the same. One student can't copy the neighbor's paper, because of the differences.

6. EXAMPLES

Here, it is a title of Live Algorithms.

Turing Machine: the exercise shows theory, and a small TM. There is also, an input stream of bits. The student is invited to say in what state the TM was conducted.

Many have algorithms examples (sorts, searches, arithmetic manipulation, nested ifs and else's, mix of whiles and repeats, etc). In this exercises the student is invited following the flow and predict what are the results found when the algorithms end.

One LA has the Clavius and Lilus algorithm to find the day of Eastern in all the years past 1587. The student must have found the Eastern in a random year offered by the program.

The recursive approach has around 10 different exercises: The chess horse path across $n \times n$ board, the anagrams producer, and the quad tree implementation. In all cases, the student must answer what is the final computation result.

The fundamental structures (stacks, queues, lists, linked lists, skip lists and similar) have 14 exercises. One important question to student is finding errors in linked lists dumped.

Application of linked lists is also presented. An MS-DOS alocator (FAT processor) and a UNIX alocator ask to the student to construct the control tables in both environments.

The tree chapter is the wider. There are theory exercises, with inclusions, deletions, and replacing in binary trees, B-trees, binary search trees. There is also, practical approach: Huffman trees, dBase trees (ndx files), Sybase trees.

Graph theory is also presented. Minimal path, Dijkstra, Kruskal, Warshall, Ford-Fulkerson, Edmonds-Karp Algorithms, topological sort, etc.

In sort methods, there are 5 papers of exercises. Beginning with the simply ones (bubble, insertion, selection, shake, etc) and beyond: Shell, quick, heap etc. There are also nets of comparison.

In artificial intelligence world there are some algorithms: A*, Minimax, Genetic algorithms, neural nets, expert systems. By the way, these examples are almost trivial and the objective is demonstrating the main functionality of each one.

Petri Nets is here. 3 exercises were produced. An ATM machine, semi-adder and a randomic petri net. Theory needed is described in the exercise, to help the student to answer the question.

Images BMP and GIF are also studied. The internal characteristics are showed. All kind of bit mapped images, the LZW algorithm, the convolution filters, the histogram equalization algorithms are here.

Cryptography is other main chapter: The DES and the RSA approach are exposed and also, the ENIGMA machine. Steganography exercise, signature in files, and an invitation to break one RSA communication are examples of the existence exercises.

7. COMPARISON BETWEEN CLASSIC AND LIVE ALGORITHMS APPROACHES

Before the existence of this toolbox, any matter of algorithms or data structures was presented through the sequence tasks:

- a) The topic presentation, its importance in the computer science and informatics profession.
- b) Main algorithms presentation, working with the students in algorithms flows.
- c) Aspects of generality, complexity, advantages and disadvantages of each one
- d) Practical cases presentation, where these algorithms are used
- e) Question the students to produce a new exercise about these matters.

Comments about:

1. The a,b,c and d tasks don't demand the student an active work. So, the feedback that teacher receives is small and without meaning.
2. When the "e" task begins, the majority of the students copy each other.
3. Serious difficulty: or the students quit during the scholar time, or the teacher needs to open the criteria necessary to graduate the student.

When the Live Algorithms is used, the task list above can be rewrite as

- a) Topic presentation and comments about its importance in computer science
- b) Main algorithms presentation
- c) Practical classes using Live Algorithms: Each student gets an individual challenge. Some can do it in fewer minutes. Other needs one or two working days. It doesn't matter.

Comments about:

1. The students play an active role in learning with LA. The majority said the learning is facilitated. More people can produce some kind of programming code when compared with other classes that use traditional approach
2. Use LA implements an important rule in modern pedagogy: Give to each student the time he needs to solve any problem. In a student set, "*treat each on as an individual and not as a group*".

8. SUMMARIZING:

In the 10 last years, I used LA in data structure and Advanced Topics classes. The students are improver and happy with this approach. Average grades are better today as an opposite on 12 years back. The computer power is been used to enrich matters and topics in classroom.

This is more special and noble function for the computer, than just use it as a tool in presentations using power point.

9. REFERENCES

- [1] Baecker R. "Sorting out Sorting", sonorous and colored 30 min movie, Dynamic Graphics Project, University of Toronto, 1981.
- [2] Brown M. e Sedgewick R. "A system for algorithm Animation", Proceedings of ACM SIGGRAPH 1984, Minneapolis, July 1984, pp 177-186.

- [3] Byrne M, Catrambone R. e STASKO J. "Do algorithms animations aid learning?" Technical Report GIT-GVU-96-18, GVU Center, Georgia Institute of Technology, Atlanta, august 1996.
- [4] Haajanen, M. Pesonius, E. Sutinen, J. Tarhio, T. Teräsvirta, P. Vanninen: Animation of user algorithms on the Web. In: Proc. VL '97, IEEE Symposium on Visual Languages, IEEE 1997, 360-367. <http://www.cs.helsinki.fi/research/aaps/Jeliot/>
- [5] Hill T. "Assessing the instructional value of students predictions in tree animations", SIGCSE Doctoral Consortium Application, University of Mississippi, 2002.
- [6] Stasko J. "TANGO: A framework and system for Algorithm Animation", Computer, vol 23, no.9, September 1990, pp-27-39.
- [7] Wirth, Niklaus. Algorithms and Data Structures. Prentice Hall. 1986.

1 Listagem dos exercícios

A seguir uma descrição de todos os exercícios. Para aqueles exercícios que consultam uma base de dados para gerar cada exemplar, lista-se a base completa contendo todas as propostas, com as respectivas respostas. Esta é a versão de 1/ 1/2026.

001a - Programar é...

Uma apresentação rústica do que a programação é. Mostra a máquina de Turing e a arquitetura de Von Neumann. Daí mostra os comandos básicos de programação e pede para o aluno simular 4 algoritmos numéricos. Concluído em 11/10/20 com a sequência 648.

002b - Matemática Básica

6 conjuntos de 6 operações simples de aritmética básica envolvendo soma, subtração, multiplicação, divisão, chão e teto. O aluno deve exercer algumas contagens ao final. Concluído em 10/11/12 com a sequência 361.

002c - Funções inventadas

Expressões aritméticas simples envolvendo 3 funções inventadas especialmente para este exercício. São elas $\oplus$, $\ominus$ e $\boxplus$ que tem comportamento simples e que são usadas junto com $+$, $-$, $\times$, $\div$, $\lfloor$ e $\lceil$. Ao final o aluno deve fazer algumas contagens. Concluído em 10/11/12 com a sequência 362.

002d - Funções inventadas2

Expressões aritméticas simples envolvendo 3 funções inventadas especialmente para este exercício. São elas ∇ , ρ e $\diamond$ que tem comportamento simples e que são usadas junto com $+$, $-$, $\times$, $\div$, $\lfloor$ e $\lceil$. Ao final o aluno deve fazer algumas contagens. Concluído em 11/11/12 com a sequência 363.

002e - Funções inventadas3

Expressões aritméticas simples envolvendo 3 funções inventadas especialmente para este exercício. São elas $\uparrow$, $\downarrow$ e $\nexists$ que tem comportamento simples e que são usadas junto com $+$, $-$, $\times$, $\div$, $\lfloor$ e $\lceil$. Ao final o aluno deve fazer algumas contagens. Concluído em 12/11/12 com a sequência 364.

002f - Funções inventadas4

Expressões aritméticas simples envolvendo 3 funções inventadas especialmente para este exercício. São elas $\wr$, $\sim$ e $\top$ que tem comportamento simples e que são usadas junto com $+$, $-$, $\times$, $\div$, $\lfloor$ e $\lceil$. Ao final o aluno deve fazer algumas contagens. Concluído em 13/11/12 com a sequência 365.

002g - Funções inventadas5

Expressões aritméticas simples envolvendo 3 funções inventadas especialmente para este exercício. São elas $\P$, $\clubsuit$ e $\heartsuit$ que tem comportamento simples e que são usadas junto com $+$, $-$, $\times$, $\div$, $\lfloor$ e $\lceil$. Ao final o aluno deve fazer algumas contagens. Concluído em 26/11/12 com a sequência 367.

003a - Algoritmo da raiz quadrada

Implementa o algoritmo (antigamente aprendido no início do ciclo fundamental, hoje não mais) que permite calcular a raiz quadrada. É um processo razoavelmente complexo e ele permite vivenciar exatamente o que um algoritmo faz. Concluído em 12/11/07 com a sequência 191.

004a - Introdução a algoritmos

Apresenta os conceitos de divisão inteira e resto e a seguir pede para analisar o algoritmo de mdc (máximo divisor comum) devido a Euclides (aprox 400aC). Além disso, o exercício pede um algoritmo de uma tarefa trivial do dia a dia do aluno. Concluído em 20/12/05 com a sequência 144.

005a - Matemática Básica

Apresenta 50 exercícios simples de matemática envolvendo as 4 operações, teto, chão e resto de. O aluno deve resolvê-las de 5 em 5, ordenar os resultados e indicar qual expressão ocupa o lugar médio nessa ordenação. Concluído em 29/09/12 com a sequência 355.

006a - Jogo da vida

Um exercício simples, no qual o Jogo da Vida (Conway, 1970) é apresentado e são feitos comentários sobre a sua importância na Ciência da Computação. As 4 regras do jogo são apresentadas e depois é dado um tabuleiro 10×10 . Pede-se ao aluno que simule 4 gerações e informe depois como ficaram 4 células do tabuleiro especialmente escolhidas. O objetivo do exercício é divertir e depois trabalhar ainda informalmente os conceitos de: matriz, algoritmo, iteração, entre outros. Concluído em 23/10/07 com a sequência 189.

007a - GPS: Global Positioning System

Este exercício começa descrevendo o acidente do avião VARIG 254 que se perdeu no Mato Grosso achando estar no Pará. A seguir o princípio de funcionamento dos sistemas GPSs é mostrado. Uma simulação em 2D é pedida ao aluno, usando-se características da física e operações da geometria analítica. Concluído em 02/01/06 com a sequência 147.

008a - Problemas introdutórios

Apresenta 2 questões comuns sobre regras de nomes de variáveis e tipos de variável a usar em determinados algoritmos. Depois gera uma coleção de 10 exercícios matemáticos individuais sobre raciocínio. A resposta é sempre um número. Concluído em 29/08/06 com a sequência 164.

009b - Achar um número em uma sequencia

São mostradas 30 sequencias de 11 números cada. A lei de formação não está explicitada e deve ser descoberta pelo aluno. Um dos 11 números está faltando e o aluno precisa descobrir qual é. Os números buscados são somados de 3 em 3 para facilitar a correção. Este exercício nasceu de uma inspiração obtida olhando os exercícios da Olimpíada Brasileira de Informática. Concluído em 12/12/07 com a sequência 192.

009d - Talento de D. Mariquinha

Apresenta um conjunto de 7 regras lógicas que se devem aplicar cumulativamente sobre 10 pontos em uma matriz numérica. Cabe ao aluno fazer uma contagem final observando quais pontos obedecem a uma de 3 regras emitidas. Treina o sentido de observação e de seguir regras lógicas. Concluído em 02/03/12 com a sequência 319.

010a - Geografia Terrestre

Dois exercícios: no primeiro, dois pontos próximos ao equador são dados através de suas latitudes e longitudes e o exercício pergunta qual a distância em quilômetros entre eles. No segundo, o fato central do livro “Os filhos do capitão Grant” é descrito e os alunos convidados a descobrir qual o comprimento da busca do naufrago, a partir da recuperação de um bilhete de socorro, contendo uma latitude fornecida. Concluído em 18/11/08 com a sequência 225.

011b - Matemática e lógica básicas

60 exercícios (30 de matemática e 30 de lógica) são apresentados ao aluno. Ele deve achar 10 respostas (cada grupo de 6 exercícios tem suas respostas somadas. Isto facilita a correção). Concluído em 01/09/06 com a sequência 165.

011c - Matemática e lógica básicas (formato C)

60 exercícios (30 de matemática e 30 de lógica) no formato da linguagem C++ são apresentados ao aluno. Ele deve achar 10 respostas (cada grupo de 6 exercícios tem suas respostas somadas. Isto facilita a correção). Concluído em 12/03/18 com a sequência 570.

012a - Problemas envolvendo inteiros

7 problemas distintos, a saber:

- Ciclo na geração de números palíndromos
- Cálculo da função de Ackerman
- Achar o número “amigo” de um número dado
- Dizer se um número dado é ou não é um número de Armstrong
- Dizer se um número dado é ou não é um número de Wodall
- Dizer se um número dado é ou não é um número vampiro
- Dado um número dizer se ele é um dos elementos de um par de gêmeos.

Concluído em 31/01/09 com a sequência 234.

013a - Mais Problemas envolvendo inteiros

10 problemas distintos, a saber:

- Obter o número de complexidade de um número
- Cálculo da função de Cullen
- Achar o número Euclideano de um número dado
- Dado P , achar n , x e y no âmbito do Teorema de Euler conhecido como $6n + 1$.
- Dizer se um número dado é ou não é um Primo Fatorial

- Achar x , y e z em uma expressão cuja soma é 100
- Dado um número dizer se ele é ou não um número de Harshad
- Dizer se um número dado é ou não é um número Heptagonal
- Dizer se dois números dados são ou não são Homogêneos
- Dizer se um número dado é ou não é um Primo Bom.

Concluído em 05/02/09 com a sequência 235.

014a - Mais problemas de inteiros

Uma nova lista de problemas nos quais o aluno precisa exercer algum desempenho em programação. Alguns deles podem ser resolvidos com papel e lápis, mas outros exigem implementação. Concluído em 01/07/09 com a sequência 242.

015b - Comandos SE simples

Uma sequencia de 7 comandos SE bastante simples, acompanhados do fluxograma correspondente a cada comando. O aluno deve seguir o código (e o fluxograma) e encontrar a resposta pedida. Concluído em 08/02/11 com a sequência 272.

015d - Comandos SE simples (apenas código)

Uma sequencia de 15 comandos SE bastante simples, iguais (em estrutura) aos do exercício 015a, acompanhados de um esquema cômodo para fazer o teste de mesa. O aluno deve seguir o código e encontrar a resposta pedida. Concluído em 10/02/11 com a sequência 273.

015j - 27 comandos condicionais simples

Este exercício mostra ao aluno 27 comandos condicionais (SE) simples. Cada comando gera um resultado e eles devem ser somados 3 a 3, gerando 9 resultados para facilitar a correção. Concluído em 12/03/09 com a sequência 236.

016b - 15 comandos condicionais

Este exercício mostra ao aluno 15 comandos condicionais (SE). Cada comando gera um resultado e eles devem ser somados 3 a 3, gerando 5 resultados para facilitar a correção. Concluído em 13/03/09 com a sequência 237.

016h - Comandos SE aninhados

Apresenta 9 ninhos de SE e pede que o aluno calcule o valor de uma expressão que depende de todas as condicionais. Não é um exercício trivial. Concluído em 05/12/14 com a sequência 405.

017b - SEs compostos e encadeados

6 exercícios (funções) nos quais 6 valores numéricos aleatórios são fornecidos no início. A seguir uma série de comandos alternativos (SE) tanto compostos como encadeados, usando os conectivos $\wedge$ (e), $\vee$ (ou) e $\sim$ (não). Depois pede-se o resultado da soma de 5 variáveis. Concluído em 28/01/06 com a sequência 148.

017t - Working Path em algoritmos

Conceitua working path e apresenta 6 algoritmos condicionais pedindo que o aluno calcule o WP em cada um deles. Concluído em 23/01/14 com a sequência 404.

018f - Pseudocódigo X Fluxograma

8 trechos de programa são apresentados através de um fluxograma e através do pseudocódigo equivalente. Entretanto, em alguns desses 8 trechos ambos são equivalentes e em alguns outros trechos houve uma sutil alteração ou no fluxograma ou no pseudocódigo de maneira que ambos deixaram de ser equivalentes. O aluno deve investigar quais dos 8 trechos são efetivamente equivalentes. Concluído em 05/05/07 com a sequência 176.

018n - Pseudocódigo X Diagramas de Nassi-Schneidermann

8 trechos de programa são apresentados através de um diagrama de Nassi-Schneidermann e através do pseudocódigo equivalente. Entretanto, em alguns desses 8 trechos ambos são equivalentes e em alguns outros trechos houve uma sutil alteração ou no diagrama ou no pseudocódigo de maneira que ambos deixaram de ser equivalentes. O aluno deve investigar quais dos 8 trechos são efetivamente equivalentes. Concluído em 05/05/07 com a sequência 177.

019a - 7 algoritmos para seguir

7 problemas usuais são apresentados (qualidade em uma linha de produção; calculo do prêmio a funcionários; calculo de uma taxa de juros bancária; preço de um comercial de TV; verificação se uma decolagem é ou não é possível; uma brincadeira com quatro letras e o calculo de um imposto devido). Em cada um destes problemas os dados de entrada e o algoritmo (exclusivamente com comandos de atribuição e estruturas SE) são dados e cabe ao aluno achar a resposta que o algoritmo achará. Concluído em 06/05/09 com a sequência 241.

020a - Comando SE complexidade crescente

O comando SE e suas encarnações de complexidade crescente. Primeiro só uma condição simples. Depois uma condição composta. Depois o comando com então e senão. Depois, um se dentro de outro se e assim sucessivamente. São 11 comandos que o aluno deve responder. Concluído em 17/04/12 com a sequência 326.

021a - Chines de funções simples

São apresentadas 5 funções de mesma estrutura com até 3 níveis de IF's encadeados. Não há ainda estruturas de repetição. As funções inicializam as variáveis A, B e C no começo, e ao final, o exercício pergunta qual o valor de uma delas (que seguramente mudou de valor durante o processamento). Concluído em 03/04/06 com a sequência 154.

022a - Diversos SEs

Sugere manipulações lógicas em condições de comandos SE e pergunta sobre equivalências e negações. Usado na disciplina de lógica matemática, embora tenha um bom pé na programação. Concluído em 01/04/12 com a sequência 344.

023a - Enquanto e para básicos

Apresenta 7 exercícios simples de algoritmos envolvendo os comandos enquanto, para e repita (este último apenas é apresentado). Os comandos integram entre si e com algum comando condicional (se... fimse). Concluído em 08/10/12 com a sequência 356.

024b - Dígitos verificadores

Mostra-se a importância dos DVs e depois mostra-se os algoritmos de cálculo do CPF e do CNPJ. O aluno deve calcular os DVs de 2 CPFs e de 2 CNPJs. Concluído em 04/02/01 com a sequência 069.

024e - Boleto Bancário

Mostra as normas do BACEN e da FEBRABAN para construir um boleto bancário padrão e mostra os algoritmos para cálculo do fator de vencimento e dos 4 dígitos verificadores. Pede para o aluno calcular um de cada. Concluído em 05/01/15 com a sequência 462.

025a - Aritmetica complexa

Descreve a origem e a lógica de funcionamento dos números complexos. Pede que o aluno ache 4 raízes de um número complexo. Concluído em 12/01/24 com a sequência 673.

026a - Aritmetica inteira básica

Trabalha algumas características dos inteiros: primalidade, redução ao absurdo, teorema fundamental da aritmética, mmc e mdc, algoritmo de Euclides, critérios de divisibilidade e operações com frações. Concluído em 20/01/24 com a sequência 675.

027a - Aritmética não decimal

É mostrada uma generalização da aritmética para bases não decimais, variando entre base = 3 e base = 33, excetuando-se as bases 2, 10 e 16. Conversões, somas e subtrações nestas bases são pedidas. Concluído em 28/11/01 com a sequência 077.

028b - Condicionais aninhadas

Apresenta uma série de 3 comandos condicionais com 11 alternativas de distribuição de 7 comandos imperativos (sempre os mesmos) entre as diversas localizações de *else* e *endif*. Os exercícios são apresentados sem nenhum tipo de identificação. O aluno precisa resolver o valor apresentado pelo código ao final. Concluído em 11/01/18 com a sequência 566.

028e - Condicionais aninhadas e compostas

Apresenta uma série de 3 comandos condicionais e compostos. Os exercícios são apresentados sem nenhum tipo de identificação. O aluno precisa resolver o valor apresentado pelo código ao final. Concluído em 12/01/18 com a sequência 567.

028k - Condicionais aninhadas e compostas II

Apresenta uma série de 4 comandos condicionais e compostos. Os exercícios são apresentados sem nenhum tipo de identificação. O aluno precisa resolver o valor apresentado pelo código ao final. Concluído em 20/01/18 com a sequência 568.

028n - Condicionais aninhadas e compostas III

Apresenta uma série de 5 comandos condicionais e compostos. Os exercícios são apresentados sem nenhum tipo de identificação. O aluno precisa resolver o valor apresentado pelo código ao final. Concluído em 22/01/18 com a sequência 569.

029a - Condicionais Literais

14 conjuntos de comandos condicionais simples, compostos e aninhados são apresentados e o aluno deve descobrir o que ocorre. Note-se que tanto as condições quanto às ações são apenas literais e não envolvem matemática. Concluído em 23/02/13 com a sequência 373.

030a - Integridade de funções

9 erros possíveis de cometer ao desenhar a integração entre função chamadora e chamada dentro da linguagem C. Concluído em 15/07/13 com a sequência 389.

031a - Cálculo de calendários

Mostra o algoritmo de Aloysius Lilius e Christopher Clavius do século XVI para calcular o Domingo da Páscoa, do qual decorrem todos os feriados móveis de nosso calendário: Carnaval, Sexta da Paixão e Corpus Christi. Além de mostrar a distribuição de bissextos, o exercício mostra o cálculo de qualquer dia da semana. Finalmente, o exercício sugere um ano hipotético (entre 1600 e 2400) e pede que o aluno calcule o dia da semana de um dia qualquer e os 3 feriados móveis deste ano. Concluído em 25/02/02 com a sequência 088.

034a - Manipulação de Datas

Somas e subtrações envolvendo datas gregorianas. Ao aluno cabe resolver três expressões deste tipo. Concluído em 08/11/12 com a sequência 360.

035b - Localização de um oleoduto

9 poços de petróleo são dispostos sobre uma grade de 100km × 100km. Pede-se que o aluno calcule de que maneira ligá-los através de tubos de maneira a minimizar a quantidade de tubos usados. Exercício extraído do livro do Cormen, pág 155 da edição brasileira. Concluído em 27/12/02 com a sequência 095.

035c - Exponenciação, diagonais e treliças

Apresenta três problemas difíceis para serem resolvidos com papel e lapis. Os três extraídos do portal eulerproject.net. São os problemas 122, 28 e 15). Concluído em 29/04/14 com a sequência 410.

036b - Aritmética Decimal, Binária e Hexadecimal

O funcionamento interno do computador no que diz respeito à aritmética e as operações de adição e conversão entre os 3 sistemas principais: decimal, binário e hexadecimal. Concluído em 13/07/11 com a sequência 284.

036f - Fechamento em bases não decimais

Um resumo deste tema com 60 exercícios de conversões, somas e subtrações com bases entre 2 e 36. Concluído em 07/06/13 com a sequência 384.

037a - Base64, funcionamento e conversoes

O esquema de codificação chamado Base64 é apresentado e pede-se que o aluno faça 6 conversoes, 3 delas em cada sentido. Concluído em 07/11/12 com a sequência 359.

038a - Sequências de numeração

Conta-se de 1 até 60 em inúmeros sistemas de numeração diferentes. Daí o aluno é convidado a fazer o mesmo em um sistema inédito e depois 3 adições corretas são mostradas, cabendo ao aluno descobrir em que sistema de numeração elas ocorreram. Concluído em 28/12/12 com a sequência 369.

039a - Um ambiente estimulante: Bitcoin

Descreve o ambiente Bitcoin e algumas de suas contribuições à Ciência da Computação. É a deixa para introduzir o sistema de numeração de base 58. Ao final o aluno tem que fazer 6 conversões e 6 operações de adição e subtração neste sistema. Concluído em 10/05/24 com a sequência 699.

040a - Comandos de repetição simples

Uma sequência de 11 comandos de repetição bastante simples. O aluno deve seguir o código e encontrar a resposta pedida. Eventualmente pode programar os algoritmos em VisualG DEPOIS de resolvê-los, a fim de confirmar a resposta. Concluído em 11/02/11 com a sequência 274.

041a - 21 comandos de repetição

São gerados 21 comandos de repetição aleatórios (enquanto, para e repita), de um grupo de 6 esqueletos. Os resultados devem ser somados 3 a 3 para facilitar a correção. Concluído em 26/03/09 com a sequência 239.

042a - 14 comandos de repetição

São gerados 14 comandos de repetição aleatórios (enquanto, para e repita), de um grupo de 5 esqueletos. A complexidade é um pouco maior do que os exercícios do tipo 041, acima descritos. Os resultados devem ser somados 3 a 3 para facilitar a correção. Concluído em 27/03/09 com a sequência 240.

043c - O que faz o código em C++

Um número indefinido (entre 10 e 20) funções em C++ são apresentadas. O foco está em estudar os laços de IFs encadeados alguns com e outros sem cláusula ELSE. O aluno precisa descobrir quanto cada função imprime. Concluído em 18/05/24 com a sequência 706.

043e - O que faz o código em pseudocódigo

Um número indefinido (entre 10 e 20) funções em pseudo-código são apresentadas. O foco está em estudar os laços de SEs encadeados alguns com e outros sem cláusula SENÃO. O aluno precisa descobrir quanto cada função imprime. Concluído em 18/05/24 com a sequência 705.

044c - Qual função faz o que - C++

Dá 10 definições de programas C++ e apresenta os programas C++ que as atendem, mas em uma ordem aleatória. O aluno tem que estabelecer a ordem certa. Concluído em 11/05/24 com a sequência 700.

044p - Qual função faz o que - Python

Dá 10 definições de programas Python e apresenta os programas que as atendem, mas em uma ordem aleatória. O aluno tem que estabelecer a ordem certa. Concluído em 15/05/24 com a sequência 704.

044z - Escreva 10 programas C++

Dá 10 definições de programas C++ e pede que o aluno codifique tais programas. Concluído em 11/05/24 com a sequência 701.

045a - Cálculo de π

Apresenta 4 métodos de cálculo de π : trigonometria simples, algoritmo de Arquimedes, Algoritmo de Chudnovsky e o algoritmo de Bailey, Borwein e Plouffe (BBP). Depois pede que o aluno ache 3 dígitos de π em posições alatórias entre 100..1000. Concluído em 21/01/24 com a sequência 677.

046b - Condicionais em C++

12 exercícios envolvendo condicionais simples e compostas em linguagem C++. O aluno deve achar o que é impresso ao final. Concluído em 15/03/18 com a sequência 571.

046d - Condicionais em C++ - II

6 exercícios envolvendo condicionais simples e compostas em linguagem C++. O aluno deve achar o que é impresso ao final. Concluído em 17/03/18 com a sequência 573.

046p - Condicionais em Python

16 exercícios envolvendo condicionais simples e compostas em linguagem Python. O aluno deve achar o que é impresso ao final e somar os valores de 4 em 4. Concluído em 16/03/18 com a sequência 572.

046q - Condicionais em Python - II

8 exercícios envolvendo condicionais simples e compostas em linguagem Python. O aluno deve achar o que é impresso ao final. Concluído em 17/03/18 com a sequência 574.

047a - Mais repetição

12 exercícios envolvendo enquanto, repita e para. Aqui o aluno é desafiado a dizer se um conjunto de dados de entrada e de saída é possível (repondendo 1) ou impossível (0). Ao final ele ainda deve converter o número binário resultante em número decimal. Concluído em 19/04/11 com a sequência 277.

048c - Repetição em C++

10 exercícios envolvendo repetição (usando while) e eventualmente condicionais. O aluno deve achar o que é impresso ao final. Concluído em 27/03/18 com a sequência 512.

048p - Repetição em Python

10 exercícios envolvendo repetição (usando while) e eventualmente condicionais. O aluno deve achar o que é impresso ao final. Concluído em 27/03/18 com a sequência 513.

049b - Primeiros passos em programação - C++

Apresenta 4 programas bastante simples e pede que o aluno construa outros 4 igualmente simples. Alguns resultados numéricos devem ser obtidos. Concluído em 04/08/19 com a sequência 630.

049d - Primeiros passos em programação II - C++

Apresenta 6 programas bastante simples e pede que o aluno construa outros 3 igualmente simples. Alguns resultados numéricos devem ser obtidos. Concluído em 04/08/19 com a sequência 631.

049p - Primeiros passos em programação-python

Apresenta 4 programas bastante simples e pede que o aluno construa outros 4 igualmente simples. Alguns resultados numéricos devem ser obtidos. Concluído em 01/08/19 com a sequência 630.

049q - Primeiros passos em programação II-python

Apresenta 6 programas bastante simples e pede que o aluno construa outros 3 igualmente simples. Alguns resultados numéricos devem ser obtidos. Concluído em 03/08/19 com a sequência 631.

050c - Gerador movimentos de xadrez (C++)

Descreve um tabuleiro de xadrez como uma matriz $c++\ 8 \times 8$ e pede que o aluno desenvolva funções $c++$ capazes de contar o número de jogadas possíveis de peão, cavalo, torre, bispo, dama e rei. Concluído em 06/12/19 com a sequência 606.

050p - Gerador movimentos de xadrez (python)

Descreve um tabuleiro de xadrez como uma matriz python 8×8 e pede que o aluno desenvolva funções Python capazes de contar o número de jogadas possíveis de peão, cavalo, torre, bispo, dama e rei. Concluído em 06/12/19 com a sequência 605.

051c - Prática de programação (inicial C++)

Manda escrever 5 programas quase triviais. Os dados estão transcritos na folha e também publicados para facilitar o corte-cole. Concluído em 24/03/19 com a sequência 611.

051d - Prática em programação

Descreve 10 exercícios de programação nos quais o aluno deve buscar extensos dados de entrada em uma mídia magnética. Concluído em 16/05/17 com a sequência 538.

051e - Prática em programação II

Dá 6 problemas e um arquivo contendo os dados de entrada para resolvê-los. Concluído em 03/08/18 com a sequência 595.

051p - Prática de programação (inicial Pyt)

Manda escrever 5 programas quase triviais. Os dados estão transcritos na folha e também publicados para facilitar o corte-cole. Concluído em 23/03/19 com a sequência 610.

052a - Prática em programação II

Descreve 4 problemas e pede que o aluno escreva os programas que resolvem tais problemas. Concluído em 26/05/17 com a sequência 540.

053a - Prática de Programação III

Apresenta 3 problemas nos quais o aluno deve decodificar uma definição em português e gerar código de programação conveniente para resolver os problemas dados. São eles: custo de um estacionamento, análise da correção de expressões parentizadas, busca de um padrão em um texto e achar a cor de um pixel no fractal de Mandelbrot. Concluído em 10/07/19 com a sequência 628.

054a - Kumon de algoritmos

Esta folha é para treinar os comandos básicos de algoritmos (se, enquanto, repita e para). São 17 exercícios simples, de apenas um único comando cada um deles. Concluído em 23/06/06 com a sequência 160.

055a - programação via arquivos

4 programas que devem ser construídos. O primeiro pede a análise de 50 tabuleiros de xadrez. O segundo deve calcular 50 expressões numéricas *ad hoc*. O terceiro, deve preencher 20 matrizes com as regras lá informadas. Finalmente, 20 sequências de fibonacci também com regras *ad hoc*. Concluído em 13/11/18 com a sequência 599.

056a - Introdução à Programação

Apresenta 3 problemas diretamente retirados da OBI 2018, na qual o aluno deve resolver o problema das figurinhas da copa, geração de cápsulas e movimentação de cargas. Concluído em 13/07/19 com a sequência 629.

057a - Geração de sequências

Neste exercício são apresentadas 8 funções que contêm laços (loops). Na parte final de cada laço há a impressão de uma variável. O exercício mostra os primeiros 4 valores impressos e pede que o aluno ache o oitavo elemento impresso. timo exercício para ser implementado. Concluído em 30/12/05 com a sequência 145.

058a - Distribuição de cadeiras no legislativo

Apresenta três algoritmos de alocação de cadeiras em eleições proporcionais. São eles o método D'Hondt, Sainte-Lague e o definido pelo Código Eleitoral Brasileiro. O aluno deve calcular três eleições, uma em cada método. Concluído em 18/04/14 com a sequência 409.

059a - Introdução à programação - Euler

Apresenta 10 tarefas fortemente baseadas (mais simples) no projeto Euler. São os problemas 1, 2, 3, 7, 8, 10, 12, 14, 21 e 11 do referido site. Concluído em 16/08/19 com a sequência 632.

061a - Grupos algébricos

Descreve as propriedades comutativa, associativa, fechamento, identidade e inverso em estruturas algébricas. Depois descreve o sistema relógio-12 e depois a aritmética modular. Termina definindo grupos algébricos. Ao final o aluno tem que resolver 2 expressões, 2 equações e dois cálculos envolvendo nosso calendário. Concluído em 27/01/24 com a sequência 680.

062a - Engenharia reversa de algoritmos

Este exercício mostra 4 funções. Em cada uma há 3 variáveis, cujos valores são inicializados no começo. Mas, apenas 2 destes valores são mostrados ao aluno, cabendo a ele, descobrir qual a inicialização da terceira variável a partir do comportamento da função e dos resultados que ela gera ao ser executada. Concluído em 31/01/06 com a sequência 149.

063a - Equações e inequações

Descreve as inequações envolvendo a função modular. Trata também de sistemas lineares de inequações. O aluno deve resolver 5 exercícios. Concluído em 03/02/24 com a sequência 681.

064a - Treino em estruturas de repetição

Exercícios que colocam 2 estruturas de repetição (enquanto, repita e para) sendo uma dentro da outra. A folha apresenta 7 funções todas elas com 2 repetições. Ao final, pede-se que o aluno simule a função e informe o que cada uma devolve. Este exercício é mais complexo que o 54 e menos complexo que o 065. Concluído em 26/06/08 com a sequência 220.

065a - Funções envolvendo se, enquanto e repita

Este exercício gera 4 funções razoavelmente complexas, nas quais se misturam comandos se, enquanto e repita, além de um conjunto de cerca de 15 variáveis em cada função. O aluno deve seguir o fluxo e informar ao final qual o valor que será impresso. É um convite à implementação. Concluído em 20/04/06 com a sequência 156.

066a - Geometrias não euclidianas

Apresenta o postulado das paralelas de Euclides e pergunta o que acontece se ele for negado: surgem as geometrias Lobachevskiana, Riemanniana e do motorista de taxi. O exercicio pede um cálculo nesta última. Concluído em 25/01/24 com a sequência 679.

067a - Siga o código: repetições – pseudocódigo

Um número indefinido (entre 10 e 20) de funções em pseudo-código são apresentadas. O foco está em estudar os laços de repetição (enquanto e para). O aluno precisa descobrir quanto cada função imprime. Concluído em 27/05/24 com a sequência 709.

069a - Chines de algoritmos II

4 funções geradas aleatoriamente, mas que garantidamente executam em tempo finito são mostradas. Cada função recebe 5 valores mostrados. Pede-se que o aluno calcule e informe, ao final da execução de cada função, o valor da soma das 5 variáveis originais. Concluído em 29/02/00 com a sequência 005.

- Genérico de construção de algoritmos

Este exercicio comporta uma base de dados de exercicios para uso em muitas aulas de algoritmos. Para cada uma das aulas, a base contém cerca de 10 exercicios distintos. Ao gerar as folhas para os alunos, cada um deles receberá 4 (5 ou 6, depende da aula) de exercicios distintos. Todos eles exigem implementação nas aulas de laboratório. Concluído em com a sequência .

071c - Programas C com parâmetros

Pede que o aluno construa 12 programas simples em C todos eles recebendo parâmetros na linha de comando. Concluído em 20/05/14 com a sequência 412.

072b - Iniciando na programação

Três programas básicos são apresentados (soma de divisores, quantidade de primos dentro de um intervalo e soma dos pares da sequência de Fibonacci). Os algoritmos são mostrados e o aluno deve implementá-los em alguma linguagem. Um resultado numérico de cada um é pedido. Concluído em 30/05/24 com a sequência 710.

072d - Iniciando na programacao II

Da 3 algoritmos e pede que o aluno os implante e ache um resultado numerico esperado. Concluído em 30/05/24 com a sequência 718.

072e - Iniciando na programacao III

Da 3 algoritmos e pede que o aluno os implante e ache um resultado numerico esperado. Baseado em problemas do portal Euler. Concluído em 14/06/24 com a sequência 719.

072g - Iniciando na programacao IV

Mostra o Euler500 (quantos números com 8 divisores há ?) e o Euler4 (quantos números palindromos há ?) e pede que os alunos implementem os algoritmos citados. Concluído em 21/06/24 com a sequência 722.

072j - Iniciando na programação V

Mostra o Euler18, 21 e 28. Depois pede que os alunos implementem os algoritmos citados. Concluído em 03/07/24 com a sequência 727.

072k - Iniciando na programação VI

Mostra o Euler80 e Euler 25. Depois pede que os alunos implementem os algoritmos citados. Concluído em 06/07/24 com a sequência 728.

073a - Achar o 10º termo de uma sequência

Este exercicio mostra 20 sequências de 7 números cada. Cada uma é resultado de um ciclo *para* no qual estão envolvidas 1, 2 ou 3 variáveis, cujo valor é desconhecido. Ao aluno cabe:

- sugerir valores para x , y e z
- verificar se os valores sugeridos geram a sequência dada
- corrigir – se necessário – e voltar a testar
- após o acerto, gerar e responder o 10º termo.

Concluído em 04/06/06 com a sequência 158.

075a - Desenhando funções

Descreve como obter facilmente o gráfico de uma função. A ênfase é em Python abordando também Maple, Octave e também o pacote simbólico sympy. Pede para o aluno gerar um gráfico simples de um polinômio dado. Concluído em 13/02/24 com a sequência 684.

076b - Leitura de dados pré-existentes em C

Baseado no exercicio **p76** pede aos alunos que leiam 3 arquivos em C e processem os dados. Concluído em 12/08/14 com a sequência 415.

076c - Leitura de dados pré-existentes em Python

Baseado no exercicio **p76** pede aos alunos que leiam 3 arquivos em Python e processem os dados. Concluído em 28/09/17 com a sequência 553.

077a - Chines de algoritmos

4 funções usuais na informática são mostradas: busca em cadeias, médias em um vetor numérico, ordenação por inserção e soma de colunas em uma matriz. Dados de entrada são fornecidos, junto com o código de cada função pedindo-se ao aluno que infira os resultados que serão produzidos. Concluído em 21/02/00 com a sequência 004.

078a - Autovalores e autovetores

Descreve a teoria da Álgebra Linear associada a autovalores e autovetores mostrando os algoritmos básicos para o aluno calcular e depois mostra onde buscar ajuda no mundo computacional. O aluno terá que resolver à mão 2 exercicios pequenos (matrizes 2×2). Concluído em 08/02/24 com a sequência 682.

079a - Aplicações da Álgebra Linear

Mostra 2 aplicações: o cálculo de um elemento da Sequência de Fibonacci sem a necessidade de calcular os elementos anteriores usando a matriz de autovalores e a contabilização de caminhos entre 2 nodos em um grafo usando operações de multiplicação matricial. Concluído em 10/02/24 com a sequência 683.

080a - Introdução a Python

Começa a descrever o Python com 3 tipos de comando: E/S, atribuição e comandos aritméticos. Concluído em 05/10/22 com a sequência 676.

082a - Algoritmo de Intercalação

Descreve-se preliminarmente o conceito de ordenação e depois com mais detalhe o de intercalação. O algoritmo é apresentado, com seus comandos devidamente numerados. O objetivo do exercicio é também praticar a execução de testes de mesa e o que aluno deve fazer é, dados dois conjuntos ordenados, intercalá-los e informar quais comandos do algoritmo foram executados e em que ordem. Concluído em 30/12/09 com a sequência 254.

084a - Balance Line

O algoritmo de balance line é mostrado e é discutida a sua importância para atualização em lote de grandes volumes de dados. O exercicio oferece dois arquivos de 30 registros cada (o arquivo de cadastro antigo e o arquivo de movimentações), pedindo-se ao aluno que gere o novo cadastro atualizado bem como o relatório de incompatibilidades da atualização. Perguntas são feitas sobre o conteúdo dessas duas saídas. Concluído em 02/03/00 com a sequência 022.

086a - Algoritmo de quebras

Apresenta-se o algoritmo de quebras em relatórios. Um exercicio é proposto e os alunos devem resolvê-lo. Entendido o conceito, este exercicio é muito fácil. Ele exige também a habilidade de ordenar um conjunto de dados. Concluído em 15/01/10 com a sequência 256.

088a - Sequências e séries

Mostra como calcular uma função em um ponto usando as séries de Taylor e Maclaurin. Depois pede que o aluno calcule 3 funções. Concluído em 24/02/24 com a sequência 686.

089a - Otimização de funções

Descreve a teoria por trás da otimização de funções e como as ferramentas de cálculo simbólico podem ser usadas para resolver problemas deste tipo. O aluno recebe 3 problemas aleatórios e diferentes para resolver. Concluído em 17/02/24 com a sequência 685.

090a - se...então: proposição X comando

Compara o conceito do conectivo proposicional se...então com o comando condicional se...então e pede a resolução de 10 exercícios. Concluído em 27/02/17 com a sequência 533.

093a - Lógica Proposicional

Gera 64 exercícios baseados em no cap. 2 do livro Iniciação à Lógica Matemática de Edgar Alencar. Concluído em 21/01/12 com a sequência 314.

094a - Tabelas Verdade

Gera 40 exercícios baseados em no cap. 3 do livro Iniciação à Lógica Matemática de Edgar Alencar. Concluído em 24/01/12 com a sequência 315.

095b - Construção de Tabelas Verdade

Define proposição simples e composta e os 5 conectivos (não, e, ou, se...então e se e somente se... então e depois pede 8 tabelas verdade simples (4 com 2 variáveis e 4 com 3 variáveis). Concluído em 04/01/15 com a sequência 461.

095e - Tabelas Verdade

Gera 10 exercícios de construção de tabela verdade para 10 expressões aleatoriamente geradas. Os primeiros 4 exercícios são para expressões de 2 variáveis, os próximos 3 são para 3 variáveis e os últimos 3 para 4 variáveis. Concluído em 27/01/12 com a sequência 316.

095f - Tabelas Verdade (pós 095e)

Melhoramento da folha 095e (vide). Contempla alguns exemplos para serem feitos em conjunto na sala de aula. Concluído em 27/02/16 com a sequência 491.

095h - Prática em Tabelas Verdade

20 exercícios de construção de tabelas verdade. A resposta deve ser dada de maneira numérica. Concluído em 13/02/17 com a sequência 532.

096a - Semântica em Lógica Proposicional

Apresenta os conceitos de tautologia, contradição e contingência. Depois mostra *modus ponens* e *modus tolens* e finalmente mostra as Leis de De Morgan. A seguir pede 20 classificações ao aluno. Concluído em 21/03/12 com a sequência 322.

097a - Equivalência e Redução ao absurdo

Apresenta os conceitos de Equivalência de proposições, de redução ao absurdo e de conjunção e disjunção de negações. A seguir pede 20 classificações (certo ou errado) ao aluno. Concluído em 24/03/12 com a sequência 323.

098b - Álgebra das proposições - 1.parte

Explicações e teoria sobre álgebra das proposições, seguida de 2 demonstrações. Concluído em 01/05/13 com a sequência 381.

098f - Álgebra proposicional-2.parte

Demonstrações de lemas, teoremas e proposições dentro da lógica matemática. São feitos 4 exercícios junto com os alunos e depois estes devem provar 2 teoremas. Concluído em 21/04/12 com a sequência 327.

099b - Argumentos e validade

O conceito de argumento, suas premissas e conclusão. Conceito de validade, com diversas demonstrações e 2 exercícios para o aluno fazer. Concluído em 05/05/12 com a sequência 329.

099h - Argumentos e validade ad hoc

O conceito de argumento, suas premissas e conclusão. Usa um gerador de teoremas (ainda não suficientemente testado) mas que parece muito bom. Concluído em 12/05/12 com a sequência 331.

099j - Argumentos Literais e Validade

Mostra 4 argumentos lógicos e 6 argumentos lógicos literais e pede que o aluno demonstre a validade. Nos literais, antes da demonstração o aluno tem que passar do texto para o enunciado lógico. Concluído em 09/03/16 com a sequência 492.

100a - Demonstrações de argumentos

Apresenta 5 maneiras de provar argumentos lógicos e depois mostra 10 textos que devem ser passados para notação simbólica e depois provados ou não (50% de chance para cada caso). Concluído em 19/05/12 com a sequência 334.

101b - Lógica e Diagramas de Venn

Apresenta 3 problemas onde o aluno deve construir um diagrama de Venn para 10 conjuntos distintos a partir de 3 conjuntos originalmente dados. Concluído em 18/07/12 com a sequência 337.

101c - Teoria de Conjuntos e Diagramas de Venn

Apresenta 3 problemas onde o aluno deve construir um diagrama de Venn para 10 conjuntos distintos a partir de 3 conjuntos originalmente dados. Concluído em 25/08/14 com a sequência 418.

102a - Lógica e Quantificadores

Apresenta 10 frases em português e pede que o aluno as escreva em lógica de predicados. Depois dá 10 frases em lógica e as pede em português. Baseado no exercício 856. Concluído em 19/07/12 com a sequência 338.

103a - Demonstração automática de teoremas

Parecido com a família 857, mas com a vantagem dos teoremas serem gerados automaticamente, isto é inéditos sempre. O aluno tem 10 teoremas e deve indicar quais podem ser provados. A probabilidade é 50% para cada lado. Concluído em 19/07/12 com a sequência 339.

104a - Demonstração automática de teoremas

Um clone do exercício 858, adequado ao 1º ano do curso de EC Concluído em 10/08/12 com a sequência 342.

105a - Lógica e Teoria dos Conjuntos

É feita a analogia entre lógica e teoria dos conjuntos. Feito isso, dão-se 4 conjuntos A , B , C e D . Depois, mediante operações lógicas e de teoria dos conjuntos pedem-se 4 outros conjuntos P , Q , R e S . Daí pedem-se 4 características deste últimos conjuntos. Concluído em 27/12/12 com a sequência 368.

106a - Álgebra de Boole

Apresenta-se a Álgebra de Boole e suas regras principais. Dadas diversas expressões, o aluno deve construir um circuito elétrico que as satisfaçam. Em outro exercício deve construir a tabela verdade. E finalmente, no último exercício recebe a descrição de um problema e deve construir o circuito que o resolve. Concluído em 15/01/13 com a sequência 371.

108a - Outras lógicas

Apresentam-se as lógicas prabilística e nebulosa e pede-se que o aluno faça uma análise comparativa entre elas. Concluído em 18/01/13 com a sequência 372.

110c - Indexação e indireção

Dados 3 vetores de 16 valores numéricos cada e dadas 5 variáveis numéricas globais, o exercício pede que o aluno ache o resultado de 48 operações de indexação e de indireção, tomando por base o compilador C/C++ e portanto com origem dos índices em zero. Concluído em 18/02/01 com a sequência 070.

110v - Indexação e indireção

Dados 3 vetores de 16 valores numéricos cada e dadas 5 variáveis numéricas globais, o exercício pede que o aluno ache o resultado de 48 operações de indexação e de indireção, com origem dos índices em um. Concluído em 18/02/01 com a sequência 070.

111a - Cadeias como vetores de caracteres

Apresenta 9 códigos de funções que mostram como manipular cadeias na forma de vetores de caracteres. São eles: contagem de vogais, qual a letra mais frequente, upper e lower, abreviar um nome, achar sub-cadeias dentro de uma cadeia, justificar um texto, achar a maior palavra em uma frase, localizar pláindromos, verificar acerto de uma expressão parentizada e efetuar uma análise estilística. Para fazer juz à nota, o aluno deve implementar uma delas. Concluído em 26/07/08 com a sequência 223.

112a - Prática em vetores (freemat)

Descreve algumas funcionalidades do Freemat e pede para os alunos praticarem. Não há correção envolvida. Concluído em 19/05/16 com a sequência 499.

114a - Vetor, registro e matriz

Define vetor, registro e matriz e pede que o aluno faça 5 definições específicas. Todos os alunos recebem as mesmas perguntas. Concluído em 29/07/08 com a sequência 224.

116b - Manuseio de vetores - parte 1

Este exercício descreve o mecanismo dos vetores, suas operações de ordenação, inclusão, exclusão e busca. O exercício para o aluno é meramente folclórico (buscar o nome de 5 algoritmos de ordenação). Concluído em 07/08/06 com a sequência 161.

116c - Manuseio de vetores - parte 2

Na parte de busca, apresenta e compara 3 técnicas (busca linear, busca linear ordenada e busca linear com sentinela). Mostra também o algoritmo da busca binária. O aluno deve implementar os 3 algoritmos e inferir o desempenho deles nas 3 técnicas. Concluído em 20/05/11 com a sequência 281.

119a - Manuseio de tabelas II

São mostrados 5 algoritmos: inclusão em tabelas desordenadas, exclusão puxando os sobrantes, exclusão preenchendo com um indicador de exclusão, busca binária e inclusão em tabela ordenada. Por enquanto, a folha pede apenas o preenchimento de algumas complexidades. As folhas são todas iguais entre si. Concluído em 10/08/06 com a sequência 162.

120a - Prática em conjuntos

Dá 5 enunciados literais pedindo que o aluno construa os Diagramas de Venn correspondentes e a seguir responda perguntas sobre eles. Concluído em 30/03/15 com a sequência 470.

121p - Matrizes em Python

Pede para o aluno resolver 3 problemas envolvendo matrizes em Python (há versão para Freemat). Concluído em 06/07/16 com a sequência 501.

125b - Certificação de algoritmos básicos

Neste exercício são apresentados 5 algoritmos básicos (um número é primo?, soma dos divisores de um número, achar a potência de um número, validar uma data e alocar bolas de brinquedo). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos. Concluído em 11/07/07 com a sequência 179.

125c - Certificação de algoritmos de vetor

Neste exercício são apresentados 5 algoritmos de vetor (achar o maior elemento, onde está uma determinada chave, quantas vezes um elemento existe no vetor, calcular a média, moda e mediana e responder se o vetor está em ordem). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos. Concluído em 11/07/07 com a sequência 180.

125d - Certificação de algoritmos de vetor

Neste exercício são apresentados 5 algoritmos de vetor (ver se uma frase é palíndromo, incluir uma chave em um vetor, achar o número da maior palavra em uma frase, exclusão de todos os valores k no vetor V e achar a amplitude de um vetor). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos. Concluído em 12/07/07 com a sequência 181.

125e - Certificação de algoritmos de matriz

Neste exercício são apresentados 5 algoritmos de matriz (achar o vetor soma na vertical, calcular uma média ad-hoc na matriz, achar o vetor menor na horizontal, calcular o produto de duas matrizes e dada uma matriz com dados eleitorais, responder quem ganhou a eleição). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos. Concluído em 27/07/07 com a sequência 185.

128a - Matrizes

Apresenta 5 algoritmos de matrizes. Uma totalização nos dois sentidos, o cálculo da matriz transposta, uma matriz com elementos abaixo da diagonal principal zerados (um grafo não dirigido?), depois a multiplicação de 2 matrizes e finalmente a solução de um sistema de equações lineares. O algoritmo é mostrado e um exemplo completo está descrito na folha. Finalmente, o aluno é convidado a resolver (a mão ou usando o computador, a escolha é dele) um sistema com 8 equações e 8 incógnitas. Concluído em 22/08/06 com a sequência 163.

129a - Prática em matrizes

10 Exercícios práticos (exigindo implementação) envolvendo algoritmos de manipulação de matrizes. Concluído em 30/05/13 com a sequência 383.

131a - Conceito introdutório de árvore

O conceito intuitivo de árvore é apresentado e depois, 12 tipos de árvores são apresentadas. Comenta-se sobre árvore como suporte ao algoritmo (por exemplo no xadrez) e árvore como estrutura de dados para guardar informações (por exemplo em um índice de DB). Ao final, pede-se que o aluno desenvolva 3 árvores bem simples. Concluído em 23/07/08 com a sequência 222.

132a - Ordenações quadráticas

Uma primeira abordagem para os algoritmos de ordenação a ser usada ainda no primeiro ano (algoritmos 2). Concluído em 13/06/11 com a sequência 282.

135a - Manipulações no cubo Rubik

O cubo Rubik é apresentado e é sugerida uma estrutura de dados que suporte manipulações sobre ele. Depois uma lista de 9 operadores é também mostrada, aplicada à estrutura de dados recém definida. O aluno é convidado a aplicar dois operadores sobre 2 cubos aleatórios e depois deve resolver uma instância trivial do cubo (apenas 2 movimentos). Concluído em 21/09/06 com a sequência 170.

136b - Programação I

Desafios de programação baseados em provas/exercícios da UFPR. Acervo de 12 questões das quais 5 são aleatoriamente escolhidas por exercício. Concluído em 08/07/24 com a sequência 729.

136c - Programação II

Desafios de programação baseados em provas/exercícios da UFPR. Acervo de 12 questões das quais 5 são aleatoriamente escolhidas por exercício. Concluído em 08/07/24 com a sequência 730.

136d - Programação III

Desafios de programação baseados em provas/exercícios da UFPR. Acervo de 12 questões das quais 5 são aleatoriamente escolhidas por exercício. Concluído em 09/07/24 com a sequência 731.

136e - Programação IV

Desafios de programação baseados em provas/exercícios da UFPR. Acervo de 12 questões das quais 5 são aleatoriamente escolhidas por exercício. Concluído em 09/07/24 com a sequência 732.

136f - Programação V

Desafios de programação baseados em provas/exercícios da UFPR. Acervo de 12 questões das quais 5 são aleatoriamente escolhidas por exercício. Concluído em 09/07/24 com a sequência 733.

136g - Programação VI

Desafios de programação baseados em provas/exercícios da UFPR. Acervo de 12 questões das quais 5 são aleatoriamente escolhidas por exercício. Concluído em 09/07/24 com a sequência 734.

140a - Prática em eixos cartesianos

10 Exercícios simples envolvendo aplicação dos eixos cartesianos e noções primitivas de geometria analítica. Concluído em 28/06/13 com a sequência 387.

144a - Números romanos

Apresenta o sistema romano de numeração. Os algoritmos de conversão de arábico para romano e vice-versa são apresentados. Há 6 conversões em cada sentido para o aluno fazer. Deve-se alertar o fato de que na conversão de arábico → romano há várias respostas possíveis (todas corretas) devendo o aluno fazer aquela que é dada pelo algoritmo estudado. Isto atrapalha o uso de conversores livremente disponíveis para resolver o problema proposto. Concluído em 16/09/06 com a sequência 169.

145a - Completando algoritmos

4 algoritmos (primo, somatório de divisores, criação de uma árvore binária de pesquisa e mnemônico de número real) são apresentados incompletos, junto com execuções corretas. O aluno precisa completar os algoritmos (em Python) e depois executá-los em instâncias propostas. Concluído em 30/11/18 com a sequência 600.

146a - Completando algoritmos II

O algoritmo de Huffman devidamente apresentado em Python. O aluno deve descobrir a sequência de bits de um determinado caractere. Concluído em 03/12/18 com a sequência 601.

147a - Completando algoritmos III

Os algoritmos de mmc, mdc, multiplicação matricial, inclusão e exclusão em árvore binárias de pesquisa são devidamente apresentados em Python. O aluno deve processar um exemplo de cada algoritmo. Concluído em 08/12/18 com a sequência 602.

148a - Completando algoritmos IV

Apresenta o algoritmo da Árvore de Merkle e o conceito do blockchain na moeda virtual Bitcoin. Sugere um sistema de numeração de base 58 e pede que o aluno descubra se 4 mensagens fazem ou não parte do bloco de controle. Concluído em 26/12/18 com a sequência 603.

149a - Completando algoritmos V

Apresenta o algoritmo ingênuo de escalonamento de Gauss para a solução de sistemas lineares. Pede para o aluno resolver um sistema de 10 equações a 10 incógnitas. Depois mostra o algoritmo do Caixeiro Viajante e seu relaxamento usando o algoritmo guloso. O aluno deve resolver ambos para uma instância com 7 cidades. Concluído em 01/01/19 com a sequência 604.

150a - Desenvolvendo a lógica de programação

define 3 conceitos: árvore binária de pesquisa, matriz quadrada crescente e ponto de lombada e pede que o aluno ache uma instância de cada problema. Concluído em 02/11/19 com a sequência 639.

157b - Prática em matrizes em C++

Pede que o aluno construa 5 matrizes diversas em C++ e responda a 5 perguntas (as mesmas) sobre cada matriz. Concluído em 28/05/18 com a sequência 583.

157c - Prática em matrizes em C++ - II

Pede que o aluno construa 4 matrizes diversas em C++ e responda a 5 perguntas (as mesmas) sobre cada matriz. Concluído em 28/05/18 com a sequência 584.

157p - Prática em matrizes em Python

Pede que o aluno construa 5 matrizes diversas em Python e responda a 5 perguntas (as mesmas) sobre cada matriz. Concluído em 28/05/18 com a sequência 583.

157q - Prática em matrizes em Python - II

Pede que o aluno construa 4 matrizes diversas em Python e responda a 5 perguntas (as mesmas) sobre cada matriz. Concluído em 28/05/18 com a sequência 584.

157q - Prática em matrizes em Python - II

Pede que o aluno construa 4 matrizes diversas em Python e responda a 5 perguntas (as mesmas) sobre cada matriz. Concluído em 28/05/18 com a sequência 584.

158c - Matriz Inversa em C++

Mostra o algoritmo de Gauss Jordan do cálculo de uma matriz inversa e pede que o aluno o aplique a uma matriz dada de ordem 10, usando a linguagem C++. Concluído em 30/05/18 com a sequência 585.

158p - Matriz Inversa em Python

Mostra o algoritmo de Gauss Jordan do cálculo de uma matriz inversa e pede que o aluno o aplique a uma matriz dada de ordem 10, usando a linguagem Python. Concluído em 30/05/18 com a sequência 585.

159a - Matriz Inversa

Descreve o que é a matriz inversa em um sistema linear $A.x = B$, mostra suas propriedades, descreve 6 métodos para obtê-la e apresenta 2 usos importantes para ela. O aluno deve responder a 3 perguntas sobre o tema. Concluído em 30/08/18 com a sequência 596.

160a - Números Primos

Conta um pouco da história e importância dos números primos, com destaque para a Hipótese de Riemann. Pede que o aluno programe 9 funções baseadas no conceito de número primo. Concluído em 03/12/15 com a sequência 476.

162c - Otimização de Algoritmos em C++

Descreve e resolve o problema 12 de projecteuler.net. Mostra como o conhecimento de matemática e algoritmos pode reduzir o tempo de processamento em várias ordens de grandeza. No final, pede que o aluno faça uma implementação. Concluído em 31/05/18 com a sequência 586.

162p - Otimização de Algoritmos em Python

Descreve e resolve o problema 12 de projecteuler.net. Mostra como o conhecimento de matemática e algoritmos pode reduzir o tempo de processamento em várias ordens de grandeza. No final, pede que o aluno faça uma implementação. Concluído em 08/06/17 com a sequência 541.

170c - São equivalentes as condicionais? (C++)

Mostra 20 pares de condicionais e pergunta se são equivalentes. Concluído em 06/03/19 com a sequência 608.

170p - São equivalentes as condicionais? (Python)

Mostra 20 pares de condicionais e pergunta se são equivalentes. Concluído em 06/03/19 com a sequência 609.

175a - Varejão I - Pérolas de Programação

Mostra dois problemas: a ordenação de números complexos e o fractal de sierpinski e desafia os alunos a manipular os códigos python destes problemas. Concluído em 09/07/21 com a sequência 662.

183c - Truques de programação I - C++

Apresenta 3 problemas (despacho de um trem, distâncias em um pátio de containers e pontuação de mãos de poker) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 100 trens, 4 pátios e 1000 mãos. Concluído em 09/06/19 com a sequência 615.

183p - Truques de programação I - Python

Apresenta 3 problemas (despacho de um trem, distâncias em um pátio de containers e pontuação de mãos de poker) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 100 trens, 4 pátios e 1000 mãos. Concluído em 25/06/19 com a sequência 621.

184c - Truques de programação II - C++

Apresenta 3 problemas (números similares=têm os mesmos dígitos, presentes tipo kinder ovo e distâncias cartesianas) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 500 similares, 1000 presentes e 1000 pares de pontos. Concluído em 10/06/19 com a sequência 616.

184p - Truques de programação II - Python

Apresenta 3 problemas (números similares=têm os mesmos dígitos, presentes tipo kinder ovo e distâncias cartesianas) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 500 similares, 1000 presentes e 1000 pares de pontos. Concluído em 25/06/19 com a sequência 622.

185c - Truques de programação III - C++

Apresenta 3 problemas (números crescentes, números perfeitos, abundantes e deficientes e melhor localização de um supermercado) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 500 crescentes, 500 tipos de números e 50 supermercados. Concluído em 14/06/19 com a sequência 617.

185p - Truques de programação III - Python

Apresenta 3 problemas (números crescentes, números perfeitos, abundantes e deficientes e melhor localização de um supermercado) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 500 crescentes, 500 tipos de números e 50 supermercados. Concluído em 25/06/19 com a sequência 623.

186c - Truques de programação IV - C++

Apresenta 3 problemas (matrizes simétricas, vulcão no Havaí e sequências de Fibonacci genéricas) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 50 matrizes possivelmente simétricas, 50 vulcões e 200 sequências genéricas. Concluído em 14/06/19 com a sequência 618.

186p - Truques de programação IV - Python

Apresenta 3 problemas (matrizes simétricas, vulcão no Havaí e sequências de Fibonacci genéricas) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 50 matrizes possivelmente simétricas, 50 vulcões e 200 sequências genéricas. Concluído em 25/06/19 com a sequência 624.

187c - Truques de programação V - C++

Apresenta 3 problemas (manipulação de conjuntos, cálculo de determinantes e cálculo de matrizes inversas) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 40 grupos de conjuntos, 40 determinantes e 10 pares de matrizes inversas. Concluído em 15/06/19 com a sequência 619.

187p - Truques de programação V - Python

Apresenta 3 problemas (manipulação de conjuntos, cálculo de determinantes e cálculo de matrizes inversas) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 40 grupos de conjuntos, 40 determinantes e 10 pares de matrizes inversas. Concluído em 25/06/19 com a sequência 625.

188c - Truques de programação VI - C++

Apresenta 3 problemas (localização do ponto central, construção de um histograma e manipulação de um tabuleiro de xadrez) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 50 grupos de pontos, 50 histogramas e 100 tabuleiros de xadrez. Concluído em 16/06/19 com a sequência 620.

188p - Truques de programação VI - Python

Apresenta 3 problemas (localização do ponto central, construção de um histograma e manipulação de um tabuleiro de xadrez) e mostra para cada um um truque de programação. O aluno precisa processar muitas instâncias de cada um destes problemas: 50 grupos de pontos, 50 histogramas e 100 tabuleiros de xadrez. Concluído em 27/06/19 com a sequência 626.

189c - Truques de programação VII - OBI

Apresenta 3 problemas (Bolas, Cinco e Muro, todos retirados do site da OBI). O aluno precisa processar muitas instâncias de cada um destes problemas. Não há soluções em C++ pelo que qualquer linguagem pode ser usada. Concluído em 09/07/19 com a sequência 627.

190a - 10 problemas a base de papel e lápis

10 problemas baseados na prova da OBI de 2003. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. Concluído em 17/12/07 com a sequência 193.

191b - 5 problemas a base de papel e lápis-a

5 problemas baseados na prova da OBI de 2004. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: supermercado(2), maionese(1) e mapas da prefeitura(2). Concluído em 18/12/07 com a sequência 194.

191k - 5 problemas a base de papel e lápis-k

10 problemas baseados na prova da OBI de 2004. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Atendimento ao consumidor(2), Clara e Luiz (1) e Economia do transporte(2). Concluído em 18/12/07 com a sequência 195.

192a - 10 problemas a base de papel e lápis

10 problemas baseados na prova da OBI de 2005. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Equipe de software (2), Flores (2), Pane seca (1) Linguagem de programação (2), Semana de provas (1) e Futebol(2). Concluído em 19/12/07 com a sequência 196.

193b - 7 problemas a base de papel e lápis-a

7 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Excursão dos alunos(2), Conjunto de rock (2), azulejos (1) e transporte de alunos (2). Concluído em 22/12/07 com a sequência 197.

193c - 6 problemas a base de papel e lápis-b

6 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Pantanal (2), Torneio de tennis (2), hidrologia (1) e Tornado (1). Concluído em 23/12/07 com a sequência 198.

193d - 6 problemas a base de papel e lápis-c

6 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Ciclo de palestras (2), Analista (2), Casa mal afamada (1) e Notação pós-fixa (1). Concluído em 23/12/07 com a sequência 199.

193e - 7 problemas a base de papel e lápis-d

7 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Mapas a colorir (2), MP3 (3), Horta da Maria (1) e Sanduiche do João (1). Concluído em 23/12/07 com a sequência 200.

215a - Revisão de logaritmos

Exercício prévio para o estudo da complexidade algorítmica. São apresentadas as bases da potência, da radiciação e dos logaritmos. Em cada um destes temas, há 3 exercícios numéricos. Concluído em 06/01/03 com a sequência 099.

220a - Complexidade, recursividade e relação de recorrência

Descreve como um algoritmo é medido e daí mostra dois algoritmos equivalentes (mesma entrada $\Rightarrow$ mesma saída): um iterativo e outro recursivo e mostra como analisar ambos desenhos: usam-se relações de recorrência no caso recursivo. Concluído em 31/05/21 com a sequência 656.

222a - Alto e Baixo nível na programação I

Diferencia a programação de baixo e alto nível, usando para tanto uma implementação de conjuntos e suas 3 operações: união, intersecção e diferença. Concluído em 22/05/21 com a sequência 655.

223a - Alto e Baixo nível na programação II

Diferencia a programação de baixo e alto nível, usando para tanto uma implementação de pilhas e filas em C (baixo) e Python (alto). Concluído em 06/06/21 com a sequência 657.

224a - Alto e Baixo nível na programação III

Diferencia a programação de baixo e alto nível, usando para tanto uma implementação de árvores binárias de pesquisa em C e em Python. Concluído em 07/06/21 com a sequência 658.

225a - Alto e Baixo nível na programação IV

Diferencia a programação de baixo e alto nível, usando para tanto uma simulação de linguagens de alto e baixo nível, ambas definidas *ad hoc*. Concluído em 08/06/21 com a sequência 659.

226a - Alto e Baixo nível na programação V

Diferencia a programação de baixo e alto nível, usando para tanto o cálculo da matriz inversa em C++ (baixo nível) e em APL, Python, Octave,... (alto nível). Concluído em 10/06/21 com a sequência 660.

231a - Problema da partição

Um algoritmo aproximado (guloso) para resolver o problema da partição. Dada uma coleção de objetos, cada um com seu peso, divisor o conjunto em 2 subconjuntos disjuntos, de modo que a soma dos dois subconjuntos seja igual. Este problema é NP. Concluído em 17/07/02 com a sequência 094.

232a - Requisitos de um automóvel

Aplicação de algoritmo guloso: qual o menor conjunto de veículos (cada um com a sua lista de habilidades e seu preço) que satisfaz a uma lista de requisitos do comprador ? Concluído em 21/01/01 com a sequência 064.

233a - Habilidades de um jogador de futebol

Aplicação de algoritmo guloso: qual o menor conjunto de jogadores que satisfazem a todas as habilidades que um clube necessita ? Concluído em 18/01/01 com a sequência 063.

240a - Problema da Mochila (Knapsack)

Mais um exemplo de programação dinâmica (recursão otimizada, guardando dados em uma tabela para evitar os recálculos). Dada uma mochila que tem certo limite de peso na hora de ser preenchida com objetos, e uma lista de objetos cada um com seu peso individual e seu valor, resolver este problema é determinar qual a coleção de objetos disponíveis que maximiza o valor da mochila sem ultrapassar o peso máximo da mesma. A folha sugere um problema de 8 objetos e pede que o aluno o resolva. Concluído em 27/01/09 com a sequência 233.

245a - Mínimos quadrados

Apresenta a teoria do ajuste de uma reta a um conjunto dado de pontos. Mostra a teoria associada e a maneira de solução do problema. Apresenta também um algoritmo simples que o resolve. Finalmente, uma instância do problema é dada e pede-se ao aluno que infira dois novos pontos. Concluído em 25/11/10 com a sequência 268.

246a - Regressão múltipla

Descreve o algoritmo dos mínimos quadrados para regressão múltipla e apresenta um problema para ser resolvido. Concluído em 17/07/14 com a sequência 414.

251a - Prática de recursividade

3 funções recursivas e que se chamam entre si são definidas. Algumas variáveis globais também. Finalmente pede-se que o aluno siga as funções e reproduza as suas saídas. Concluído em 18/04/00 com a sequência 006.

252a - Anagramas

Pede que o aluno construa os anagramas com um determinado conjunto de letras. O algoritmo é recursivo. Ainda na versão 1. Concluído em 18/04/00 com a sequência 023.

253a - Passeio do cavalo

O conceito de *backtracking* é mostrado através de um exemplo: o passeio completo do cavalo sobre um tabuleiro de xadrez. Dado um tabuleiro $n \times n$ (do qual o 8×8 é um caso particular) e dada uma casa inicial pede-se qual a sequência de passos do cavalo de maneira a passar em todas as casas do tabuleiro sem repetir nenhuma. Finalmente uma instância do problema é oferecida ao aluno, pedindo-se uma parte do caminho feito pelo cavalo. Ótimo candidato a implementação. Concluído em 18/04/00 com a sequência 024.

254a - 3 funções recursivas

São mostradas 3 funções recursivas: uma ordenação recursiva, o cálculo da potência recursiva (aquela que abaixa a complexidade normal de $O(n)$ para $O(\log n)$) e o algoritmo do mdc de Euclides. Pede-se que o aluno simule as recursões e informe o que acontece após 2 ou 3 passos de recursão. Concluído em 24/07/03 com a sequência 058.

255a - Árvore quad

Apresenta a árvore quad como um exemplo de estrutura de dados recursiva. Visto o conceito, dá-se uma árvore armazenada em uma matriz e pede-se que o aluno construa o desenho correspondente. Depois dá-se um desenho e pede-se a matriz. No fim o aluno é surpreendido pelo fato de que os 2 exercícios referem-se à mesma árvore (e portanto a resposta de um é o enunciado do outro e vice versa), mas só quem gastou algum tutano descobre isto. Concluído em 04/02/02 com a sequência 084.

256a - Casamento Estável

Descreve o algoritmo (conforme mostrado pelo Niklaus Wirth no livro Algoritmos e Estruturas de dados, pág. 129) e apresenta dois conjuntos de 10 elementos cada, e suas respectivas preferências. São pedidas as perdas em dois casos: casamentos com minimização de perdas do primeiro conjunto e também do segundo conjunto. Este exercício tem que ser implementado. É quase impossível resolvê-lo à mão, por conta da recursividade. Concluído em 11/04/06 com a sequência 155.

257a - Sudoku para computadores

Descreve o sudoku e apresenta o algoritmo recursivo que o resolve. O aluno é convidado a seguir o algoritmo e preencher apenas a primeira linha do sudoku. Esta restrição é para que o aluno consiga seguir o algoritmo sem se enredar em chamadas recursivas. Concluído em 08/09/06 com a sequência 167.

258a - Programação Literata e 3 funções recursivas

Apresenta-se o conceito de Programação Literata, (ou literária) como foi proposto por D. Knuth em 1981. A seguir apresentam-se 3 funções recursivas simples: A geração do verso “99 garrafas de cerveja”, a geração dos números de Ackerman e a publicação por extenso de um valor qualquer. O aluno é desafiado a programar estas 3 coisas usando programação literata. Não há

propostas individuais diferentes nem correção automatizada. Este é um exercício de implementação, inclusive com verificação de contrafação. Concluído em 29/11/08 com a sequência 227.

259b - TSP de pequeno porte (6 cidades)

Apresenta o problema do caixeiro viajante entre 6 cidades (permite a solução manual: são 60 roteiros) e compara o resultado com o algoritmo guloso. Concluído em 06/02/14 com a sequência 406.

259e - Caixeiro Viajante em Python

Descreve o problema do caixeiro viajante e mostra os programas Python que o resolvem. O aluno deve resolver uma instância inédita de 10 cidades reais do estado do Paraná. Concluído em 25/05/17 com a sequência 539.

259h - Travelling Salesman Problem

O problema do caixeiro viajante é apresentado e são feitas considerações sobre os algoritmos heurísticos (não exatos) de sua solução. Depois é apresentado o algoritmo recursivo exaustivo que só pode ser usado em instâncias pequenas. Aqui são 10 cidades e o aluno deve implementar o algoritmo e descobrir a sequência de cidades e o comprimento da viagem. Concluído em 03/02/10 com a sequência 257.

260b - Árvores usando recursividade e apontadores

Todos os algoritmos de uma árvore binária de pesquisa usando recursividade e apontadores programados em Python. O aluno precisa criar uma árvore com 20 valores, excluir 6 deles e depois caminhar em largura na árvore. Concluído em 17/10/20 com a sequência 649.

260c - Árvores usando recursividade e cursores

Uma folha muito parecida com a 260b, mas ao invés de apontadores usa cursores. A idéia é comparar as duas e concluir sobre benefícios de uma e de outra. Concluído em 17/10/20 com a sequência 650.

261a - Árvores k-d

Descreve a árvore k-d como uma estratégia para estudar a distribuição geográfica de pontos no espaço. Apresenta dois sabores de AKD e pede que o aluno faça uma implementação com 20 pontos no espaço 2D. Concluído em 28/06/21 com a sequência 661.

262a - Recursividade: Strassen e Quickselec

Descreve o algoritmo de Strassen (para multiplicar 2 matrizes) que representa um ganho sobre o algoritmo canônico. Também mostra o quickselect que acha a mediana de um conjunto de dados sem precisar ordená-lo. Em ambos os casos são mostrados dados de desempenho reais. Concluído em 16/10/18 com a sequência 598.

263a - Programação Linear: Método Simplex

Descreve o método simplex usando um tableaux algébrico e depois sugere usar o pacote Python PULP (que implementa o método simplex). Concluído em 16/10/18 com a sequência 597.

265a - Programação Dinâmica

Instâncias inéditas dos problemas Euler 67, 81 e 82 apresentando o conceito da programação dinâmica e os algoritmos necessários para resolver estes problemas. Os dados são entregues em mídia magnética devido a seu volume. Concluído em 17/08/14 com a sequência 416.

266a - Seleção de Atividades

Aplicação do algoritmo guloso: seleção de atividades. Um centro cirúrgico com 30 cirurgias previstas deve ser otimizado. Pede quantas cirurgias poderão ser feitas e quantos minutos do total do dia (1440 minutos) poderão ser efetivamente usados no CC. Concluído em 30/09/19 com a sequência 635.

271a - Estratégias de alocação

Descreve as estratégias de alocação next fit, first fit, best fit, worst fit e alocação offline. Depois uma certa arrumação de sacos de compra em um mercado é pedida em cada uma das estratégias. Concluído em 26/10/01 com a sequência 076.

272a - Alocação em um teatro

Um teatro usa uma estrutura encadeada para reservar lugares, além de guardar informações sobre os compradores nessa mesma estrutura. Dado um teatro parcialmente vendido, pede-se que o aluno responda perguntas sobre o teatro e sobre os compradores de lugares. Concluído em 15/05/01 com a sequência 073.

273a - Uma família representada em uma estrutura de dados

Uma família com 5 gerações é mostrada através de uma estrutura encadeada. Trata-se de uma matriz de 50 linhas e 6 colunas informando as relações parenterais na família. Pede-se que o aluno desenhe a árvore genealógica, além de responder a diversas perguntas sobre a família. Este é o exercício inaugural em “estrutura de dados” e serve para mostrar a lógica dos algoritmos vivos. Concluído em 18/02/02 com a sequência 085.

291a - Alocação do DOS

Mostra-se o mecanismo que o DOS usava (usa) para gerenciar alocações de espaço em disco. Uma série de criação, modificação e apagamento de arquivos é pedida ao aluno. Ao final, pede-se o estado do diretório e da FAT do disco. Concluído em 04/05/00 com a sequência 007.

292a - Alocação do UNIX

Mostra-se o mecanismo que o UNIX usa para gerenciar alocações de espaço em disco. Uma série de criação, modificação e apagamento de arquivos é pedida ao aluno. Ao final, pede-se o estado dos blocos em disco. O superbloco não é pedido, mas deve ser montado pelo aluno para ajudar a responder. Concluído em 04/05/00 com a sequência 008.

293a - Alocação de memória pelo LISP

Descreve-se como o LISP aloca memória para as variáveis e as funções que são nele definidas. O conceito de célula, CONS e CAR de cada célula é descrito. Finalmente é dada uma lista LISP e pede-se que o aluno faça o mapeamento interno da memória, simulando o que o LISP faz. O exercício pede a representação interna da lista dada. Concluído em 06/11/01 com a sequência 078.

311a - Simulação de pilhas

Dadas os algoritmos de empilhamento e desempilhamento e dadas 5 pilhas e sugeridas 50 operações de empilhamento e desempilhamento nestas 5 pilhas pede-se, ao final, o que cada uma das 5 pilhas contém. O aluno deve responder a 15 perguntas sobre os conteúdos das pilhas. Concluído em 02/05/00 com a sequência 041.

312a - Pilhas em baixo nível

São dados os algoritmos de criação, inclusão e exclusão em pilhas tanto sequenciais como encadeadas, em baixo nível (instruções semelhantes ao assembler). Diversas operações são feitas em um bloco de memória e ao final pede-se informações sobre como ficou esse bloco de memória. Este é um dos exercícios mais difíceis da coleção. Concluído em 02/05/03 com a sequência 107.

313a - Notação polonesa reversa

Uma aplicação de pilhas. São mostrados s algoritmos de conversão entre notação convencional (infixa) e as notações pré e pós-fixa. Depois é mostrado o algoritmo de cálculo de expressão pós-fixa. O exercício mostra uma expressão parentizada convencional e pede a expressão RPN correspondente. Depois pede que a expressão RPN seja calculada. Concluído em 02/05/00 com a sequência 009.

314a - Passeio do rato no labirinto

Uma aplicação de pilhas. Dado um labirinto, um rato é colocado em uma certa posição. Um algoritmo de busca da saída é dado. Pede-se a simulação do caminho percorrido pelo rato. A pilha é usada para memorizar posições de bifurcação, permitindo ao rato retornar quando encontrar becos sem saída. Concluído em 19/05/00 com a sequência 039.

315a - Envoltória convexa de Graham

Aplicação de pilhas. Dados um conjunto de pontos no plano, pede-se quais são os pontos da envoltória convexa. (em outros termos: quais os pontos que seriam tocados por um elástico que englobasse o conjunto). É uma aplicação de geometria computacional interessante. Concluído em 13/01/03 com a sequência 102.

318a - Construção de um compilador

Uma aplicação de pilhas. Dadas uma linguagem fonte (estruturada, contendo enquanto, repita, para e se) e uma objeto (contendo apenas testes e desvios incondicionais e condicionais) e dada uma descrição do processo de compilação usado, dá-se um programa fonte e pede-se que o aluno simule a compilação apresentando o objeto gerado. Concluído em 18/12/05 com a sequência 143.

319a - Engenharia reversa de compiladores

Este exercício descreve as regras de uma linguagem fonte e sua correspondente linguagem objeto, e daí pede duas coisas

1. Dado um programa objeto, qual foi o seu programa fonte ?
2. Dado um fonte e seu objeto compilado a menos de 2 comandos, quais são estes dois comandos ?

Deve ser usado logo após o exercício 318. Concluído em 02/02/06 com a sequência 150.

321a - Filas sequenciais

Dadas os algoritmos de enfileiramento e desenfileiramento e dadas 5 filas e sugeridas 50 operações nestas 5 filas pede-se, ao final, o que cada uma das 5 filas contém. O aluno deve responder a 15 perguntas sobre os conteúdos das filas sequenciais. Concluído em 12/06/00 com a sequência 040.

322a - Filas sequenciais circulares

Dadas os algoritmos de enfileiramento e desenfileiramento e dadas 5 filas circulares e sugeridas 50 operações nestas 5 filas pede-se, ao final, o que cada uma das 5 filas contém. O aluno deve responder a 15 perguntas sobre os conteúdos das filas sequenciais circulares. Concluído em 12/06/00 com a sequência 043.

323a - Filas em baixo nível

São dados os algoritmos de criação, inclusão e exclusão em filas tanto sequenciais circulares como encadeadas, em baixo nível (instruções semelhantes ao assembler). Diversas operações são feitas em um bloco de memória e ao final pede-se informações sobre como ficou esse bloco de memória. Este é um exercícios bem difícil. Concluído em 03/05/03 com a sequência 108.

324a - Ordenação topológica usando filas

Dados todos os componentes de roupa que uma pessoa normal veste em um dia de frio e chuva, pede-se ao aluno que ache a sequência de operações para vestir-se. O algoritmo de ordenação topológica é mostrado no exercício. Concluído em 30/05/00 com a sequência 045.

331a - Truco em listas encadeadas

Uma memória encadeada contendo os dados de uma partida de truco é mostrada. Pede-se o resultado das 3 rodadas e da partida, a partir dos dados da memória e das regras oficiais (?) do truco. Este exercício é uma festa em sala de aula. Concluído em 27/06/00 com a sequência 042.

332a - 4 erros em listas encadeadas

São descritos os erros: loop na lista, endereços de próximo registro inválidos, junção de 2 listas distintas e dados inválidos na lista. Dada um bloco de memória que contém 4 listas (cada uma com um dos erros acima) pede-se que o aluno localize cada lista e cada erro. Não é um exercício trivial. Concluído em 23/02/01 com a sequência 065.

333a - Criação e manuseio de 5 listas encadeadas

Uma biblioteca contendo 25 livros com 5 encadeamentos (título do livro, idioma, preço, páginas e cidade de edição) é mostrada. Pede-se que o aluno inclua 2 novos livros, acertando os 5 encadeamentos de cada livro. Usam-se cursores ao invés de apontadores para permitir a execução e correção do exercício. Concluído em 01/08/00 com a sequência 026.

341a - Listas duplamente encadeadas em baixo nível

Descreve-se uma lista duplamente encadeada e são dados os algoritmos de manutenção (os algoritmos não cabem na folha de exercício, eles estão apenas no texto da aula). Após 13 operações pede-se como ficou o bloco de memória. Concluído em 29/04/01 com a sequência 047.

342a - Manuseio de listas duplamente encadeadas

São mostradas 3 listas: de países, de idiomas e de cidades, as 3 com encadeamentos duplos usando cursores. Feitas inclusões e exclusões nas 3 listas, pede-se ao final como elas ficaram. Concluído em 01/08/00 com a sequência 017.

343a - BDs usando encadeamento duplo

São apresentados 2 bancos de dados contendo cada um 4 atributos devidamente encadeados duplamente (em ordem ascendente e descendente) Totalizando 8 encadeamentos por BD. Em cada um deles, o aluno deve fazer uma inclusão e a seguir uma exclusão respondendo a seguir como ficaram os 8 encadeamentos. Concluído em 17/05/10 com a sequência 258.

351a - Equações diferenciais

Uma pequena e não completa introdução às equações diferenciais. Pede que o aluno resolva 3 instâncias. Concluído em 24/03/24 com a sequência 689.

352a - Transformada de Laplace

Apresenta a Transformada de Laplace, a convolução e a T.L.inversa. Mostra como resolver equações diferenciais usando esta ferramenta. Pede que o aluno resolva 3 instâncias. Concluído em 14/03/24 com a sequência 687.

353a - Taxas de variação relacionadas

Mostra como resolver problemas com taxas relacionadas. Relembra a regra da cadeia e a regra do produto. Pede que o aluno resolva 3 instâncias. Concluído em 17/03/24 com a sequência 688.

411a - Representação e caminhamento em árvores binárias

Este exercício mostra 3 representações de árvores binárias (usando cursores, usando apontadores em um bloco de memória e usando parênteses. O exercício mostra 3 árvores (uma em cada metodologia de representação) e pede que o aluno faça os 3 caminhamentos (em-ordem, pré-ordem e pós-ordem) nas 3 árvores apresentadas. Concluído em 07/08/00 com a sequência 046.

412a - Conversão de árvores n-árias em binárias

O exercício mostra uma árvore de grau 6 usando cursores e pede que o aluno escreva uma árvore binária equivalente. Concluído em 01/07/02 com a sequência 090.

421a - Balanceamento em árvores ABP

Um desenho de uma ABP desbalanceada é apresentado pedindo-se ao aluno que refaça o desenho balanceando a árvore. Depois disso pede-se o caminhamento em largura da árvore. Concluído em 14/07/00 com a sequência 048.

431a - Caminhamento e inclusões em ABPs

Uma árvore é apresentada através de sua matriz de cursores. Pede-se que o aluno a desenhe e a seguir responda diversas questões sobre ela. Concluído em 11/03/00 com a sequência 010.

432a - ABPs: busca, criação e altura

Cerca de 8 ABPs devem ser construídas a partir de uma alimentação sequencial de itens. A seguir sobre cada uma das árvores são feitas diversas perguntas que devem ser respondidas. Concluído em 18/06/02 com a sequência 089.

433a - Inclusões e exclusões em ABPs

Os algoritmos de inclusão e de exclusão são mostrados. A seguir 3 árvores compartilhando a mesma matriz de cursores são mostradas. 28 operações são aplicadas sobre as árvores. Ao final pede-se o caminhamento sobre as 3 árvores. Concluído em 07/08/00 com a sequência 054.

434a - Árvore binária de pesquisa com as letras do nome

Pede que o aluno construa uma árvore de pesquisa binária com as letras de seu nome. Ainda na versão 1. Concluído em 17/07/08 com a sequência 221.

434a - Inclusão e Exclusão em 3 ABPs

O aluno é convidado a construir 3 Árvores Binárias de Pesquisa a partir dos elementos numéricos de cada uma, na sequência de entrada. Depois, 20 operações de inclusões e exclusões de uma para a outra árvore são feitas. Depois, perguntas sobre a sequência de visitação das árvores que sobram depois da movimentação são feitas. O número 434 foi reaproveitado de um exercício velho. Concluído em 17/07/08 com a sequência 221.

435a - Árvores binárias de partilha

As ABPs são estruturas que associam a eficiência de uma ABP equilibrada garantindo ao mesmo tempo que as chaves mais acessadas estarão mais próximas da raiz do que as menos acessadas. Com isso a complexidade média, que é $O(\log n)$ no caso das ABPs torna-se ainda menor. Os algoritmos de criação e de busca são mostrados e o aluno deve construir uma ABPa. Concluído em 04/08/03 com a sequência 111.

436a - Tries

As tries são árvores muito eficientes para funcionar como dicionários. A folha mostra 2 implementações possíveis (usando-se árvores de grau k , onde k é a cardinalidade do alfabeto e usando-se árvores binárias). É dado um pequeno alfabeto e o aluno deve construir as 2 tries. Concluído em 09/08/03 com a sequência 112.

437a - Treinamento em ABP

Descreve a árvore binária de pesquisa e pede para o aluno treinar a teoria construindo 10 árvores cada uma com 15 elementos. Concluído em 17/07/21 com a sequência 654.

441a - Aquecimento em árvores B

Apenas exemplos de árvores B são mostrados sem maior preocupação com o rigor na definição. Algumas inclusões *ad hoc* são feitas e o aluno é convidado a antecipar alguns resultados. Concluído em 02/07/02 com a sequência 091.

442a - Árvore B com grau $t=3$

É apresentada a definição formal da árvore B e os algoritmos de criação são mostrados. O aluno é convidado a construir uma árvore B de grau $t=3$ com 40 números. Concluído em 26/09/04 com a sequência 027.

443a - Simulando índices do SYBASE

Um exemplo real de árvores B. É dado um arquivo com 8 registros e 5 índices, sendo um esparsos e 4 não granulados. O aluno deve construir as páginas de índices do SYBASE (que são árvores B) e responder a perguntas sobre elas. Concluído em 22/11/04 com a sequência 055.

444a - xBase

O ambiente xBase (originalmente criado pelo dBase) é descrito com seus arquivos DBF, DBT e NDX. Este último (que é uma árvore B) é descrito com algum detalhe. Finalmente os alunos recebem um *dump* de memória contendo um arquivo NDX real e devem responder a perguntas sobre ele. Concluído em 10/08/01 com a sequência 074.

445a - B árvore com $t=5$

Uma B-árvore com $t=5$ é apresentada parcialmente preenchida e o aluno é convidado a nela incluir mais 20 chaves. Depois são feitas perguntas sobre a árvore que já sofreu as inclusões. Concluído em 01/10/04 com a sequência 028.

450a - Codificação e decodificação

Este exercício é uma preparação para a compressão de Huffman. Solicita conversões entre binário, hexadecimal, ASCII. São criados 2 códigos novos: o ORTELINO (no qual cada caracter usa 5 bits) e o TROCALETRA (no qual cada caracter usa um número variável de bits. No mínimo 4 e no máximo 8 bits por caracter). Conversões de um para o outro e do outro para o um são pedidas. Concluído em 13/03/01 com a sequência 071.

451a - Compressão usando o algoritmo de Huffman

O algoritmo de Huffman é apresentado através de um simples exemplo. As etapas do algoritmo são descritas. Finalmente, apresenta-se um universo comum a todos os exercícios (o que garantirá a mesma árvore para todos) e pede-se a compressão e a descompressão usando essa árvore de 2 falas de peças de Shakespeare. Concluído em 01/01/93 com a sequência 001.

452a - Huffman, árvores distintas

Na continuação do exercício 451, novas frases de Shakespeare devem ser convertidas e desconvertidas. Mas agora, cada aluno terá um universo de caracteres diferentes, fazendo com que cada um tenha que construir a sua própria árvore. Concluído em 01/01/94 com a sequência 002.

453a - Huffman, árvore dinâmica

A construção de uma árvore dinâmica de Huffman é mostrada passo a passo ao mesmo tempo que os textos vão sendo comprimidos e/ou descomprimidos. Agora as mensagens são curtas (tipicamente 8 caracteres). Os algoritmos não são mostrados, estando apenas no texto da aula. Concluído em 16/02/04 com a sequência 118.

455a - Compressão: Transformadas BWT e MTF

O método de Burrows e Wheeler para compressão é apresentado. Ele pressupõe 3 transformações nos dados:

- Primeiro a aplicação da transformada Burrows-Wheeler, que não modifica a entropia dos dados, mas os reagrupa aumentando a localidade de referência espacial.
- Depois aplica-se a transformada “Move To Front” (MTF) que diminui ainda mais a entropia dos dados, transformando o bloco processado em um conjunto de zeros e números pequenos
- Finalmente, um compressor qualquer (por exemplo, Huffman) que ganhou condições agora de fazer uma super compressão.

A folha descreve tudo isso e apresenta o resultado de uma transformação MTF. Ao aluno cabe fazer a MTF inversa e depois a BWT inversa. Concluído em 26/12/08 com a sequência 230.

456a - Caso prático de Huffman

Uma situação hipotética é apresentada, onde se sugere usar o algoritmo de Huffman devidamente modificado: ele comprimirá partes muito frequentes do arquivo e deixará sem comprimir partes pouco frequentes. Este exercício é para mostrar que uma vez dominado o algoritmo ele pode ser manuseado e flexibilizado à vontade. Concluído em 11/07/11 com a sequência 283.

457a - Huffman para imagens BMP

Pede que o aluno leia uma imagem BMP como sendo um arquivo binário. Daí pede que ele monte a árvore de Huffman com as diversas cores existentes na imagem. 7 perguntas são feitas sobre tal árvore. Concluído em 21/06/12 com a sequência 336.

458a - Prática em Huffman: os lusíadas

Entrega um arquivo sequencial contendo uma árvore e um string comprimido contendo 3 versos aleatórios d’Os Lusíadas. O aluno deve descomprimir o texto. Usa-se o utilitário DUMP.PY para examinar o arquivo. Concluído em 10/07/17 com a sequência 543.

461a - Heap - construção e pesquisa

Um HEAP é formalmente definido e são pedidos 4 exercícios aos alunos abordando todas as facetas de um heap. Concluído em 22/02/02 com a sequência 018.

480a - Unicode e UTF-8

Descreve o código Unicode e as razões que levaram a ele. Descreve também o mecanismo UTF-8 de codificação e pede que o aluno decodifique 5 caracteres Unicode codificados em UTF-8. Concluído em 25/10/14 com a sequência 437.

511a - Algoritmos de ordenação - 1ª parte

Apresenta os algoritmos de ordenação bolha, bolha-estrela, inserção e seleção. Comparativos entre os 4 métodos em termos de tempo são mostrados. O aluno deve responder a questões sobre o estado das variáveis internas alguns passos após o início de cada algoritmo. Concluído em 03/01/03 com a sequência 097.

511b - Aula única de ordenação

Esta folha é apenas um resumo teórico das aulas de ordenação. Não há exercício a ser feito, apenas um desafio de pensar na *sort* externo Concluído em 01/11/09 com a sequência 253.

512a - Algoritmos de ordenação - 2ª parte

Apresenta os algoritmos de ordenação shell, heap e quick fazendo um comparativo entre eles e com os algoritmos do exercício 511. O aluno deve responder a questões sobre o estado das variáveis internas alguns passos após o início de cada algoritmo. Concluído em 04/01/03 com a sequência 098.

513a - Redes de comparação e de ordenação

Estabelece a discussão de ordenação em ambientes multiprocessados, baixando o melhor custo da ordenação que era $O(n \times \log_2 n)$ para $O(\log_2^2 n)$. Perguntas são feitas sobre o estado da rede de ordenação. Concluído em 02/01/03 com a sequência 096.

521a - Tabelas de dispersão: hash tables

Este exercício introduz o conceito de tabelas de dispersão e pede ao aluno que construa duas delas: a primeira com áreas primária e secundária separadas e com encadeamento entre elas. A segunda com apenas área primária e dispondo as colisões à direita do elemento destino. Discussões sobre funções hash adequadas também são conduzidas. Concluído em 11/09/99 com a sequência 019.

531a - Busca em cadeias: algoritmo de Boyer-Moore

O exercício começa descrevendo o algoritmo de força bruta para realizar esta tarefa, mostrando a quantidade de testes necessários para conduzir a pesquisa em uma pequena frase. A seguir as características do algoritmo de Boyer-Moore são apresentadas. Finalmente uma busca é conduzida usando este algoritmo. O exercício permite concluir pela superioridade do método apresentado. Concluído em 24/01/02 com a sequência 082.

532a - Busca em tabelas

Este é um exercício importante da série ao permitir a comparação de diversos métodos de busca em tabelas. São mostrados o método de busca linear, o de busca linear em tabela ordenada, o de busca linear com sentinela, o busca através de tabelas hash, a busca binária, e a busca usando uma árvore binária de pesquisa. Dados numéricos para um determinado problema são mostrados, comentados e comparados. Finalmente, os alunos são convidados a usar todos os métodos e apresentar a quantidade de testes em cada um. Concluído em 01/09/03 com a sequência 115.

535a - Casamento Aproximado de Cadeias

O problema de buscar as modificações (inclusões, substituições e exclusões) que são necessários em um padrão a fim de obter batimento com um texto qualquer. Extraído de Skiena, S. The Algorithm Design Manual, págs 60 e 406. Concluído em 05/06/08 com a sequência 216.

536b - Bioinformática - 1

Começa descrevendo a bioinformática e alguns de seus algoritmos fundamentais. O primeiro deles é o problema da maior subsequência comum a duas cadeias. Mais uma aplicação de programação dinâmica. Baseado no livro Cor14 (901-3). Concluído em 09/10/14 com a sequência 435.

536c - Bioinformática - 2

Continua onde o exercício 536b parou. Trata os custos de substituição de uma cadeia por outra e do problema de busca de padrões em textos, conhecido como CTRL-F. Baseado no livro Cor14 (901-3). Concluído em 09/10/14 com a sequência 436.

538a - Expressões Regulares

Dá um resumo das expressões regulares e pede que o aluno resolva 4 casos. Concluído em 02/12/14 com a sequência 443.

555a - Representação de estruturas bidimensionais

Descreve os múltiplos uso dos parênteses em estruturas não lineares: árvores, árvores de pesquisa, análise sintática de expressões aritméticas, lógica proposicional e notação pré-fixada de expressões aritméticas. Concluído em 23/06/24 com a sequência 724.

560a - Análise Estilística de Textos

Descreve como pode ser construído um analisador simples de estilo de escrita e descreve algumas métricas possíveis. Mostra a Lei de Heaps e a de Zipf. Concluído em 24/06/14 com a sequência 413.

570a - Codificação Reed Solomon

Descreve a codificação Reed Solomon, os conceitos de grupo, campo e dos Campos de Galois. Descreve a adição e a multiplicação em campos finitos. Finalmente a codificação é descrita e apenas é citada a decodificação. Os alunos são convidados a calcular 4 símbolos de paridade em uma mensagem aleatória. Concluído em 27/01/15 com a sequência 464.

571a - QR Code

Descreve os padrões de leitura ótica conhecidos como *QR Code*. Descreve a versão (capacidade de dados), nível (de erro e de correção) e modo (de codificação de informações que pode ser numérica, alfanumérica, de byte e de caracteres Kanji já que estes códigos nasceram no Japão. Ao aluno é pedido que decodifique 2 códigos numéricos e 2 códigos alfanuméricos. Concluído em 31/01/15 com a sequência 465.

611a - Grafos usando listas encadeadas de arestas

Um ambiente de descrição de grafos usando arestas encadeadas em uma estrutura similar a uma memória é mostrado. A seguir pede-se que o aluno estabeleça a matriz de caminhos mínimos (ainda de maneira intuitiva: é fácil, o grafo tem apenas 6 vértices e 20 arestas). Depois pergunta-se quantos pares de vértice tem caminhos que custam menos do que a ligação direta e qual o diâmetro (o maior valor dentro da matriz de caminhos mínimos) do grafo. Concluído em 14/07/02 com a sequência 092.

612a - Representação e caminhamento em grafos

O exercício mostra a representação de grafos através de matrizes de adjacência e de incidência. Depois são mostrados algoritmos de caminhamento em profundidade e em largura. É dado um grafo de 14 vértices e pede-se que o aluno monte os caminhos em largura e em profundidade. Concluído em 10/10/97 com a sequência 012.

618a - Uso do pacote GraphViz

Este exercício trata do desenho de grafos, mostrando a complexidade do problema. Recomenda o uso do GRAPHVIZ e sugere um problema a ser resolvido pelo aluno usando este software. Concluído em 07/05/08 com a sequência 215.

621a - Caminho mínimo: algoritmo de Dijkstra

O algoritmo é mostrado e são feitas 2 simulações: uma pequena com um grafo de 5 vértices e uma maior com um grafo de 10 vértices. A seguir o aluno deve fazer o mesmo com um grafo de 10 vértices. Concluído em 11/10/98 com a sequência 013.

622a - Caminho mínimo 1:1

Pede que o aluno calcule o caminho mínimo entre dois vértices de um dado grafo. Ainda na versão 1. Concluído em 05/05/98 com a sequência 014.

623a - Caminho mínimo: algoritmo de Floyd-Warshall

Este algoritmo calcula os caminhos mínimos de todos os vértices para todos os vértices, tendo complexidade $O(n^3)$. Primeiro é apresentado um exemplo simplório de 3 vértices e depois outro maior de 6 vértices que é feito em conjunto professor e alunos. Finalmente, os alunos são convidados a resolver um grafo de 7 vértices, devendo responder o valor do caminho para 5 pares de vértices aleatoriamente escolhidos. Concluído em 20/02/97 com a sequência 015.

624a - Caminho mínimo: roteamento através de Floyd modificado

O algoritmo de roteamento de Floyd é mostrado e um exemplo de 5 vértices é resolvido. Os alunos são convidados a resolver um grafo de 7 vértices devendo responder a 3 perguntas sobre custos dos caminhos e a 3 perguntas sobre as rotas dos caminhos achados. Este exercício deve ser feito após o exercício 623. Concluído em 30/03/97 com a sequência 016.

627a - Caminho mínimo no metrô de Curitiba

Aplicação prática do algoritmo de Floyd Warshall. Transplantou-se a idéia do metrô de Paris para Curitiba. Lá são 14 linhas completamente interligadas e aqui foram postas 11 linhas. O aluno deve estudar o problema e resolver a matriz dos caminhos mínimos bem como a matriz de roteamentos. Quando nada, é um exercício prá lá de divertido, dá para se imaginar morando em uma cidade do primeiro mundo. Concluído em 09/10/09 com a sequência 252.

628a - Rotas e tempos mínimos

Propõe ao aluno um problema real de logística pelo qual deve-se distribuir uma quantidade de cargas em 15 cidades do Paraná e Santa Catarina. Concluído em 31/08/12 com a sequência 343.

632a - Cálculo de fluxos: Ford-Fulkerson, Edmonds-Karp

Descreve-se o problema de análise de fluxos. É como se um grafo indicasse a vazão de conexões entre pontos diversos de uma planta. A pergunta a responder é qual a vazão máxima entre 2 pontos da planta ? É resolvida uma instância de 7 pontos e pede-se que o aluno resolva outra instância inédita de 7 pontos. Concluído em 15/07/02 com a sequência 093.

633a - Ordenação topológica usando grafos

O sequenciamento de um curso de informática hipotético é apresentado com suas disciplinas e pré-requisitos. A seguir o algoritmo que o resolve é apresentado e o aluno deve achar qual o caminho ideal (a sequência) de disciplinas que deve ser cursada. Concluído em 22/10/01 com a sequência 075.

634a - Árvore de cobertura mínima: algoritmo de Kruskal

A árvore de cobertura mínima é o menor grafo que conecta todos os pontos de uma determinada planta. Sempre tem a característica de uma árvore, daí a razão do nome. Um algoritmo (guloso) é mostrado e o aluno convidado a

calcular uma ACM para um grafo de 10 vértices. Concluído em 18/01/02 com a sequência 080.

635a - Coloração de grafos

A característica NP do problema é ressaltada. A seguir um exemplo real de tempos de um semáforo em um cruzamento hipotético de 5 ruas é mostrado. Depois o algoritmo (guloso) é mostrado e o aluno convidado a colorir um grafo de 12 vértices. Concluído em 19/08/03 com a sequência 113.

636a - Aplicação de grafos: quebra cabeça dos tijolos

Um dominó contendo 10 peças e 3 animais em cada peça é apresentado. Mostra-se como montá-lo em um dominó. O problema pode ser convertido em buscar um caminho hamiltoniano em um grafo de 10 vértices (1 a cada peça) e com 15 arestas (cada vértice está ligado a outros 3, representando os animais de cada peça. Se o problema puder ser representado em um grafo de Petersen, não haverá solução, caso contrário haverá. O aluno é convidado a tentar resolver 2 dominós diferentes. Concluído em 29/10/05 com a sequência 141.

639a - Coloração de grafos: aplicações

Esta folha endereça várias habilidades. Descreve o problema NP-completo da coloração de grafos e apresenta 5 aplicações reais para ele. Depois descreve a solução de um sudoku usando esta teoria. Finalmente sugere a utilização do utilitário DSATUR que se utiliza de uma heurística conhecida como grau de saturação do nodo. O aluno deve resolver uma instância de sudoku usando este utilitário. Vê-se também o formalismo chamado DIMACS para a descrição de um grafo não dirigido. Concluído em 08/10/19 com a sequência 636.

640a - Método PERT-CPM

Este exercício propõe um projeto formado de eventos e tarefas, estas com suas durações e principalmente com a lista de precedências (uma tarefa só pode ser começada depois que outras terminarem). O exercício ensina o algoritmo do caminho crítico e pede que o aluno calcule a duração do projeto proposto e quais os eventos (e tarefas) que estão no caminho crítico. Concluído em 15/02/06 com a sequência 151.

645a - Onde pôr o hipermercado ?

Descreve um possível algoritmo de localização otimizada para um hipermercado dentro de uma cidade. É dado o mapa populacional da cidade e a localização de 3 hipermercados já existentes. Uma regra de influência é estabelecida e a pergunta é em que local o novo mercado deve se estabelecer a fim de otimizar a quantidade de clientes que ele terá. Concluído em 14/12/10 com a sequência 270.

646a - Central 0800

Um grupo de 12 cidades paranaenses gera um certo tráfego telefônico, sujeito a 4 degraus tarifários. A pergunta é qual cidade gerará menos custos para a instalação da central 0800. Concluído em 24/06/13 com a sequência 386.

650a - Algoritmo Húngaro

Um problema de otimização combinatória. Trata-se de uma lista de pessoas e de uma lista de tarefas. Cada combinação pessoa-tarefa tem um custo associado. O algoritmo informa, ao final, qual a combinação que garante custo mínimo ao conjunto. Concluído em 01/05/07 com a sequência 175.

656a - Algoritmo de Gale-Shapley

Este Algoritmo resolve o problema originalmente apresentado como “Problema de Admissão à Faculdade” na década de 70. Os autores apresentaram um algoritmo que sempre acha uma solução ótima para o problema. Este algoritmo é uma generalização do problema do casamento estável e ele é orientado a residentes. Existe outra versão orientada a hospitais. Neste exercício uma instância de 5-9 hospitais e 13-19 residentes é apresentada e pede-se que o aluno ache a solução. A referência é a Encyclopedia of Algorithms de Ming-Yang Kao (ed.), pág. 390. Concluído em 17/01/09 com a sequência 232.

660a - Torneio de Tennis

Um exemplo da estratégia dividir para conquistar. Trata-se de construir uma tabela de jogos envolvendo todos os oponentes jogando entre si no menor número possível de dias. O exercício não pede a tabela, já que esta pedida é muito difícil de ser feita sem implementação (fica como ótimo desafio), mas apenas a sequência de chamadas recursivas à função. Concluído em 07/08/07 com a sequência 187.

666b - Estratégias de Programação 1

Baseado no livro BHARGAVA, Aditya. Entendendo Algoritmos. São Paulo, Novatec, 2017. Descreve algumas estratégias de programação e algoritmos: Teoria da Complexidade, Busca Binária, alocação encadeada e sequencial e Quicksort. Concluído em 27/07/17 com a sequência 545.

666c - Estratégias de Programação 2

Baseado no livro BHARGAVA, Aditya. Entendendo Algoritmos. São Paulo, Novatec, 2017. Descreve algumas estratégias de programação e algoritmos: Tabelas Hash, Grafos, Busca em largura, Algoritmo Guloso. Concluído em 27/07/17 com a sequência 546.

666d - Estratégias de Programação 3

Baseado no livro BHARGAVA, Aditya. Entendendo Algoritmos. São Paulo, Novatec, 2017. Descreve algumas estratégias de programação e algoritmos: Programação dinâmica, Árvores, Compressão de Huffman e o algoritmo dos k vizinhos mais próximos. Concluído em 27/07/17 com a sequência 547.

666e - Estratégias de Programação 4

Baseado no livro BHARGAVA, Aditya. Entendendo Algoritmos. São Paulo, Novatec, 2017. Descreve algumas estratégias de programação e algoritmos: Transformada de Fourier, Algoritmos Paralelos e Filtros de Bloom. Concluído em 27/07/17 com a sequência 548.

668a - Algoritmo dos k-vizinhos

Supõe que você é o programador do NetFlix e deve sugerir 3 filmes para um usuário, pergunta a ele qual foi o último filme que ele viu e gostou. Usa o algoritmo dos 3-vizinhos baseado em 6 dimensões. Concluído em 29/07/17 com a sequência 549.

669a - Filtros de Bloom

Descreve o conceito de algoritmos probabilísticos e exemplifica com um Filtro de Bloom. A idéia e o algoritmo são explicados e pede-se que o aluno construa um filtro para 2000 elementos, consultando 10 elementos novos depois. Concluído em 31/08/17 com a sequência 551.

671a - Multiplicação de matrizes

Descreve o problema da minimização de multiplicações em uma sequência de multiplicação de matrizes. Uma aplicação prática de programação dinâmica (e alguma recursividade). Concluído em 25/09/15 com a sequência 473.

672a - Program. Dinâmica: plano de produção

Descreve a estratégia da programação dinâmica oferecendo como problema de trabalho um plano de produção de 2 linhas de produção devendo descobrir qual o plano que gera o menor tempo. De quebra apresenta o conceito de memoitização na recursividade. Concluído em 06/10/15 com a sequência 474.

680a - Padrão PDF

Descreve-se o padrão PDF de descrição de páginas. Suas principais características e vantagens. Pede-se que o aluno construa um arquivo PDF a partir de um editor ASCII convencional. Concluído em 09/08/13 com a sequência 391.

690c - Compressão JPG em C/C++

O processo de criação de um arquivo JPG é comentado e especial ênfase é dada à transformada discreta do cosseno (DCT). Apresenta-se um bloco 8×8 de uma imagem e pedem-se dois valores da DCT. Concluído em 07/08/13 com a sequência 390.

690p - Compressão JPG em Python

O processo de criação de um arquivo JPG é comentado e especial ênfase é dada à transformada discreta do cosseno (DCT). Apresenta-se um bloco 8×8 de uma imagem e pedem-se dois valores da DCT. Concluído em 07/08/13 com a sequência 390.

701a - Detecção e correção de erros: algoritmo de Reed-Muller

Baseado num problema real (a transmissão de dados da nave Mariner entre Marte e a Terra na década de 70) este exercício introduz diversos conceitos: distância de Hamming, códigos Gray e matrizes de Hadamard. Depois disso tudo o algoritmo de Reed Muller é mostrado. Um exercício de 16 bits (podendo recuperar até 3 errados) é feito em conjunto. Depois o aluno recebe 3 mensagens que podem ou não ter erro e é convidado a informar quais as mensagens corretas. Concluído em 20/01/02 com a sequência 081.

705a - Certificação prática

Oficina prática de certificação. Mostradas as regras e os algoritmos, o professor se impõe como autoridade certificadora raiz. A seguir, ele certifica 4 alunos como CA e depois todos os alunos precisam se certificar junto a essa CA. Finalmente, há um desafio-resposta que certifica todos os alunos. Concluído em 01/09/10 com a sequência 265.

708a - Cifra Playfair

O exercício descreve a cifra de Playfair (originalmente proposta pelo conde de Playfair, daí o nome) que usa substituição de grupos de 2 letras. Uma variante que usa 4 cifras é também apresentada. Pede-se que o aluno faça uma criptografia e duas decryptografias. Concluído em 23/07/11 com a sequência 286.

709a - Cifra de Viginère

Apresenta a história do algoritmo e mostra como ele é utilizado. O exercício é bastante simples, adequado a platéias com pouca ou nenhuma habilidade na área de criptografia. Concluído em 13/09/10 com a sequência 266.

710a - Criptografia simétrica: DES

Este exercício simula com bastante suavidade o modelo de criptografia DES (Data Encryption Standard). Os alunos devem criptografar e decryptografar uma mensagem. Concluído em 10/05/01 com a sequência 020.

711a - Algoritmo TEA

Apresenta o algoritmo TEA, desenvolvido na Universidade de Cambridge em 1994, através de sua codificação em C. Pede que o aluno decryptografe uma dada mensagem. Concluído em 20/08/10 com a sequência 263.

712a - Criptografia assimétrica: RSA

O exercício começa mostrando como elevar números grandes (mais de 100 dígitos) a potências igualmente grandes. Depois o algoritmo RSA é mostrado. O exercício mostra a chave pública de 6 pessoas: Alice, Bento, Carlos, Dante, Erasmo e Frida e dá a chave pública e privada do aluno. A seguir 6 perguntas envolvendo todos os aspectos da criptografia RSA são feitos. Este exercício tem que ser feito usando computador. Concluído em 03/04/01 com a sequência 021.

713a - Esteganografia

É a ciência (arte?) de esconder mensagens em local inesperado. O exercício descreve como a Al Qaeda usou esta técnica para disseminar informações. O aluno recebe um arquivo BMP e é convidado a descobrir onde está escondida a mensagem e qual é ela. Este exercício precisa ser feito em computador. Concluído em 03/04/02 com a sequência 038.

714a - Autenticação de arquivos

Descreve como uma função hash especial pode analisar o conteúdo de um arquivo e assiná-lo (ou seja certificar o seu conteúdo). Um algoritmo similar ao MD5 é mostrado e os alunos são convidados a segui-lo, assinando ao final um pequeno conteúdo. Concluído em 14/10/00 com a sequência 059.

715a - Enigma

A máquina nazista de codificação de mensagens é mostrada através dos seus componentes e modo de funcionamento. O trabalho de Allan Turing na sua “quebra” é descrito. Uma determinada mensagem é mostrada e a seguir codificada. Finalmente, os alunos recebem uma máquina e uma determinada regulagem e devem decodificar 4 caracteres. Concluído em 08/01/03 com a sequência 100.

716a - Quem quebra o RSA ?

Este exercício visa vivenciar a dificuldade esperada na quebra do algoritmo RSA de criptografia assimétrica. Embora usando primos entre 40 e 70, ainda assim os alunos precisam usar computador para resolver este problema. Os algoritmos de força bruta são mostrados e o mecanismo da quebra é estudado. Uma mensagem criptografada é entregue e os alunos convidados a decryptografá-la sem saber a chave que deve ser usada. Concluído em 09/01/03 com a sequência 101.

717a - Exemplos históricos: one way e ADFGVX

Primeiro mostra-se o algoritmo one-way e a história da cifra de Beale (conforme descrita por Sing, S em O livro dos códigos). Depois a cifra ADFGVX usada na primeira grande guerra, cuja descoberta mudou o resultado de uma batalha entre franceses e alemães. Os alunos são convidados a decodificar uma mensagem ADFGVX. Concluído em 03/09/03 com a sequência 116.

718a - Criptografia El Gamal

Este método baseado no problema do Logaritmo Discreto foi proposto para assinar conteúdos, mas também pode ser usado para criptografar coisas. O algoritmo é mostrado e os alunos convidados a decryptografar uma mensagem cifrada com ele. Concluído em 21/05/04 com a sequência 119.

719a - Esteganografia linear

Apresenta o esquema esteganográfico proposto por Andrew Tanenbaum, em seu livro *Redes de Computadores*, pág 876 da 17^o tiragem. Nele uma imagem BMP true-color tem os bits menos significativos de cada cor usados para codificar uma mensagem em ASCII. A folha mostra um exemplo onde a imagem de um gato teve o Hino Nacional Brasileiro codificado. O exercício gera um arquivo contendo uma imagem inofensiva e que tem uma mensagem codificada. Cabe ao aluno achar a mensagem. Concluído em 14/12/10 com a sequência 269.

720a - Aplicativos comerciais: Contabilidade

Neste exercício são apresentadas as principais idéias associadas à contabilidade. O conjunto de ciclos de processamento associados, os sistemas correlatos e uma rápida explicação do sistema. Para o aluno é apresentado um pequeno plano de contas e um arquivo de lançamentos associado a ele. Pede-se que o aluno calcule o saldo daquele balanço. Concluído em 19/12/08 com a sequência 228.

721a - Aplicativos comerciais: Folha de Pagamento

Neste exercício são apresentadas as principais idéias associadas à Folha de Pagamento (simplificada). O aluno recebe uma lista de 10 empregados e deve calcular os principais valores a pagar pela empresa. A correção é feita sobre 3 valores:

- O Imposto de renda a recolher,
- O salário família que a empresa deve pagar aos empregados e
- A despesa total da empresa.

Concluído em 20/12/08 com a sequência 229.

723a - Aplicativos comerciais: Controle de Estoque

Este exercício descreve algumas idéias da Teoria de Estoques e depois de apresentar o estoque de uma empresa fictícia (composto de 20 produtos) solicita que o aluno:

- Construa a distribuição ABC deste estoque
- Calcule o custo total de reposição de um item do estoque em caso da reposição ter um ciclo qualquer (também dado pelo problema)
- Calcule a quantidade de aquisições em um ano se a política for a do “lote econômico”.

Concluído em 30/12/08 com a sequência 231.

731a - Algoritmo Lempel-Ziv e Welch (LZW)

Proposto em 1977 e melhorado em 1984 este é um dos bons algoritmos de compressão conhecidos. Usado no padrão GIF de imagens é genérico e pode comprimir qualquer conteúdo. Os algoritmos de ida e volta são mostrados e os alunos convidados a fazer os 2 trajetos com pequenos conteúdos. Concluído em 20/10/03 com a sequência 034.

732a - Fractal

O conceito de fractal é mostrado, bem como os conceitos correlatos: geometria com números imaginários, caos, atratores caóticos e transformada fractal. Depois é estudado com algum detalhe o fractal de Mandelbrot. As regras para colori-lo são estabelecidas e o aluno convidado a colorir uma pequena parte do fractal. Este exercício exige o uso do computador. Concluído em 03/04/99 com a sequência 029.

735f - Cálculo Numérico-Sistemas Lineares

Descreve 2 métodos para resolução exata de sistemas lineares (Escalonamento de Gauss e matrizes U e L). Pede que o aluno resolva por ambos métodos um sistema linear de 5 equações e 5 inógnitas. Soluções em Freemat. Concluído em 27/08/16 com a sequência 511.

735p - Cálculo Numérico-Sistemas Lineares

Descreve 2 métodos para resolução exata de sistemas lineares (Escalonamento de Gauss e matrizes U e L). Pede que o aluno resolva por ambos métodos um sistema linear de 5 equações e 5 inógnitas. Soluções em Python. Concluído em 24/03/18 com a sequência 511.

750a - Tomografia computadorizada axial

A história da descoberta do Raio X por Roentgen é comentada e a evolução em direção à tomografia também. O princípio de funcionamento de uma TCA é estabelecido, o algoritmo exposto e a seguir uma simulação simples (3 caixas de dinamite dispostas em uma matriz de 11×11 com 6 tomadas de Raio X circulares) é proposta. Cabe ao aluno indicar onde estão as bananas de dinamite tendo acesso apenas aos dados numéricos da tomografia. Concluído em 03/11/00 com a sequência 061.

752a - Manuseio de imagens BMP

O padrão BMP (bit mapped) é descrito, bem como as diversas modalidades (P&B, cores mapeadas, *true color images*,...) são descritas. A seguir uma análise interna no conteúdo do arquivo é proposta. Os alunos recebem o *dump* de uma imagem e devem descrevê-la sem poder olhá-la como imagem, apenas como arquivo. Ao final, eles recebem a imagem monocromática e podem estabelecer se acertaram a análise ou não. Concluído em 03/03/99 com a sequência 030.

753a - BMP coloridos

Neste exercício, o conteúdo de uma imagem colorida (mapeada) é mostrado ao aluno e perguntam-se coisas sobre ele. A resposta deve ser oferecida sem poder olhar para a imagem, o que só é feito após o aluno ter respondido ao questionário da folha. Concluído em 20/04/99 com a sequência 044.

754a - Proc. e análise de imagens: convoluções

Algumas informações sobre o tratamento aritmético de imagens digitais são estabelecidos como premissa. A seguir as operações de melhoramento de imagens orientadas à vizinhança são dados. A operação de convolução é descrita. Os alunos são convidados a aplicar 3 métodos a uma mesma imagem e a responder perguntas sobre os resultados alcançados. Concluído em 30/03/00 com a sequência 035.

755a - Proc. e análise imagens: equalização de histogramas

Esta técnica modifica as “cores” de uma imagem, modificando o contraste da imagem. Os algoritmos desta técnica e de algumas similares são estabelecidos e o aluno convidado a manipular aritmeticamente uma pequena imagem obtendo outra mais contrastada. As perguntas a responder referem-se a esta segunda imagem. Concluído em 10/05/00 com a sequência 037.

756a - Padrão GIF

O padrão GIF é descrito e a lógica de seu funcionamento é comentada. Diversas animações são construídas e desconstruídas na sala de aula. O exercício é localizar as características da imagem a partir de um *dump* de um arquivo GIF distribuído aos alunos. O exercício é em equipe. Concluído em 28/08/00 com a sequência 056.

760a - Padrão FITS

Descreve o padrão de imagens FITS e mostra como recuperar as informações nele armazenadas. O aluno recebe um arquivo FITS e deve interpretá-lo. Concluído em 31/03/17 com a sequência 535.

763b - Redes de Petri: definição

São descritas as redes de petri do tipo condição/evento. As regras de marcação e disparo são estabelecidas. Uma rede aleatória é fornecida ao aluno e este convidado a descobrir e descrever suas características. Concluído em 05/05/00 com a sequência 031.

763c - Redes de Petri: hardware de um semi-somador

Um circuito de hardware (um semi-somador binário) é estabelecido através de uma rede de petri do tipo condição evento. Ele é exhaustivamente simulado e operado. Os alunos são convidados a predizer seu comportamento a partir de condições iniciais diferentes para cada um. Concluído em 05/10/00 com a sequência 032.

763d - Redes de Petri: modelando uma ATM

Uma máquina bancária (24hr) conhecida como Automatic Teller Machine é descrita e sua operação modelada através de uma rede de petri do tipo condição evento. Os alunos são instigados a dizer o que ocorrerá em certas situações e convidados modificar a rede a partir de novas funcionalidades introduzidas no processo. Concluído em 06/07/01 com a sequência 036.

766a - Técnicas de compressão - parte 1

São apresentadas 5 técnicas de compressão e treinadas 3 delas: O código Morse (interesse apenas histórico), a técnica de compressão de fileira, que é bastante útil, pois fácil e com bastante potencial de uso, e a técnica de huffman, que já foi aprendida no segundo ano. Lá o interesse era com a árvore de huffman e aqui é com a compressão propriamente dita. A técnicas que não é treinada, apenas apresentada é a de substituição de padrões. Concluído em 14/07/07 com a sequência 182.

767a - Técnicas de compressão - parte 2

São apresentadas 4 técnicas de compressão e treinadas 3 delas: A codificação relativa, a compressão de Shannon- Fano, os códigos de vírgula e a compressão auto-adaptável. 3 mensagens distintas, cada uma delas comprimidas com uma das técnicas são apresentadas aos alunos, pedindo-se a eles que as descomprimam. Concluído em 17/07/07 com a sequência 183.

768a - Algoritmos de Compressão

Junta em um único exercício as questões principais das folhas 766 e 767, de maneira a poder apresentar este tópico em uma única aula. Concluído em 26/08/10 com a sequência 264.

770a - Teoria de filas: generalidades

O tratamento básico de filas através de ferramentas matemáticas é estabelecido. Ambientes de servidor único ou multiservidores, fila única ou múltiplas filas, bem como os conceitos de ocupação e taxas de atendimento são apresentados. O aluno é convidado a modelar o funcionamento de um estabelecimento comercial (diferente para cada aluno) com suas especificidades. Cabe a ele recomendar qual a sugestão de implementação mais adequada. Concluído em 18/09/05 com a sequência 139.

772a - Teoria de filas: simulação *ad hoc*

Feito logo após o exercício 770, neste se descreve como e porque as ferramentas analíticas lá vistas nem sempre podem ser usadas. A simulação como método de trabalho é sugerida e seus contornos discutidos. O aluno é convidado a resolver dois problemas através da simulação: a quantificação de recursos em uma nova agência bancária e o desempenho esperado de um despachante de tarefas em um sistema operacional qualquer. Concluído em 19/09/05 com a sequência 033.

775a - Teoria de filas: uso do simulador ARENA

Após ter feito os exercícios 770 (ferramentas analíticas da teoria) e 772 (simulação *ad hoc*) o aluno é convidado a conhecer e usar o software ARENA, que conta com versões para estudante na modalidade freeware. 5 problemas simples são pedidos aos alunos e devem ser resolvidos usando computadores com este software. Concluído em 11/10/05 com a sequência 140.

780a - Números randômicos

É comentada a importância do processo de geração de números (pseudo) aleatórios. São descritos os métodos de metade do quadrado, de Von Neumann, o método de Fibonacci e o método Congruencial Linear. Aos alunos é pedido a geração de um conjunto de aleatórios usando cada um dos métodos. Concluído em 17/01/02 com a sequência 079.

781a - Listas skip

Descreve a estrutura lista skip, que ao usar números randômicos, garante um desempenho $O(\log n)$ na busca, independentemente da distribuição de chaves na lista, ao contrário da busca binária, por exemplo. O exercício oferece uma lista skip e uma chave de busca e pede quantas interações foram feitas. Baseado em Goodrich e Tomássia, pág 195. Concluído em 06/02/03 com a sequência 105.

782a - Simulação: porta da esperança e questões de prova

Duas situações reais em que o computador pode e deve ser acionado para responder a perguntas que são muito mais difíceis de responder apenas usando a teoria matemática. Na primeira especula-se sobre o melhor comportamento no programa de TV chamado porta da esperança. Na segunda descobre-se o tamanho médio de um banco de questões para a prova do atual EJA, antigo exame supletivo. Concluído em 01/11/11 com a sequência 306.

785a - O problema da parada

Apresenta 3 abordagens para o problema da parada, baseado nos autores

- John MacCormick (9 algorithms that changed the world)
- Behrouz Forouzan e Firouz Mosharraf (Fundamentos da Ciência da Computação) e
- Walter Fuchs (Matemática Moderna)

Concluído em 09/08/21 com a sequência 664.

790a - Processamento Fonético

Descreve os princípios do processamento fonético de chaves. Este algoritmo é usado quando se pretende buscar itens dentro de banco de dados, não a partir de seus códigos e sim a partir de seus nomes, acertando-se neste caso as semelhanças fonéticas de nomes escritos diferentes. Por exemplo, SMITD, SCHMIT, SCHIMDT e similares podem dar origem à mesma chave. No exercício duas frases do poema VOU ME EMBORA PRA PASÁRGADA são dadas ao aluno e pedem-se as chaves fonéticas usando dois algoritmos distintos. Concluído em 26/11/08 com a sequência 226.

798a - Pós Graduação: um projeto de vida

Um roteiro para a palestra sobre o funcionamento da pós graduação no Brasil e no mundo. Não tem exercícios e logo não deve ser devolvida pelos alunos. Lista os principais cursos em nosso entorno e dá as principais informações. É uma tentativa de “apaixonar” os alunos pela continuação do estudo. Concluído em 25/07/09 com a sequência 245.

799a - Perdidos na Lua

Simulação de uma decisão em grupo. Primeiro cada aluno responde um questionário. Depois, a sala responde o mesmo questionário agora em conjunto. Finalmente, as respostas são comparadas com o gabarito correto fornecido pela NASA. Sempre, as respostas coletivas tem um resultado numérico melhor do que a média das respostas individuais. Baseado no livro How To Make Meetings Work, de DOYLE, Michael e STRAUS, David. Cada cópia deste material gera 4 kits. Concluído em 18/02/06 com a sequência 152.

800a - Pode uma máquina pensar?

Este exercício é uma atividade coletiva na sala analisando o texto de Allan Turing “can a machine think” de 1950, no qual as bases da Inteligência Artificial são estabelecidas. 4 perguntas são feitas e devem ser coletivamente respondidas pela turma, após reflexões obtidas a partir do texto. Concluído em 16/07/05 com a sequência 136.

802a - Máquina de Turing

A máquina de Turing é descrita e alguns resultados são obtidos usando-se um simulador *ad hoc*. Uma máquina é proposta e o seu trabalho simulado até a obtenção dos resultados esperados. Finalmente, uma máquina é proposta a cada aluno, com estado e entradas e pergunta-se o que ocorre. Concluído em 30/01/03 com a sequência 104.

805a - Análise do artigo *Can a machine think?*

A partir do artigo do Allan Turing, esta folha divide a sala em 6 equipes e organiza o trabalho de análise. Concluído em 23/11/05 com a sequência 142.

807a - Computadores auto replicantes

Descreve as pesquisas de Turing, Von Neumann, Codd, Langton e John Byl a respeito de morfogênese e Computadores Construtores Universais (UCCs). Ao final o aluno é convidado a simular 2 instâncias do autômato auto-replicante de John Byl. Concluído em 03/02/15 com a sequência 467.

809a - Definições da Inteligência Artificial

O exercício começa listando 14 definições de autoridades em IA. A sala é dividida em equipes e existe um roteiro estabelecido para discutir as definições e construir uma definição coletiva. A ênfase está nos aspectos antropomórficos das definições e na dificuldade com a palavra inteligência. Concluído em 15/01/03 com a sequência 103.

810a - Diagrama PAGE para sistemas de IA

Descreve o conceito de agente (visão moderna da IA) e solicita que o aluno escreva 2 diagramas PAGE para dois sistemas escolhidos ad-hoc para o aluno. Concluído em 07/03/13 com a sequência 374.

814a - O mundo dos aspiradores autônomos de pó

Este exercício originalmente citado no livro do Russel, pede aos alunos que escrevam o diagrama PAGE (perceptor-ação-objetivo-ambiente) para este dado problema. Uma proposta de solução é apresentada e os alunos devem simulá-la em uma casa apresentada também comentando o desempenho dos robôs. Concluído em 05/03/03 com a sequência 106.

815a - Sistemas de produção

O exercício começa descrevendo um sistema de produção e mostrando como este paradigma equivale a uma máquina de Turing (ou seja a um computador funcional). O ciclo reconhece-atua é descrito e os alunos convidados a especificar um de 5 sistemas de produção para resolver um problema. Concluído em 03/02/05 com a sequência 129.

817a - Construção de um sistema especialista

Os alunos são divididos em equipes de até 4 alunos e cada equipe deverá usar um shell de SE (o EXSINTA, da Universidade Federal do Ceará), e a partir de um domínio conhecido produzir um SE. Caso não haja nenhum domínio à disposição, o exercício sugere 3: um sistema para escolha de vinhos, um para tirar boas fotos e um terceiro para fazer boas viagens de turismo. Concluído em 24/05/05 com a sequência 132.

819a - Sistemas Especialistas

Este exercício implementa um algoritmo bem simples (devido a Herbert Schildt) para resolver uma busca em base de conhecimentos. O exercício implementa uma base de cidades e suas características guiando o aluno em pesquisas feitas sobre a base. Concluído em 28/03/02 com a sequência 086.

820a - Aplicação real do A*: cubo Rubik

O exercício analisa as dificuldades (o tamanho do espaço de estado) do cubo de Rubik. Os alunos são convidados a especificar um programa que resolva o cubo, usando técnicas do algoritmo A*. Concluído em 03/01/05 com a sequência 127.

821a - Busca cega em espaço de estados

O objetivo deste exercício é vivenciar a dificuldade de pesquisar às cegas em um espaço de estados, mesmo pequeno. O aluno deve trabalhar 5 problemas (criptoaritmética, menor caminho, as 8 rainhas, o jogo da velha e o problema do caixeiro viajante) em pequenas instâncias. Cada um deles gera uma pergunta. Concluído em 01/11/04 com a sequência 126.

824a - Busca em árvore de jogos: MINIMAX

O algoritmo MINIMAX é mostrado, e um novo jogo é proposto ao aluno: o jogo do fubá. Ele tem objetivo apenas pedagógico e serve para mostrar o algoritmo em ação. Aos alunos cabe escolher qual jogada fazer em um determinado momento do jogo, lá estipulado. Concluído em 27/01/02 com a sequência 083.

825a - Corte Alpha Beta

Construída uma árvore minimax (exercício 824), este exercício mostra o que faz o corte $\alpha - \beta$ e como ele diminui radicalmente o espaço de buscas. Além do algoritmo básico, 2 árvores já valoradas são dadas ao aluno pedindo-se a ele que aplique o corte $\alpha - \beta$ nestas duas árvores. Concluído em 19/07/05 com a sequência 137.

826b - A*: juncao do 822 e 823 acima

Alterna exercícios 822 e 823 Concluído em 17/01/07 com a sequência 174.

826j - A*: jarras com implementação

Solicita que o aluno implemente o algoritmo A* baseado no jogo de enchimento das jarras. Concluído em 17/03/12 com a sequência 321.

826t - A*: tarkin com implementação

Solicita que o aluno implemente o algoritmo A* baseado no jogo do tarkin. Concluído em 17/03/12 com a sequência 320.

830a - A*: roteiro de um robot

Apresenta um labirinto com obstáculos e paredes e dois pontos (origem e destino) de um robot. Apresenta também a árvore do algoritmo A*. Cabe ao aluno interpretar a árvore, desenhar o trajeto do robot e contar quantas células ele percorre. Concluído em 27/02/14 com a sequência 408.

831a - A*: caminho mais barato

Dada uma matriz 50 x 50 (em meio magnético) pede-se que o aluno calcule o caminho mais barato saindo da célula 1,1 e chegando na 50,50 podendo deslocar-se para qualquer um dos 4 vizinhos de uma determinada célula. Baseado no problema 83 do projeto euler. Concluído em 18/08/14 com a sequência 417.

835a - Xadrez

Apresenta 5 problemas de Xadrez. Dois básicos sobre o jogo e outros 3 sobre estratégias de implementação de um eventual algoritmo minimax que jogue xadrez. Os problemas são dicionarizados. Concluído em 24/03/06 com a sequência 153.

841a - Redes neurais artificiais

Um neurônio artificial é apresentado e estudado. A seguir uma rede simples (8 entradas binárias e 4 saídas) é estabelecida, inicializada e treinada. Após isso, 5 perguntas são feitas sobre ela. Concluído em 10/10/96 com a sequência 050.

845a - Algoritmo de Backpropagation

Mostra como funciona o algoritmo de Backpropagation em uma Rede Neural Artificial, e pede que o aluno calcule a correção dos pesos em uma rede feedforward simples. Concluído em 21/12/11 com a sequência 311.

846a - Número de Neurônios

Trata o problema fundamental em RNAs, a saber o estabelecimento do número ideal de neurônios para resolver dado problema. Trata o compromisso: poucos neurônios $\Rightarrow$ convergência pobre e muitos neurônios $\Rightarrow$ muitas soluções e uso excessivo de computador. Concluído em 23/12/11 com a sequência 312.

847a - Redes Neurais: uma implementação Python

Usa o livro de Redes Neurais disponível em <http://neuralnetworksanddeeplearning.com> e pede que o aluno recupere o código e os dados do MNIST para implementar a rede. Concluído em 29/10/19 com a sequência 638.

848a - Projeto de redes neurais

Um problema hipotético, mas possivelmente real, a respeito de mineração de dados em uma universidade é apresentado. Alunos são acompanhados por 4 anos após sua formatura e classificados em 6 categorias. O modelo pede que o aluno projete uma rede capaz de inferir o desempenho futuro a partir das notas e frequência do primeiro ano. Futuramente, este exercício poderá ser implementado (no JOONE ?). Concluído em 08/05/06 com a sequência 157.

851a - TensorFlow

Descreve a ferramenta Google chamada TensorFlow para uso em Machine Learning. É uma continuação normal das aulas de implementação de Redes Neurais, já que o TF é o estado da arte. Concluído em 25/07/17 com a sequência 544.

856a - Expressões lógicas

Como uma preparação para o tópico “demonstração automática de teoremas (exercício 827) os alunos são convidados a verter para a lógica de predicados 40 frases expressas em português convencional. Cada aluno recebe 40 frases distintas. Concluído em 27/07/04 com a sequência 123.

857a - Demonstração automática de teoremas

O método da demonstração através de refutação é mostrado. O algoritmo da unificação é comentado como base para a demonstração automática (e como suporte à linguagem PROLOG). 5 enunciados (levemente confusos) são apresentados aos alunos, pedindo-se a eles que identifiquem quais teoremas (como são conhecidos os enunciados neste contexto) podem ser provados. O exercício é para ser feito a papel e lápis (e massa cinzenta) mas os alunos que quiserem podem se valer do auxílio de um compilador PROLOG. Concluído em 21/07/04 com a sequência 121.

860b - Mergulho em PROLOG

Alguns exercícios em PROLOG. Começa demonstrando 2 teoremas vistos na aula e no exercício passados, devidamente resolvidos em PROLOG. Ensina a instalar o software e a seguir pede a resolução de algumas assertivas simples. Concluído em 10/06/06 com a sequência 159.

860c - PROLOG: uma introdução

Descreve superficialmente o prolog e apresenta 4 gerações de uma família. O aluno deve descrever os fatos e as regras dessa família. Depois deve responder a 10 questionamentos. Concluído em 28/10/11 com a sequência 304.

860d - Introdução ao Prolog

Junção das folhas 860b e 860c. Concluído em 11/10/13 com a sequência 392.

861a - Exercícios de Prolog 1 de Sílvio Lago

Material de Sílvio Lago, recuperado na Internet. Descreve o Prolog e pede alguns exercícios. Concluído em 06/09/12 com a sequência 347.

862a - Exercícios de Prolog 2 de Sílvio Lago

Material de Sílvio Lago, recuperado na Internet. Descreve o Prolog e pede alguns exercícios. Concluído em 06/09/12 com a sequência 348.

863a - Exercícios de Prolog 3 de Sílvio Lago

Material de Sílvio Lago, recuperado na Internet. Descreve o Prolog e pede alguns exercícios. Concluído em 07/09/12 com a sequência 349.

864a - Exercícios de Prolog 4 de Sílvio Lago

Material de Sílvio Lago, recuperado na Internet. Descreve o Prolog e pede alguns exercícios. Concluído em 07/09/12 com a sequência 350.

865a - Exercícios de Prolog 5 de Sílvio Lago

Material de Sílvio Lago, recuperado na Internet. Descreve o Prolog e pede alguns exercícios. Concluído em 07/09/12 com a sequência 351.

866a - Exercícios de Prolog 6 de Sílvio Lago

Material de Sílvio Lago, recuperado na Internet. Descreve o Prolog e pede alguns exercícios. Concluído em 07/09/12 com a sequência 352.

869a - Linguagens da IA

Descreve LISP e PROLOG e convida o aluno a instalar e usar duas versões *freeware* de ambas. Fugindo do padrão esta folha ocupa 2 páginas, mas foi o que se pôde fazer para resumir algo tão rico. Concluído em 29/08/17 com a sequência 550.

870a - Ambientação em LISP

Uma coleção de 77 exercícios a fim de ambientar os alunos com o ciclo leia-calcule-imprima do LISP. São vistos o conceito de predicado, de listas, de car, cdr e cons, list, append, condicionais e definição de funções pelo usuário. Concluído em 04/08/05 com a sequência 138.

871a - Definição de funções em LISP

Define as regras para a criação de funções (programas) em LISP e pede que o aluno construa 15 programas (funções) em Lisp. Concluído em 26/09/14 com a sequência 431.

872a - Mais funções de LISP

Pede que o aluno construa diversos comandos primitivos de LISP através de funções do usuário. Concluído em 26/09/14 com a sequência 432.

877a - Funções LISP

8 funções LISP (distintas para cada aluno) são mostrados ao aluno, pedindo-se que ele descreva o que elas fazem. A sugestão da folha de exercícios é que o aluno implemente cada função com o que poderá examinar realmente o que a função faz. Concluído em 08/06/03 com a sequência 109.

878a - Funções LISP

Ao contrário do exercício 830, aqui a definição é dada ao aluno, pedindo-se que ele construa 8 funções LISP que atendem ao solicitado. Concluído em 28/08/04 com a sequência 125.

879a - Programando em LISP

Trabalho em equipe, devendo cada uma delas construir um pedaço de um editor de textos de propósito geral em LISP. A junção de todos os trabalhos mostrará uma tarefa comum de programação feita em paradigma funcional. Concluído em 30/06/05 com a sequência 134.

891a - Modalidades de representação do conhecimento

São apresentados os planos de Quillian, os frames de casos de Simmons, os grafos conceituais de Sowa, os quadros e os roteiros. Ao aluno pede-se que escreva um exemplo de cada uma das ferramentas. Concluído em 10/02/05 com a sequência 130.

892a - FASE

FASE é um arcabouço de sistemas especialistas, produzido pelo Dr Guilherme Bittencourt na UFSC, e permite construir instâncias de sistemas especialistas permitindo intercambiar as seguintes representações: lógica de predicados, redes semânticas e quadros, usando uma linguagem comum às três. São pedidos ao aluno 5 trechos de SEs construídos usando o FASE. Concluído em 20/06/03 com a sequência 110.

900a - Operacoes sobre conjuntos

Uma revisão da matemática básica referente à Teoria dos Conjuntos. Contém 38 exercícios diferentes. A resposta por razões de quantidade não obedece ao padrão VIVO. É gerado um arquivo a parte de nome .ctx contendo o gabarito completo na base de 1 página por aluno. Concluído em 11/07/09 com a sequência 243.

901a - Divisibilidade, primos e mat. básica

O conceito de divisibilidade, primos, fatoração de primos, máximo divisor comum, mínimo múltiplo comum e sequencias do tipo progressão aritmética e geométrica, tudo entremeadado com a regra de sinais e algumas coisinhas a mais de matemática básica. Concluído em 14/07/09 com a sequência 244.

902a - Geometria Analítica

Alguns conceitos da Geometria Analítica são apresentados. Distância entre pontos, distância de ponto à reta, área de triângulo e circunferência que passa por 3 pontos, por exemplo. Daí o aluno é provocado a resolver alguns exercícios simples. O objetivo é tratar a matemática básica (polinômios, regra de sinais, ...). Concluído em 06/01/10 com a sequência 255.

903a - Albert Einstein e a T. da Relatividade

São descritas as principais idéias da Teoria Restrita da Relatividade e apresentadas as principais fórmulas. Pede-se que o aluno resolva 6 exercícios baseados nelas. Concluído em 16/07/11 com a sequência 285.

911a - Teorema de Bayes

Descreve o Teorema de Bayes e mostra a sua importância ao analisar probabilidades condicionais e sobretudo como o conhecimento aumenta com a repetição de experiências. O exercício pede para o aluno usar o teorema em duas situações triviais. Concluído em 11/01/15 com a sequência 463.

912a - Prática em probabilidade

Exercício 13.10 do AIMA (Artificial Intelligence: a modern approach) O aluno deve simular uma máquina caça níquel e verificar probabilidades de ganho. Concluído em 31/12/16 com a sequência 531.

913a - Redes Bayesianas de Crença

Descreve o Teorema de Bayes e generaliza-o para as Redes de Crença Bayesianas. Mostra o teorema fundamental de resolução e sugere a implementação de softwares para sua solução. Concluído em 13/09/17 com a sequência 552.

922a - Estereogramas

Este exercício descreve como um estereograma pode “enganar” o cérebro fazendo com que vejamos em 3D uma imagem construída em 2D. A regra é descrita e a seguir pede-se que o aluno localize as mudanças de níveis que existem em um exemplo dado. Concluído em 17/04/05 com a sequência 131.

925b - Algoritmo de localização de regiões

Este algoritmo devido a Horovitz e Pavlidis em 1976, descreve um método em duas etapas para localizar regiões homogêneas em imagens mapeadas ou true color. Ao aluno é dada uma pequena imagem de 8×8 pixels pedindo-se a ele que descreva as regiões localizadas conforme o algoritmo. Concluído em 02/02/05 com a sequência 128.

925f - Análise de imagens

Folha agrupando duas folhas: a 928 e a 925. Concluído em 13/07/13 com a sequência 388.

926a - Operadores morfológicos: erosão, afinamento, esqueletização

Os algoritmos destas transformações são apresentados pedindo-se ao aluno que aplique-as a uma pequena imagem binária de 8×8 e apresente o esqueleto dessa imagem. Concluído em 15/07/05 com a sequência 135.

928a - Visão computacional

Neste exercício é descrito o algoritmo originalmente proposto por Winston para localizar e identificar blocos em um ambiente 3D a partir das imagens em perspectiva em 2D. Uma imagem é dada ao aluno pedindo a ele que identifique arestas, vértices, junções e uniões usando o algoritmo. Concluído em 23/08/03 com a sequência 114.

930a - Listas em Python

Descreve o conceito de lista em Python e dá alguns (os principais) métodos de manipulação. Destaque especial para a operação de fatiamento (slicing). O aluno deve efetuar 25 fatiamentos. Concluído em 18/07/18 com a sequência 589.

931a - Strings em Python

Descreve o conceito de string em Python e dá alguns (os principais) métodos de manipulação. Destaque especial para a operação de fatiamento (slicing). O aluno deve efetuar 10 fatiamentos. Concluído em 19/07/18 com a sequência 590.

932a - Funções em Python

Descreve o conceito de função em Python e mostra como definir e usar. O aluno deve escrever, testar e implementar 10 funções. Concluído em 20/07/18 com a sequência 591.

933a - Arquivos em Python

Descreve o conceito de arquivo em Python e mostra para que servem e como são usados. O aluno deve ler um arquivo contendo uma distribuição e calcular 5 medidas. Concluído em 23/07/18 com a sequência 592.

934a - Tuplas, Dicionários e Conjuntos em Python

Descreve os 3 conceitos em Python e trabalha apenas o último (conjuntos). Diversos conjuntos são dados e o aluno deve fazer diversas operações sobre eles achando a cardinalidade do resultado. Concluído em 24/07/18 com a sequência 593.

935a - Objetos em Python

Descreve o objeto em python e sugere a sua utilização. Pede para o aluno definir um objeto e alguns métodos. Concluído em 18/10/19 com a sequência 637.

936a - Prática em Numpy

Prática em diversas tarefas de cunho matemático, certamente adequadas às disciplinas de cálculo numérico em qualquer curso de engenharias. Concluído em 04/06/19 com a sequência 614.

937a - Gráficos em Python: Matplotlib

Descreve o pacote Matplotlib e mostra algumas (poucas) coisas que dá para fazer com ele. O aluno é convidado a produzir 3 gráficos com distribuições ad hoc. Concluído em 26/07/18 com a sequência 594.

938a - Processamento simbólico: Sympy ou Maple

Prática em solução de equações, integrais definidas e sistemas lineares e não lineares, sempre usando um processador algébrico que pode ser o Maple ou o Sympy. Concluído em 01/06/19 com a sequência 613.

939a - Blocos em Python

Descreve o conceito de bloco em Python e apresenta diversos algoritmos parecidos, mas com sutis diferenças de indentação, logo com blocos parecidos mas diferentes. O aluno deve-os interpretar. Concluído em 31/07/24 com a sequência 740.

942a - Computação evolutiva: algoritmos genéticos

Os princípios da computação evolutiva são apresentados, e a seguir um pequeno problema (a otimização da função $y = x^2$) é seguida etapa a etapa permitindo aos alunos apreender a lógica do processo. São vistos a seleção, a recombinação (ou crossover) e a mutação por bit. Depois, pede-se ao aluno que resolva uma nova instância. Concluído em 20/03/01 com a sequência 072.

943a - Computação evolutiva: um problema real

Dá-se um problema real (a otimização de um processo de produção em uma determinada indústria) que normalmente seria (e é) resolvido usando o método simplex. Uma única passagem de geração é simulada pedindo-se ao aluno que infra os resultados alcançados. São vistos também o método de seleção por torneio, a mutação de real e a recombinação uniforme. Ao final o problema é resolvido também via simplex comparando-se os dois resultados obtidos. Concluído em 25/10/03 com a sequência 117.

945a - Computação evolutiva: programação genética

Apresenta-se outro paradigma evolutivo: o da população aleatória de programas e não de dados como até então. Discute-se a seleção, recombinação e mutação nestes casos. Um robot perdido no meio de um labirinto é conduzido usando um programador genético e o aluno deve informar medidas quantitativas neste processo. Concluído em 02/06/05 com a sequência 133.

947a - Uso do TUDEL

O TUDEL (TURING + menDEL) é um pacote de algoritmos genéticos de propósito geral desenvolvido por P.Kantek ainda como parte de seus estudos no doutorado. Este exercício sugere uma função de calibração (a F6 de Schaffer et alli) e convida os alunos a dar um passeio pela calibragem do modelo, alterando os seletores, os normalizadores, as taxas (de crossover, de mutação, de elitismo), os tamanhos de população e de gerações, tudo perpassado pela semente de geração aleatória. Concluído em 17/07/07 com a sequência 184.

950a - Robótica

Este exercício originalmente proposto por Nils Nilsson visa trabalhar as dificuldades de controlar um robô. Trata-se de uma simulação envolvendo equipes de aluno na sala de aula. Reflexões posteriores à atividade são solicitadas. Concluído em 30/07/04 com a sequência 124.

952a - Transformada de Hough

Descreve a transformada de Hough que visa destacar objetos geométricos parametrizados em imagens digitais e pede que o aluno aplique tal transformada a uma imagem digital limiarizada e localize 4 acumuladores na imagem. Concluído em 25/10/20 com a sequência 651.

953a - Transformada de Fourier

Descreve a transformada de Fourier e mostra sua importância no processamento de imagens digitais. Descreve os algoritmos discreto e rápido da TF. Pede para o aluno implementar em Python ambos os algoritmos. Concluído em 09/11/20 com a sequência 652.

954a - Espaço de Cores

Sugere a necessidade de padronizar a representação de cores. Descreve os espaços RGB (red-green-blue), o YCMK (yellow-cian-magenta-black) e o HSV (hue-saturation-value) e dá as maneiras de conversão de um para o outro. O aluno é convidado a fazer 3 conversões: rgb $\rightarrow$ cmyk, rgb $\rightarrow$ hsv e hsv $\rightarrow$ rgb. Concluído em 23/04/21 com a sequência 653.

965a - Biorritmo

A folha descreve a pseudo-ciência que está por trás dos biorritmos. Eles são ciclos de 23 dias (físico), 28 dias (emocional) e 33 dias (intelectual) aos quais todos os humanos estariam sujeitos. O exercício não coaduna com esta pagaiada, mas sugere um belo exercício de programação: a construção de um programa que desenhe os biorritmos de alguém nascido em uma data determinada. Concluído em 10/11/11 com a sequência 307.

975a - Processamento em linguagem natural

A despeito da dificuldade de tratar a linguagem natural este exercício apresenta 2 enfoques para o problema: através de uma máquina de estados e através de um analisador recursivo descendente. Para ambos enfoques são solicitadas avaliações neste exercício. Concluído em 19/07/04 com a sequência 120.

980a - Guia de apreciação do filme COLOS-SUS

O exercício divide a turma em 6 equipes (azul, vermelha, verde, amarela, cinza e laranja) e depois faz 2 perguntas sobre o filme a cada equipe. Ao final, as equipes devem socializar as suas conclusões. Concluído em 23/03/09 com a sequência 238.

985a - Um artigo científico

Descreve o conceito de artigo científico e pede que o aluno receba um problema e redija uma comunicação científica na solução do mesmo. Concluído em 17/09/19 com a sequência 633.

998b - Mnemônico de números irracionais

Dado um número irracional qualquer, o aluno deve escrever um programa de computador que descubra qual o par de inteiros (primeiro menores que 100 e depois menores que 1000) cuja divisão mais se aproxima do irracional dado. Este é um exercício ecológico, já que economiza papel. Todos os enunciados estão em uma única folha e as respostas também devem ser dadas aí. Concluído em 16/02/11 com a sequência 275.

998c - Algoritmo guloso: problema do troco

Dado um sistema de moedas em um país qualquer, este exercício pede que o aluno calcule o troco mais eficiente (com menor número de moedas). O algoritmo é simples, mas nem sempre responde com eficiência. Concluído em 20/02/11 com a sequência 276.

998d - Criptoaritmética

Exercício ecológico (todas as instâncias na mesma folha) no qual é dada uma soma ou subtração de 15 casas, na qual os dígitos são letras. Cabe ao aluno achar a equivalência entre letra e dígito. Não é nem um pouco trivial. Concluído em 15/10/11 com a sequência 302.

- Listas em geral

Este exercício deixou de ser usado há anos, mas é mantido por ser fonte de boas idéias de implementação. Ele é genérico sobre listas e é muito complexo para ser usado tal como está. Concluído em com a sequência .

a01a - Questões de introdução à SD

A turma é dividida em 6 equipes (verde, azul, amarelo, vermelho, cinza e laranja) e são feitas 2 perguntas a cada equipe. Depois as respostas são socializadas entre todos. As perguntas referem-se as questões introdutórias de sistemas distribuídos. Concluído em 18/05/10 com a sequência 259.

a05a - Questões sobre modelos de SDs

Questionário para resposta em grupo (a sala é dividida em 6 grupos), sobre o tema modelos de sistemas distribuídos. Concluído em 09/06/10 com a sequência 261.

b02a - Questões sobre o Google

Questionário para resposta em grupo (a sala é dividida em 6 grupos). Sobre a empresa GOOGLE, seus produtos e sua estratégia de mercado. Concluído em 26/05/10 com a sequência 260.

b20a - 3 MER feitos e 1 a fazer

Dá uma possível resposta ao MER da locadora de DVD (exercício 884) e mostra também 2 MERs: um para um agendamento de cirurgia e outro para um sistema acadêmico de uma universidade. Finalmente, pede que os alunos desenvolvam um MER para um sistema policial de despacho de viaturas. Concluído em 23/08/11 com a sequência 289.

b27a - Normalização

Descreve os procedimentos para normalizar um modelo de entidades e relacionamentos (ou um esquema de banco de dados), passando pelas formas 1FN (não normalizada), 1FN (primeira forma normal), 2FN e finalmente a 3FN. Pede 3 exercícios de normalização: uma nota fiscal, a matrícula de uma aluno e o agendamento de uma cirurgia. Concluído em 08/09/11 com a sequência 296.

b30a - Questionario: restaurante

Este exercício é uma atividade prática da disciplina de sistemas distribuídos, no qual os alunos são divididos em 6 equipes e é pedido à equipe que projete um sistema distribuído capaz de automatizar as principais tarefas administrativas no restaurante “Comidinha da mamãe”. Concluído em 22/10/10 com a sequência 267.

b40a - Construindo 2 MER: Escolas do Paraná

Exercício baseado em um caso real havido na Secretaria de Educação do PR, quando se contratou o desenvolvimento de um sistema para controle acadêmico, com Recursos do BIRD. O sistema não foi bem sucedido, por razões lá expostas, mas o que se pede aqui é que o aluno construa os 2 MERs da solução: a base local da escola e a base global do Estado. Concluído em 29/08/11 com a sequência 295.

b45a - Engenharia Reversa: construindo o DER

Mostra o esquema físico de um banco de dados (real) e pede que o aluno construa o DER. Concluído em 29/08/11 com a sequência 292.

b50a - Construindo o MER: Passagens aéreas

Exercício extraído de Heuser, descreve um sistema de reservas de trechos aéreos e pede que o aluno construa o MER. Concluído em 29/08/11 com a sequência 293.

b50a - BD: última tarefa

Como um fecho para a disciplina de banco de dados pede-se que o aluno proponha um sistema pessoal (em que ele seja analista e usuário ao mesmo tempo) e descreva as funcionalidades, o modelo ER e o esquema do banco de dados MySQL. Concluído em 29/08/11 com a sequência 293.

b55a - Construindo o MER: Auto locadora

Exercício extraído de Heuser, descreve um sistema de reservas de veículos em uma auto-locadora e pede que o aluno construa o MER. Concluído em 29/08/11 com a sequência 294.

b59a - Última tarefa: construir um MER

O aluno deve criar um sistema começando pelo MER Concluído em 29/10/11 com a sequência 305.

b60a - Criação de um BD simples no ACCESS

Descreve um banco de dados simples com 4 tabelas no ACCESS e comanda a sua criação. Concluído em 23/08/11 com a sequência 290.

b63a - Access-exercícios

Pede a criação de 2 bancos de dados e a criação de consultas sobre esses 2 bancos de dados. Concluído em 15/09/11 com a sequência 297.

d05a - Questionário Introdução a BD

Lista os conceitos iniciais de Banco de Dados e ao final pede para os alunos um arremedo do modelo de um negócio de locação de DVD. Concluído em 08/08/11 com a sequência 287.

d10a - Conceitos de modelo ER

Mostra os principais conceitos do modelo ER (entidade, relacionamento, auto-relacionamento, cardinalidade, atributo e identificador) e depois pede ao aluno que construa um modelo ER do negócio de aluguel de DVD. Concluído em 14/08/11 com a sequência 288.

d12a - Modelagem de 2 DBs: cia aérea e revenda

Este exercício permite a construção de um MER em conjunto (alunos e professor) sobre uma companhia aérea. Depois os alunos são cobvidados a desenvolver um MER de uma revenda de automóveis. Concluído em 07/03/16 com a sequência 492.

d14a - Ferramenta CASE

Mostra os conceitos do dbdesigner4 e do MySQL Workbeench e pede que o exercício 885 seja refeito em um desses softwares. Concluído em 26/08/11 com a sequência 291.

d30b - 10 Documentos para normalizar

O exercício mostra 10 documentos extraídos da Internet e pede que o aluno normalize uma entidade que contenha as informações do documento. Concluído em 21/03/16 com a sequência 493.

d30c - Mais 10 Documentos para normalizar

O exercício mostra outros 10 documentos extraídos da Internet e pede que o aluno normalize uma entidade que contenha as informações do documento. Concluído em 21/03/16 com a sequência 494.

d32a - Engenharia Reversa

Define 2 conjuntos de entidades e relacionamentos através de sua especificação e pede que o aluno desenhe um MER e codifique o BD em SQL. Os dois sistemas são (a) uma venda em um empório informatizado e um sistema de genealogia. Extraído de [Heu09]. Concluído em 08/02/16 com a sequência 490.

d35b - Funções str em Mysql

Descreve algumas das funções de string em MySQL e faz 6 perguntas sobre um arquivo que deve ser lido antes. Concluído em 14/07/16 com a sequência 502.

d35c - Funções de data e hora em Mysql

Descreve algumas das funções de data e hora em MySQL e faz 4 perguntas sobre períodos de tempo variados. Concluído em 15/07/16 com a sequência 503.

d60a - Introdução ao modelo relacional

Descreve informalmente o modelo relacional e as operações de restrição, projeção e junção. O aluno deve realizar diversas dessas operações sobre um mini banco de dados fornecido. Concluído em 25/12/15 com a sequência 482.

d62a - Normalização de relações

Pede para o aluno normalizar 6 relações que estão nas formas $\bar{N}N$, 1FN e 2FN para as formas 1FN, 2FN e 3FN respectivamente. Concluído em 21/01/16 com a sequência 484.

d63a - Integridade Referencial

O exercício mostra um banco de dados de locação de veículos. A seguir, 6 regras de integridade referencial são apresentadas. Um Banco de dados devidamente populado é apresentado e pergunta-se quais destas regras são desobedecidas. Há uma sétima pergunta quato à correção de um determinado cálculo. Concluído em 29/03/16 com a sequência 495.

d64a - Restrições de Integridade

Mostra um banco de dados com 6 relações, devidamente populado e dá 10 restrições formais de integridade. O aluno deve investigar quais dessas restrições são violadas. Concluído em 24/01/16 com a sequência 485.

d66a - MySQL: resumo e exercício

Mostra os principais comandos e cláusulas do MySQL e ao final pede a criação de um banco de dados com alguns arquivos. Concluído em 18/10/11 com a sequência 303.

d69b - Segurança e acesso em MySQL

Descreve os mecanismos de segurança e acesso do MySQL. Concluído em 13/04/16 com a sequência 497.

d70a - Um sistema

Tarefa de final de curso em desenvolvimento de sistemas. Pede a construção de um sistema inédito a gosto do aluno. Concluído em 23/08/24 com a sequência 753.

d71b - Consultas e Respostas: ordenação

Define um Banco de Dados em SQL sobre uma universidade. Daí define 10 consultas e lista os comandos SQL que respondem a cada consulta. O aluno deve ordenar qual resposta atende a qual consulta. Concluído em 06/02/16 com a sequência 486.

d71c - Consultas/Respostas: quais as certas ?

Define um Banco de Dados em SQL sobre uma universidade. Daí define 10 consultas e lista os comandos SQL que podem ou não responder a cada consulta. O aluno deve decidir se a resposta está ou não certa. Concluído em 06/02/16 com a sequência 487.

d75a - Construindo e consultando um bd

O exercício define um banco de dados (6 entidades) contendo regras de integridade referencial e entrega 1200 comandos insert a cada aluno em mídia magnética. Daí, o aluno precisa responder a algumas perguntas sobre os dados. Concluído em 12/04/16 com a sequência 496.

d76a - Estatística Transporte Intermunicipal

Define um banco de dados com 3 entidades: Municípios, Linhas e Viagens e cria instâncias: 50, 100 e 1500 respectivamente. Daí faz 10 perguntas que devem ser respondidas fazendo junções, seleções e ordenações sobre o acervo de dados. Concluído em 30/04/16 com a sequência 498.

d77a - Avaliação de banco de dados

Pede 4 artefatos (diagrama de contexto, MER, definição do banco de dados e alguns SELECTs que respondam perguntas) para diversos aplicativos aí definidos. Concluído em 18/10/16 com a sequência 522.

d80a - Montar DER: organização de congressos

Descreve um sistema de organização de congressos científicos e pede que o aluno desenhe o DER e codifique o BD em SQL. Extraído de [Heu09]. Concluído em 08/02/16 com a sequência 488.

d81a - Montar DER: sistema de almoxarifado

Descreve um sistema de almoxarifado e pede que o aluno desenhe o DER e codifique o BD em SQL. Extraído de [Heu09]. Concluído em 08/02/16 com a sequência 489.

d85a - BD e Contabilidade

Descreve o funcionamento de uma contabilidade do ponto de vista de um banco de dados. Ao final são pedidos 4 valores de um balanço. Concluído em 28/05/16 com a sequência 500.

d86a - BD e Folha de Pagamento

Descreve o funcionamento de uma folha de pagamento do ponto de vista de um banco de dados. Ao final são pedidos 3 valores de um processamento particular. Concluído em 03/08/16 com a sequência 505.

d87a - BD e Controle de Estoque

Descreve o funcionamento de uma gestão de estoque do ponto de vista de um banco de dados. Ao final são pedidos 4 valores de um processamento particular de estoques. Concluído em 29/07/16 com a sequência 504.

d89a - Engenhos em MySQL

Descreve o funcionamento dos engenhos no MySQL. Os tipos de arquivo e suas qualidades/deficiências são mostrados. Ao final, pede que o aluno monte um banco de dados usando 3 tipos diferentes. Concluído em 18/08/16 com a sequência 508.

d90a - Backup e Restore no MySQL

Descreve o processo de backup e sua importância. Alguns cuidados e dicas são mostrados tanto no processo de cópia quanto no da recuperação. O aluno deve carregar um backup entregue a ele e responder a 6 perguntas sobre os dados. Concluído em 10/08/16 com a sequência 506.

d92a - Stored Routines

Descreve o funcionamento dos blocos de código no MySQL. Mostra funcionalidades e exemplos e ao final pede que o aluno escreva uma função armazenada e a execute. Concluído em 15/08/16 com a sequência 507.

d93a - Triggers

Descreve o conceito de trigger e de event scheduler, que vem a ser um trigger temporal. Pede para o aluno carregar um banco de dados e criar 2 triggers, informando ao final qual o valor de variáveis atualizadas dentro dos triggers. Concluído em 24/08/16 com a sequência 510.

d94a - Cursores e Variáveis

Descreve como tratar os registros de uma tabela MySQL linha a linha usando cursores. Pede que o aluno construa uma aplicação prática deste conceito. Concluído em 30/08/16 com a sequência 512.

d95a - MySQL - Replicação

Descreve o conceito de replicação e mostra algumas tecnologias envolvidas bem como a operacionalização do conceito. Concluído em 21/09/16 com a sequência 518.

d97a - MySQL - Otimização

Descreve a importância de otimizar os acessos ao banco de dados e como algumas idéias simples podem melhorar muito os tempos de acesso. Descreve algumas áreas que impactam nos resultados obtidos. Concluído em 21/09/16 com a sequência 519.

d98a - MySQL - Dados Espaciais

Descreve o conceito de dados espaciais e faz alguma especulação sobre o que é possível fazer com eles. Ao final apresenta uma consulta MySQL de 10 restaurantes tailandeses próximos a um local dado. Concluído em 20/09/16 com a sequência 517.

d99a - MySQL e PHP

Descreve o funcionamento do MySQL e do PHP integrados em um servidor HTTP. Mostra um aplicativo simples e pede que o aluno projete um aplicativo com estas ferramentas. Concluído em 20/09/16 com a sequência 516.

e01a - Ambiente de programação e01

O aluno deve programar 4 exercícios buscando em cada um achar um número bem grande que é a resposta ao problema pedido. Concluído em 22/03/13 com a sequência 377.

e03a - Ambiente de programação e03

O aluno deve programar 4 exercícios buscando em cada um achar um número bem grande que é a resposta ao problema pedido. Concluído em 24/03/13 com a sequência 378.

e05a - Ambiente de programação e05

O aluno deve programar 4 exercícios buscando em cada um achar um número bem grande que é a resposta ao problema pedido. Concluído em 31/03/13 com a sequência 379.

e07a - Ambiente de programação e07

O aluno deve programar 4 exercícios buscando em cada um achar um número bem grande que é a resposta ao problema pedido. Concluído em 11/04/13 com a sequência 380.

e09a - Ambiente de programação e09

O aluno deve programar 4 exercícios buscando em cada um achar um número bem grande que é a resposta ao problema pedido. Concluído em 11/05/13 com a sequência 382.

e11a - Ambiente de programação e11

O aluno deve programar 4 exercícios buscando em cada um achar um número bem grande que é a resposta ao problema pedido. Concluído em 18/06/13 com a sequência 385.

e99a - Auto-aprendizado em programação

Esta folha descreve os ambientes de auto-aprendizado em programação. O site onlinejudge da Universidade de Valladolid na Espanha, o site project euler e finalmente o site da sexta brilhante da Universidade Positivo. Concluído em 28/11/13 com a sequência 396.

h07a - Interface via agentes virtuais

4 sites dotados de agentes virtuais são apresentados. O primeiro pede uma simulação do teste de turing. (www.ed.compet.gov.br). Depois 3 sites com agentes virtuais devem ser visitados (ikea.com, www.magazineluiza.com.br e alaskaair.com) finalmente pede-se uma comparação com sites convencionais de compra (www.amazon.com e www.livcultura.com.br). Concluído em 13/11/14 com a sequência 441.

h08a - Interfaces de menus e ícones

Pede que o aluno escreva duas estruturas de menu para um robô humanoide e desenhe 4 ícones para uma máquina sofisticada e 2 ícones para uma máquina simples. Concluído em 21/11/14 com a sequência 442.

h16a - Metodologia de avaliação de sites

Descreve-se uma metodologia para avaliar sites em 10 critérios. A seguir são listados 5 sites reais e o aluno é convidado a avaliá-los segundo a metodologia apresentada. Concluído em 12/11/14 com a sequência 440.

h20a - Tutorial HTML 1

Baseado nos tutorais de <http://www.w3schools.com/> HTML: de Home até atributos. Trata dos conceitos iniciais. Pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 22/12/14 com a sequência 446.

h22a - Tutorial HTML 2

Baseado nos tutorais de <http://www.w3schools.com/> pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 22/12/14 com a sequência 447.

h24a - Tutorial HTML 3

Baseado nos tutorais de <http://www.w3schools.com/> pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 23/12/14 com a sequência 448.

h26a - Meu primeiro site

Pede que o aluno construa um site para um tema específico com pelo menos 4 tópicos e 4 fotos. Concluído em 19/12/14 com a sequência 444.

h28a - Erros em documentos HTML

Descrevem-se 4 erros possíveis em documentos HTML. Daí 6 documentos (contendo um destes erros) são gerados. O aluno deve identificar qual erro eles contém. Concluído em 11/11/14 com a sequência 439.

h29a - Vamos escrever ?

Um exercício de escrita criativa. Concluído em 03/01/15 com a sequência 460.

h30a - Meu segundo site

Pede que o aluno construa um site para um tema específico com pelo praticamente todas as tags estudadas. Concluído em 19/12/14 com a sequência 445.

h32a - Tutorial HTML 4

Baseado nos tutorais de <http://www.w3schools.com/> pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 24/12/14 com a sequência 449.

h34a - Tutorial HTML 5

Baseado nos tutorais de <http://www.w3schools.com/> HTML de tabela até listas. Pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 25/12/14 com a sequência 449.

h35a - Tabelas em HTML

Apresenta a geração de tabelas em HTML (com e sem CSS) e depois pede 3 tabelas onde ocorrem `colspan` e `rowspan`. Concluído em 29/07/15 com a sequência 472.

h36a - Tutorial HTML 6

Baseado nos tutorais de <http://www.w3schools.com/> HTML: de blocos até Iframes. Trata blocos, classes, lay-out responsivo e Iframes. Pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 26/12/14 com a sequência 450.

h38a - Tutorial HTML 7

Baseado nos tutorais de <http://www.w3schools.com/> HTML formulários. Pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 26/12/14 com a sequência 450.

h40a - Tutorial HTML 8

Baseado nos tutorais de <http://www.w3schools.com/> pede que o aluno resolva diversos exercícios e entregue tudo para o professor. Concluído em 26/12/14 com a sequência 450.

h50a - Tutorial CSS 1

Baseado nos tutorais de <http://www.w3schools.com/> CSS de Home até Fonts. Trata sintaxe, seletores, background e texto. Concluído em 27/12/14 com a sequência 451.

h51a - Introdução a CSS

Introduz o conceito de folha de estilos CSS associada a um arquivo HTML. Pede 13 exercícios quase triviais. Concluído em 28/04/24 com a sequência 690.

h52a - Tutorial CSS 2

Baseado nos tutorais de <http://www.w3schools.com/> CSS de Links até positioning. Trata links, listas, tabelas, o modelo *box* e suas dimensões, display e posicionamento. Concluído em 28/12/14 com a sequência 452.

h53a - Organizando o layout com CSS

Mostra como fazer um site com 3 layouts: 3 temas na vertical e o mesmo site com os temas em formato de colunas, primeiro de mesmo tamanho e depois de 3 tamanhos diferentes. Concluído em 29/04/24 com a sequência 691.

h54a - Tutorial CSS 3

Baseado nos tutorais de <http://www.w3schools.com/> CSS de Floating até seletores de atributo. Trata float, pseudo-classes e pseudo-elementos, opacidade, tipos de mídia e seletores por atributo. Concluído em 29/12/14 com a sequência 453.

h55a - Uso de DIV em CSS

Expande o conceito semântico de DIV dentro do CSS multicolumnado. Concluído em 01/05/24 com a sequência 692.

h56a - Uso de bibliotecas padrão em CSS: Bootstrap

Uma rápida descrição da bootstrap que é uma das mais usadas bibliotecas de HTML, CSS e Javascript. Concluído em 12/05/24 com a sequência 702.

h60a - Tutorial JS 1

Baseado nos tutorais de <http://www.w3schools.com/> JS de home até statements. Trata funcionalidades básica do JS, suas saídas, operadores e variáveis. Concluído em 29/12/14 com a sequência 454.

h61a - JS como linguagem de desktop

Descreve como Javascript pode ser usado como uma linguagem de propósito geral em um desktop. Concluído em 22/05/24 com a sequência 708.

h63a - Tutorial JS 2

Baseado nos tutorais de <http://www.w3schools.com/> JS de comentários até escopo. Trata variáveis, arrays, objetos, funções, propriedades e métodos. Concluído em 30/12/14 com a sequência 455.

h66a - Tutorial JS 3

Baseado nos tutorais de <http://www.w3schools.com/> JS eventos e daí vai para o JS DOM: de DOM introdução até DOM eventos. Trata eventos e daí no DOM: métodos, documentos elementos, HTML, CSS e eventos. Concluído em 31/12/14 com a sequência 456.

h69a - Tutorial JS 4

Baseado nos tutorais de <http://www.w3schools.com/> JS DOM de eventos até navegação. Trata eventos no DOM e vai até navegação na árvore DOM. Concluído em 01/01/15 com a sequência 457.

h72a - Tutorial JS 5

Baseado nos tutorais de <http://www.w3schools.com/> JS DOM de navegação, nodes e nodelist até alguma coisa do BOM. Trata da criação de elementos DOM, remoção e substituição. Janelas (BOM) e intervalos de tempo. Concluído em 01/01/15 com a sequência 458.

h77a - Um site

Tarefa de final de curso em programação front-end. Pede a construção de um sistema a gosto do aluno. Concluído em 23/08/24 com a sequência 754.

h81a - Imagem GIF

Descreve o padrão GIF e pede que o aluno localize informações dentro do arquivo de uma imagem animada. Concluído em 14/06/24 com a sequência 717.

h82a - Espaço de cores em JS

Descreve os espaços de cores RGB, CMYK e HSV e dá os algoritmos que permitem transformar uns noutros. O aluno deve fazer estas conversões em JS (e HTML e CSS). Concluído em 17/06/24 com a sequência 721.

h83a - Caça níqueis em HTML+CSS+JS

Pede o desenvolvimento de um caça níqueis em HTML e programação associada. São 4 rolos, cada um com 4 símbolos. Oferece um esqueleto ao aluno e este terá que complementar o aplicativo. Concluído em 13/07/24 com a sequência 735.

h84a - Conversões em HTML+CSS+JS

Pede o desenvolvimento de um aplicativo em JS capaz de converter qualquer número de base $Q \leq 36$ para base decimal e vice-versa, além de resolver adições e subtrações em qualquer uma dessas bases. Concluído em 15/07/24 com a sequência 738.

h85a - Folha de Pagamento em JS

Pede o desenvolvimento de um aplicativo em JS capaz de calcular o salário líquido de um trabalhador brasileiro sujeito aos descontos da Previdência Social e do Imposto de Renda. Pede que o aluno calcule 2 casos. Concluído em 18/07/24 com a sequência 739.

h86a - Varejão em JS

Pede o desenvolvimento de um aplicativo em JS capaz de calcular o desvio padrão de uma amostra e os divisores de um número. Pede que o aluno calcule 2 casos. Concluído em 31/07/24 com a sequência 741.

h87a - 3 Cálculos elementares em JS

Pede o desenvolvimento de um aplicativo em JS capaz de calcular Um termo genérico de Fibonacci, a soma de duas frações (com os cálculos de mmc e mdc e as raízes de uma equação de segundo grau usando a fórmula de Baskhara. Pede que o aluno calcule os 3 casos. Concluído em 02/08/24 com a sequência 742.

h88a - Funções inventadas apoiadas em JS

Baseada na folha 002c, mas conduzindo o aluno a definir as funções inventadas em JS. Enfoque na função `eval` e no comando `try`. Concluído em 07/08/24 com a sequência 744.

h89a - Operações com datas

Baseada na folha 034, mas conduzindo o aluno a definir as funções em HTML/CSS e JS. Pedem-se o intervalo em dias entre 2 datas e dada uma data e um período de dias, qual a data destino. Concluído em 16/08/24 com a sequência 748.

h90a - Exame final de Design: site

Pede que o aluno construa um site de tema específico, possivelmente desconhecido pelo aluno, exigindo: interação via Javascript, uso de estilos CSS entre outros. Concluído em 22/10/15 com a sequência 475.

h91a - Máquina de troco

Baseada na folha 998, mas conduzindo o aluno a definir as funções em HTML/CSS e JS. Pedem-se uma composição de troco a partir de uma coleção de moedas e de um valor dado. Concluído em 17/08/24 com a sequência 749.

h92a - Criptograma letras e números

Baseada na folha 998, mas conduzindo o aluno a definir as funções em HTML/CSS e JS. Pede-se uma solução para um criptograma alfabético. Concluído em 18/08/24 com a sequência 750.

h93a - Leitura de constantes em JS

Baseada na folha 051, oferece ao aluno duas variáveis de 240 e 180 valores e pede que ele calcule coisas sobre elas. As variáveis são oferecidas em papel e em mídia magnética. Concluído em 19/08/24 com a sequência 751.

h94a - Diagonais da matriz em JS

Baseada na folha 072j e no problema Euler 28, oferece ao aluno a dimensão (ímpar) de uma matriz construída segundo regras descritas. O aluno precisa achar a soma das diagonais. Concluído em 21/08/24 com a sequência 752.

h95j - Rodada de poker em JS

Gera 200 rodadas de 2 jogadores de poker e pergunta em quantas delas o jogador 1 ganha. Publicados estão os códigos JS que resolvem o problema. Concluído em 11/09/24 com a sequência 759.

h95p - Rodada de poker em Python

Gera 200 rodadas de 2 jogadores de poker e pergunta em quantas delas o jogador 1 ganha. Publicados estão os códigos Python que resolvem o problema. Concluído em 26/10/24 com a sequência 778.

h96a - Formatos internos em JS

Descreve os padrões de armazenamento em 16 bits (inteiros e flutuantes, positivos e negativos). Está baseado na folha VIVXQ32, mas aqui vai além, pedindo que o aluno implemente programas para as conversões entre os formatos. Concluído em 16/09/24 com a sequência 760.

h97j - Conjuntos em JS

Insta o aluno a programar as tarefas de união, intersecção, diferença e produto cartesiano de 2 conjuntos numéricos. Apresenta duas alternativas: usando o objeto `Set` do JS e também programando em baixo nível tais funções. Concluído em 16/09/24 com a sequência 761.

h97p - Conjuntos em Python

Insta o aluno a programar as tarefas de união, intersecção, diferença e produto cartesiano de 2 conjuntos numéricos. Apresenta duas alternativas: usando o objeto `set` do Python e também programando em baixo nível tais funções. Concluído em 25/10/24 com a sequência 777.

h98a - Mais programas em JS

Dois problemas famosos: qual a fração mais próxima de um real (Baseado no padrão egípcio antigo: $\pi = \frac{22}{7}$ e quantas rotas há em uma treliça (euler:15). Concluído em 17/09/24 com a sequência 762.

h99a - Teorema Fundamental da Aritmética

Descreve este teorema (todo inteiro > 1 é composto por fatores primos. Daí pede a programação em JS de um programa que receba o inteiro e devolva os seus fatores primos. Concluído em 20/09/24 com a sequência 764.

ha1a - Sequência de Collatz

Baseado no problema 14 do projeto Euler, pede o comprimento da sequência de Collatz para um número dado. Concluído em 20/09/24 com a sequência 765.

ha2a - Primos pitagoreanos

Dá um primo pitagoreano (soma de dois números ao quadrado) e pede que o aluno ache os catetos do triângulo retângulo de hipotenusa dada. Concluído em 24/09/24 com a sequência 766.

ha3a - Cálculo do determinante

Pede o cálculo do determinante de uma matriz 3×3 usando o algoritmo de Sarrus desenvolvido em JS. Concluído em 25/09/24 com a sequência 767.

ha4a - Maior palavra

Dada uma frase, o aluno deve escrever um programa em JS que indique o comprimento da maior palavra da frase. Concluído em 25/09/24 com a sequência 768.

ha5a - MDC e MMC em JS

Pede para o aluno programar em JS as funções de máximo divisor comum e mínimo múltiplo comum. Concluído em 26/09/24 com a sequência 769.

ha6a - Algoritmo addition chaining em JS

Descreve o algoritmo (SCH96, pág. 244) e pede a sua implementação em JS. Concluído em 26/09/24 com a sequência 770.

ha7a - Números feios em JS

Descreve a plataforma UVA online, um avaliador de programas da Universidade de Valladolid, na Espanha. Depois pede a implementação do problema 136 referente aos números feios em JS. Concluído em 26/09/24 com a sequência 771.

ha8a - Qual o número com mais divisores ?

Mais um exercício envolvendo a trinca HTML+CSS+Javascript. Dado um intervalo de inteiros, o programa deve achar o número nesse intervalo que tem mais divisores. Concluído em 04/10/24 com a sequência 772.

ha9a - Fatorial como potências de primos ?

Mais um exercício envolvendo a trinca HTML+CSS+Javascript. Dado um número devolver seu fatorial em termos de potências de primos (de acordo com o Teorema Fundamental da Aritmética). Concluído em 05/10/24 com a sequência 773.

hb1a - UVa 105 - Perfil da skyline

A solução do exercício UVa (<https://onlinejudge.org/>) 105 que pede a descoberta da skyline de uma cidade. Concluído em 14/10/24 com a sequência 774.

hb2a - UVa 108 - Maior subsoma em matriz

A solução do exercício UVa (<https://onlinejudge.org/>) 108 que pede achar a maior subsoma dentro de uma matriz. Concluído em 16/10/24 com a sequência 775.

hb4a - UVa 103 - Encaixando caixas

A solução do exercício UVa (<https://onlinejudge.org/>) 103 que pede achar a maior subsequência crescente de uma série de caixas n - dimensionais regulares, dadas por suas dimensões. Concluído em 21/10/24 com a sequência 776.

j10a - Um jogo simples em app Inventor: Palito

Encomenda o desenvolvimento de um jogo simples usando o app Inventor: 2 jogadores tiram 1, 2 ou 3 palitos. Quem tirar o ultimo perde. Concluído em 13/05/24 com a sequência 703.

j12a - Um jogo de Pong no app inventor

Descreve e ajuda a construir um jogo de raquete (Pong) usando o app inventor. Concluído em 18/05/24 com a sequência 707.

j14a - Mini golfe em app inventor

Apresenta um jogo simulando um gramado de golfe. O jogador precisa encaixar bolinhas em buracos. Concluído em 04/06/24 com a sequência 713.

j14a - Mais um BLOB: Tomografia

Descreve o processo de invenção da tomografia e pede que o aluno calcule uma. Baseado no vivxk37. Concluído em 04/06/24 com a sequência 713.

j21a - Jogo de general

Apresenta as regras do general (um jogo de 5 dados baseado no poker) e pede sua implementação no app inventor. Concluído em 01/06/24 com a sequência 711.

j39a - Sudoku para computadores

Apresenta o sudoku e mostra o algoritmo e um programa C++ que o resolve. O aluno deve resolver o sudoku proposto. Concluído em 03/06/24 com a sequência 712.

j42a - app inventor: onde deixei o meu carro ?

Pede o desenvolvimento de um aplicativo de smartfone que memorize uma coordenada geográfica e depois (via google maps) mostre o caminho de outro ponto qualquer até aquele memorizado. Concluído em 14/07/24 com a sequência 736.

j45a - app inventor: xilofone

Pede o desenvolvimento de um aplicativo de smartfone que toque melodias (até 8 notas), e eventualmente memorize para posterior reprodução. Concluído em 14/05/24 com a sequência 737.

j70a - Um jogo

Tarefa de final de curso em jogos digitais. Pede a construção de um jogo inédito a gosto do aluno. Concluído em 23/08/24 com a sequência 755.

j71a - Quebra cabeça 2048

Descreve o quebra cabeça 2048 e pede a sua implementação em App Inventor. A novidade aqui é o uso de uma extensão que contém toda a lógica do quebra cabeça. Concluído em 03/09/24 com a sequência 757.

j72a - Tetris

Descreve o jogo Tetris e pede a sua implementação em App Inventor. A novidade aqui é o uso de duas extensões que contém toda a lógica do jogo e de sua interrupção/retomada. Concluído em 04/09/24 com a sequência 758.

k01a - Introdução a métodos numéricos

Descreve a evolução da disciplina de métodos numéricos e depois descreve possíveis ambientes (hardware+software) de apoio ao desenvolvimento das atividades. Ao final pede que o aluno implemente o Python e seus pacotes. Concluído em 06/12/21 com a sequência 669.

k04a - Sistemas de numeração

Apresenta-se o sistema binário de numeração, seu funcionamento e sua importância na eletrônica digital. Especial ênfase nas conversões (2 para 10 e 10 para 2) de números não inteiros. Concluído em 02/01/13 com a sequência 370.

k07a - Representação de números em máquinas

Descreve lógica de funcionamentos dos padrões IEEE754 para armazenamento de valores em 16, 32, 64 e 128 bits, tanto inteiros quanto ponto flutuante. O aluno é convidado a fazer 6 conversões (2 de inteiros negativos e 4 de flutuantes). Concluído em 13/12/21 com a sequência 670.

k08b - Armazenamento de flutuantes em máquinas binárias

Descreve a teoria por trás do armazenamento de números em máquinas de base binária e da quantidade de dígitos na mantissa e da capacidade do expoente. Pede que o aluno faça 4 simulações. Concluído em 21/10/22 com a sequência 678.

k08g - Armazenamento de flutuantes em máquinas

Descreve a teoria por trás do armazenamento de números em máquinas a depender da base e da quantidade de dígitos na mantissa e da capacidade do expoente. Pede que o aluno faça 4 simulações. Concluído em 12/02/22 com a sequência 672.

k13a - Cálculo Numérico-Solução de equações

Descreve 3 métodos de solução numérica de equações, a saber: Bisseção, método das cordas e método de Newton, oferecendo exemplos e o código em Freemat para os 3. Pede que o aluno resolva uma equação e compare os 3 resultados obtidos. Concluído em 21/08/16 com a sequência 509.

k16a - Zeros de equações via iteração linear, secante e misto

Apresenta estes 3 métodos e pede que o aluno implementa a solução de alguns exercícios. Concluído em 16/12/21 com a sequência 671.

k19a - Calculo numérico - revisão

Resume os métodos de funções (Bissecção, Cordas e Newton) e de solução de sistemas lineares (Gauss, LU e Jacobi) e pede que o aluno resolva um exemplar de cada tipo. Concluído em 15/09/16 com a sequência 514.

k22a - Método de Gauss para Sistemas Lineares

Descreve o método de Gauss e pede ao aluno para resolver um sistema linear com ele. Concluído em 09/07/18 com a sequência 596.

k25a - Sistemas Lineares: método LU

Descreve os processos de obtenção da matriz LU, necessária para resolver um sistema linear: desde uma derivação do escalonamento de Gauss, até o método próprio da LU. O aluno tem que resolver um sistema usando-o. Concluído em 09/07/18 com a sequência 587.

k28f - Sistemas Lineares-Jacobi em Octave

Descreve o método iterativo (logo não exato) devido a Jacobi, também conhecido na literatura como Jacobi-Richardson. O método é descrito e depois o código Freemat é apresentado. Resta ao aluno resolver um sistema linear de 5 equações e 5 incógnitas. Concluído em 01/09/16 com a sequência 513.

k28p - Sistemas Lineares-Jacobi em Python

Um exercício igual ao k28f, mas com o código Python Concluído em 05/04/18 com a sequência 575.

k31a - Gauss-Seidl e Relaxação

Apresenta os dois métodos iterativos de solução de sistemas lineares. Pede para o aluno resolver dois sistemas com 6 e 8 variáveis. Concluído em 02/11/21 com a sequência 665.

k34a - Problemas de Sistemas Lineares

5 problemas reais envolvendo aplicações de sistemas lineares em engenharia elétrica, de produção (logística), química e civil. Concluído em 30/03/18 com a sequência 514.

k37a - Sist Lin: tomografia computadorizada

Descreve o algoritmo de reconstituição da imagem em uma tomografia computadorizada como um sistema linear de 170 equações e 64 incógnitas. Entrega 2 arquivos ao aluno e pede que ele ache características da imagem tomografada. Concluído em 14/04/18 com a sequência 576.

k40f - Interpolação Polinomial - Octave

Descreve o conceito de interpolação polinomial. Apresenta a teoria e depois pede 2 casos à mão (de 2 e 3 pontos) e mais dois casos para fazer com computador (de 5 e 8 pontos). Concluído em 29/09/16 com a sequência 520.

k40p - Interpolação Polinomial - Python

Descreve o conceito de interpolação polinomial. Apresenta a teoria e depois pede 2 casos à mão (de 2 e 3 pontos) e mais dois casos para fazer com computador (de 5 e 8 pontos). Concluído em 16/04/18 com a sequência 577.

k43f - Interpolação - Método de Lagrange/Octave

Descreve o algoritmo de Lagrange para a interpolação polinomial. Apresenta a teoria e depois pede que o aluno resolva um caso. Concluído em 11/10/16 com a sequência 521.

k43p - Interpolação - Método de Lagrange/Python

Descreve o algoritmo de Lagrange para a interpolação polinomial. Apresenta a teoria e depois pede que o aluno resolva um caso. Concluído em 11/10/16 com a sequência 521.

k46a - Interpolação de Lagrange via sympy

Mostra o algoritmo que permite recuperar o polinômio integrador de Lagrange e mostra o código Python que o constrói usando o pacote de computação simbólica `sympy`. 3 Exercícios são pedidos ao aluno. Concluído em 13/11/21 com a sequência 666.

k49a - Polinômio de Newton e Tabela de diferenças divididas

Mostra o algoritmo que permite recuperar o polinômio interpolador mediante o cálculo da tabela de diferenças divididas. Depois o exercício mostra o algoritmo para calcular a interpolação de um ponto sem precisar calcular o polinômio. Ao final pede-se um exercício completo. Concluído em 14/11/21 com a sequência 667.

k52a - Polinômio de Newton-Gregory

Mostra o algoritmo que permite recuperar o polinômio interpolador mediante o cálculo da tabela de diferenças a intervalos constantes. Depois o exercício mostra o algoritmo para calcular a interpolação de um ponto sem precisar calcular o polinômio. Ao final pede-se um exercício completo. Concluído em 15/11/21 com a sequência 668.

k55a - Exercícios de interpolação

14 exercícios de interpolação, dos mais diversos ramos da ciência. Concluído em 08/05/19 com a sequência 612.

k61f - Ajuste de Curvas - freemat

Descreve métodos de ajuste de curvas em particular o de mínimos quadrados. Pede que o aluno resolva 2 problemas, que, a propósito são os mesmos da folha 245. Concluído em 02/11/16 com a sequência 528.

k61p - Ajuste de Curvas - python

Descreve métodos de ajuste de curvas em particular o de mínimos quadrados. Pede que o aluno resolva 2 problemas, que, a propósito são os mesmos da folha 245. Concluído em 30/04/18 com a sequência 528.

k64a - Integração Numérica

Descreve métodos de integração numérica (quadraturas). Pede que o aluno resolva uma integração usando o método do trapézio e o método do $\frac{1}{3}$ de Simpson. Concluído em 26/10/16 com a sequência 526.

k67a - Aplicações práticas de Integração

Pede que o aluno faça 7 exercícios práticos de integração numérica. Concluído em 24/05/18 com a sequência 582.

k80a - Resumo prático de cálculo numérico

Descreve rapidamente alguns métodos (zero via bissecção, falsa posição e newton-raphson; sistemas lineares via escalonamento de gauss, gauss-jordan, LU, matriz inversa, gauss-seidl; interpolação via lagrange, newton e mínimos quadrados; e integração) e pede que o aluno resolva instâncias de cada caso. Concluído em 28/01/22 com a sequência 572.

k90a - Jupyter e markdown

Mostra a utilização do ambiente Jupyter e markdown associado ao Python. Concluído em 12/02/22 com a sequência 674.

m02a - Maratona UP 2011

Os 5 problemas originais da Maratona UP 2011, realizada no dia 3 de maio de 2011. Esta folha não propõe novas instâncias de problemas, apenas lista o material entregue naquela data. Concluído em 09/05/11 com a sequência 279.

m04a - Fazenda de ostras de Zing Zhu

Este problema foi retirado da Maratona de Programação da ACM do ano de 2004. Ele apresenta uma quase-solução, já que resolve apenas algumas instâncias (e não outras). O objetivo do exercício é exatamente discutir a frequência com o profissional encontra esta situação no mundo real, quando nem sempre há um algoritmo conhecido e correto. Existe uma ilha rasa e o seu proprietário estabelece uma série de cercas. Dada uma certa altura da maré, pergunta-se quanto de área da ilha permanece seco. Concluído em 24/09/06 com a sequência 171.

m06a - Universidade Pinguinhos

Problema retirado da Maratona de Programação de 2005 da ACM. Dá-se um curso universitário composto de cerca de 15 matérias com diversas relações de pré-requisitos. Estabelecida a prioridade das matérias, e a quantidade máxima de matérias por semestre, pede-se que o aluno monte a grade proposta. Concluído em 07/10/06 com a sequência 172.

m08a - Piscina/Mágico

Dois exercícios retirados da maratona de Programação da ACM dos anos de 2005 e 2004 respectivamente. O primeiro, dá um estoque de azulejos de tamanho pequeno médio e grande respectivamente e pede a disposição de custo mínimo para azulejar uma piscina de dimensões dadas. O segundo descreve um truque de adivinhação de cartas e pede que o aluno resolva 3 instâncias desse truque. Concluído em 28/10/06 com a sequência 173.

m10a - Regata e Luzes da festa

O primeiro exercício é da olimpíada ACM de 2005 e o segundo da IOI de 1998. A regata apresenta as coordenadas de um ponto de origem e de um ponto de destino, além de diversos obstáculos (dados pelas coordenadas de suas extremidades). O aluno deve calcular qual o menor caminho da origem ao destino. O exercício das luzes, sugere N lâmpadas inicialmente acesas, que podem ser controladas através de 4 botões (1=chaveia todas, 2=só as ímpares, 3=só as pares 4=1,4,7...) O exercício informa algumas lâmpadas que devem ficar acesas e outras apagadas, e pergunta como ficaram todas as N lâmpadas, após a aplicação dos botões de controle. Concluído em 03/08/07 com a sequência 186.

m12a - Jogo do Retângulo

Este exercício extraído da Olimpíada Internacional de Informática de 2005, ocorrido na Polônia, oferece um retângulo de dimensões inteiras. Cada jogador pode fazer 1 corte (vertical ou horizontal) sempre em unidades inteiras. Feito o corte, o menor pedaço é desprezado e segue o jogo. Quem ficar com o retângulo 1 x 1 perde. São dados os métodos de solução e os algoritmos. Dadas 3 instâncias (2 vencedoras e 1 perdedora) pede-se que o aluno identifique a perdedora e ofereça o lance ganhador nas outras 2. Concluído em 09/11/07 com a sequência 190.

m14a - Caminho e Vulcão

Dois exercícios baseados em similares pedidos na Olimpíada Espanhola de Informática. No primeiro, uma matriz de letras é estabelecida e pede-se o caminho que estabelece uma palavra dada. Note-se que a matriz tem as extremidades ligadas, o que torna mais complexa a busca. No segundo, um vulcão descrito pelas altitudes de seus platôs, derrama lava e o problema pede que o aluno descubra para onde vai a lava nos diversos instantes discretos de tempo. Concluído em 11/08/07 com a sequência 188.

m16a - Genoma e Palavras cruzadas

2 problemas baseados na prova da OBI de 1999. No primeiro problema são dadas duas cadeias de DNA e o aluno deve localizar as correspondências da cadeia original e de sua complementar. No segundo problema, um tabuleiro de palavras cruzadas de dimensão conhecida é dado e são dadas as localizações das casas pretas. O aluno deve numerar o tabuleiro e informar os endereços das palavras na horizontal e na vertical. Concluído em 31/12/07 com a sequência 201.

m18a - Trem/caminhão, robo e restaurante

3 problemas baseados na prova da OBI. No primeiro problema são dadas informações sobre frete e o aluno deve escolher o transporte por trem ou por caminhão. No segundo o caminho de um robô é dado por suas coordenadas e pergunta-se quantas voltas (giros) sobre si mesmo o robô dá. No terceiro, dados de entrada e saída de pessoas em um restaurante são dados e pergunta-se: qual o maior número de pessoas que estiveram em um dado momento dentro do recinto ? Concluído em 03/01/08 com a sequência 202.

m20a - Seq.H, carga pesada e rede ótica

3 problemas baseados na prova da OBI. No primeiro problema é definido um critério definidor de uma sequência binária. 4 sequências são dadas e pergunta-se se elas atendem ou não ao critério (recursivo). No segundo um mapa de estradas com alturas máximas são dadas e pede-se qual a carga mais alta que pode ser levada de uma cidade a outra. Finalmente, no terceiro problema, da-se uma rede ótica com seus custos por ramo e pede-se o menor custo de ligação (árvore de cobertura mínima). Concluído em 04/01/08 com a sequência 203.

m22a - Quermesse, troca de notas e saldo de gols

3 problemas baseados na prova da OBI. No primeiro problema pede-se quem ganhou um concurso (fácil). No segundo problema, usando notas de 50, 10, 5 e 1, deve-se informar quantas e quais notas compõem um valor (fácil). No terceiro, deve-se escolher qual o período em que um certo grande time teve o maior saldo de gols (não tão fácil). Concluído em 04/01/08 com a sequência 204.

m24a - Macaco prego e visita ao MASP

2 problemas baseados na prova da OBI. No primeiro problema pede-se qual a delimitação de uma reserva florestal que congregue o maior número possível de macacos prego a partir dos dados parciais de suas aparições. No segundo, o roteiro de uma visita às obras do MASP deve ser estabelecido minimizando o esforço e priorizando as obras mais importantes. Trata-se de um problema de minimização. Concluído em 08/01/08 com a sequência 205.

m26a - Anéis quadrados, Balaio e Meteoros

3 problemas baseados na prova da OBI. No primeiro problema pede-se qual a configuração quando vários anéis quadrados são vistos de cima. O segundo pede determinar qual a sequência de produção de balaio que minimiza a multa por atraso e o terceiro determina uma fazenda e diversos locais de queda de meteoritos, perguntando quantos caíram dentro da fazenda. Concluído em 08/01/08 com a sequência 206.

m28a - Dominó, sorvete e pirâmide de caixas

3 problemas baseados na prova da OBI. No primeiro problema pergunta-se se um dado conjunto de pedras de dominó podem ou não formar um jogo. No segundo, dão-se diversos trajetos de sorveteiros e pergunta-se qual a parte da praia que ficará atendida por pelo menos um sorveteiro, e no terceiro, dão-se diversas caixas de diversas dimensões e pergunta-se qual a altura da maior pilha possível, desde que uma caixa não ultrapasse nenhuma dimensão da caixa que está abaixo. Concluído em 09/01/08 com a sequência 207.

m30a - Tesouro, Aeroporto e Pedágio na Coreia

3 problemas baseados na prova da OBI. No primeiro problema dão-se 2 valores (de João e de José) e o conteúdo de uma arca do tesouro. A pergunta é se é possível ambos terem o mesmo valor. O segundo pergunta qual aeroporto ficará mais congestionado, e o terceiro pergunta quais cidades uma pessoa pode alcançar gastando no máximo P pedágios. Concluído em 17/01/08 com a sequência 208.

m32a - Finlândia, Supermercado e Número de Erdos

3 problemas baseados na prova da OBI. No primeiro problema pede-se quantas cidades um ciclista pode visitar na Finlândia, desde que não tenha que subir nada. No segundo, pede-se a melhor localização para um centro de distribuição de uma rede de supermercados. No terceiro, dão-se diversos artigos com seus autores, e pede-se o número de Erdos de 3 deles. Concluído em 18/01/08 com a sequência 209.

m34a - Furos, Senha bancária e Orkut

3 problemas baseados na prova da OBI. No primeiro problema pede-se qual o menor diâmetro de uma chapa redonda que cobre todos os furos cujos centros são dados. No segundo são dadas 2, 3 ou 4 digitações de senhas em um banco eletrônico, pedindo-se ao aluno que deduza a senha original correta e no terceiro pede-se que o aluno indique se é ou não é possível criar uma comunidade Orkut com as regras que lá são dadas. Concluído em 22/01/08 com a sequência 210.

m36a - Colheita de minhocas, furos e frota de taxis

3 problemas baseados na prova da OBI. No primeiro problema pede-se determinar o valor da colheita usando a trajetória otimizada de uma máquina de colheita de minhocas. No segundo, definem-se os buracos de minhoca como ligações entre partes do universo e a partir do mapa delas, pergunta-se se todos os planetas desse universo podem ser acessados. Finalmente, no terceiro o aluno deve decidir se abastece uma frota com álcool ou com gasolina. Concluído em 22/01/08 com a sequência 211.

m38a - Colheita linear, linhas de transmissão e desculpas

3 problemas baseados na prova da OBI. No primeiro problema um campo de colheita deve ser otimizado, sabendo que a colheitadeira não faz curvas. No segundo, o estado de uma rede de transmissão de eletricidade deve ser avaliado e no terceiro um bilhete de desculpas com a maior ocorrência da palavra desculpe, deve ser construído. Concluído em 23/01/08 com a sequência 212.

m40a - Mini-poker, Margaridas e Truco

3 problemas baseados na prova da OBI. No primeiro problema 3 rodadas de mini-poker (Poker sem considerar o naipe das cartas), cada uma com 5 cartas deve ser valorizada pelo programador. No segundo, pede-se a otimização de uma colheita de margaridas e no terceiro, o aluno deve avaliar 3 rodadas de um jogo de truco. Concluído em 23/01/08 com a sequência 213.

m42a - Busca de ciclos, troca de moedas e editor gráfico

Exercícios de Skiena03. Três exercícios do livro Programming Challenges do Skiena. Podem ser submetidos à correção nos sites www.programming-challenges.com e online-judge.uva.es. Os exercícios são: a) a busca do ciclo máximo em uma sequência dada; b) a troca de moeda em uma viagem em grupo e c) o resultado de comandos de um editor gráfico. Concluído em 18/06/08 com a sequência 217.

m44a - Interpretador de comandos, palíndromo e fibonacci

Exercícios de Skiena03. Três exercícios do livro Programming Challenges do Skiena. Podem ser submetidos à correção nos sites www.programming-challenges.com e online-judge.uva.es. Os exercícios são: a) O que acontece em um interpretador de comandos dado um determinado programa numérico; b) quantos ciclos são necessários para achar um palíndromo e c) Quantos números de fibonacci existem em um intervalo de números inteiros fornecidos. Concluído em 22/06/08 com a sequência 218.

m46a - Maratona: blocos, lixo e caixas

Os problemas 101,102 e 103 do volume I do site UVA (uva.onlinejudge.org). O primeiro problema pede a programação de um braço robótico que recebe diversos blocos e deve manipulá-lo. O segundo pede qual a separação de garrafas que garante trabalho mínimo e o terceiro pede quais caixas cabem uma dentro da outra. Concluído em 03/08/09 com a sequência 246.

m48a - Maratona: fermat, cat-in-hat, sub-retângulo

Os problemas 106,107 e 108 do volume I do site UVA (uva.onlinejudge.org). O primeiro problema pede a o calculo da quantidade de triplas pitagóricas em um determinado conjunto. O segundo pede o número de gatos e a altura total da pilha de gatos e o terceiro a soma máxima em sub-retângulos de uma matriz dada. Concluído em 05/08/09 com a sequência 247.

m50a - Maratona: árvores (),s, criptografia e TSP

Os problemas 112,113 e 116 do volume I do site UVA (uva.onlinejudge.org). O primeiro problema pergunta se um dado caminho existe em uma árvore parentizada. O segundo pede que seja feita uma extração de raiz de número grande e o terceiro solicita o caminho mínimo em uma matriz retangular. Concluído em 07/08/09 com a sequência 248.

m52a - Maratona: panquecas, tubos e árvores

Os problemas 120,121 e 122 do volume I do site UVA (uva.onlinejudge.org). O primeiro problema pede as operações de “corte” para ordenar uma pilha de panquecas. O segundo pergunta qual o número máximo de tubos que cabem em um dado container de dimensões fornecidas. Existem 2 tipos de arranjos (grid e skew) e o aluno deve decidir entre eles. Finalmente, árvores são dadas pela especificação de todos seus nodos na forma de (valor, caminho desde a raiz) e pede-se qual a sequência de visitação em largura dos nodos da árvore. Concluído em 08/08/09 com a sequência 249.

m54a - Maratona: N. feios, permutações e fat-fat

Os problemas 136,146, 153 e 160 do volume I do site UVA (uva.onlinejudge.org). O primeiro problema pede para que se calcule o conjunto de números feios. O segundo e o terceiro pedem a craição e enumeração de todas as permutações de uma dada coleção de objetos e o quarto problema pede o cálculo do fatorial a partir dos fatores primos do número. Concluído em 10/08/09 com a sequência 250.

m56a - Maratona: circunferência, intersecção e repetição

Os problemas 190, 191, e 202 dos volumes I e II do site UVA (uva.onlinejudge.org). O primeiro problema pede para que se calculem as esquações de uma circunferência a partir de 3 pontos a ela pertencentes. O segundo pede a intersecção entre um segmento e um retângulo e o terceiro pede para calcular o tamanho do ciclo em uma dizima periódica. Concluído em 12/08/09 com a sequência 251.

m58a - Milhário e Vaiqueémole

Exercícios feitos pelo autor para o treino UP da Maratona 2008. No primeiro exercicio uma data válida é informada e o aluno deve calcular o milhário dela (a data informada MAIS 1000 dias). No segundo exercicio somas em bases quaisquer ($2 \leq base \leq 60$) são informadas e o aluno deve achar o resultado correto na base informada. Concluído em 22/06/08 com a sequência 219.

m60b - Maratona UP 2012

Quatro implementações simples: banquinha do xerox, assentamento de lajotas, cata corruptos e carro flex. Concluído em 14/05/12 com a sequência 332.

m60c - Maratona UP 2012 - p2

Três implementações simples: casas atacadas pelo cavalo no jogo de xadrez, custo mínimo de um pedágio simples e média arredondada de um conjunto de notas. Concluído em 17/05/12 com a sequência 333.

n02a - Problema dos Generais Bizantinos

Apresenta o problema dos generais bizantinos como um problema de confiabilidade em redes distribuídas nos quais nodos podem falhar de maneira não imediatamente detectável. Concluído em 09/11/17 com a sequência 555.

n05a - Autenticação de mensagens

Descreve o processo de autenticar um objeto binário. Realça a importância deste processo no âmbito da arquitetura Bitcoin. Sugere um exercício em *toy domain* para que o aluno vivencie a execução do algoritmo (exercício extraído de vivo714) Concluído em 10/11/17 com a sequência 556.

n10a - Criptografia assimétrica de curva elíptica

Descreve o processo de gerar uma chave privada e depois gerar a chave pública correspondente usando funções primitivas de curvas elípticas. Concluído em 28/11/17 com a sequência 562.

n20a - Numeros randômicos ou aleatórios

Descreve alguns métodos randômicos puros e outros pseudo-randômicos pedindo ao aluno que gere pseudo-aleatórios. Concluído em 16/11/17 com a sequência 558.

n25a - Base 58

Mostra as chaves pública e privada em Bitcoin e como elas podem ser manipuladas. Mostra 2 chaves públicas e pede que o aluno ache os endereços bitcoin convertidos a base 58. Concluído em 15/11/17 com a sequência 557.

n35a - Carteiras em Bitcoin

Mostra o conceito de carteira em bitcoin e mostra também o inteligente truque de transformar uma senha hexadecimal em uma série de palavras em português. Ajuda na hora de transcrever, guardar, recuperar. O aluno tem que fazer estas 2 conversões: lista de palavras $\Leftrightarrow$ senha hexadecimal. Concluído em 07/12/17 com a sequência 565.

n40a - Scripts de transação

Descreve a linguagem de script que o Bitcoin usa para trancar e destrancar a moeda enquanto esta vai fluindo desde a sua geração até o último gastador. Os alunos tem que interpretar 3 scripts usando uma mini-linguagem. Concluído em 02/12/17 com a sequência 563.

n55a - Árvore de Merkle

Descreve a Árvore de Merkle e seu uso no esquema de certificação do Bitcoin. O aluno recebe 4 transações (aqui representadas por versos de Os Lusíadas) e deve verificar se eles estão ou não no bloco de controle usando uma árvore deste tipo. Concluído em 20/11/17 com a sequência 560.

n60a - Mineração e Prova de Trabalho

Descreve como é o processo de mineração de Bitcoin. Depois o aluno tem que simular a mineração e achar algumas “pepitas”. Concluído em 18/11/17 com a sequência 559.

n65a - Filtros de Bloom

Descreve os filtros de Bloom e como eles são usados no Bitcoin. O aluno deve processar um arquivo publicado na AVA e informar o resultado de sua verificação junto a um filtro deste tipo usando 6 funções de hash distintas. Concluído em 20/11/17 com a sequência 561.

n75a - Bifurcação e disputa de blockchain

Mostra o que acontece quando dois nodos distantes mineram um bloco quase ao mesmo tempo, bipartindo a rede em 2 blocos. Mostra como a situação se resolve na sequência. Ao aluno cabe simular soma sub-rede com 10 nodos e tempos de disseminação diferentes e ao final ver com quantos nodos cada sub-rede é formada. Concluído em 04/12/17 com a sequência 564.

o00a - Matemática OBMEP

Exercícios extraídos da Olimpíada Brasileira de Matemática. Concluído em 18/10/16 com a sequência 523.

o01b - Aquecimento em Matemática I

10 Exercícios baseados da Olimpíada Brasileira de Matemática. Cada aluno tem uma instância inédita. Concluído em 19/10/16 com a sequência 524.

o01c - Aquecimento em Matemática II

10 Exercícios baseados da Olimpíada Brasileira de Matemática. Cada aluno tem uma instância inédita. Concluído em 21/10/16 com a sequência 525.

o01d - Aquecimento em Matemática III

10 Exercícios baseados da Olimpíada Brasileira de Matemática. Cada aluno tem uma instância inédita. Concluído em 31/10/16 com a sequência 527.

o01e - Aquecimento em Matemática IV

10 Exercícios baseados da Olimpíada Brasileira de Matemática. Cada aluno tem uma instância inédita. Concluído em 09/11/16 com a sequência 529.

o01f - Aquecimento em Matemática V

10 Exercícios baseados da Olimpíada Brasileira de Matemática. Cada aluno tem uma instância inédita. Concluído em 13/11/16 com a sequência 530.

p05a - maple: introdução

Apresenta as regras básicas de uso do Maple e a seguir propõe 19 exercícios distintos (extraídos de uma matriz de 57) para o aluno resolver. Concluído em 18/02/12 com a sequência 317.

p15a - Matemática básica em Maple

Exercícios de aplicação de fórmulas usando Maple. Concluído em 20/07/12 com a sequência 340.

p25a - maple: álgebra básica

Apresenta os comandos algébricos básicos (solve, isolve, msolve, simplify, factor, subs, convert, ...) e apresenta 19 exercícios distintos, extraídos de uma matriz de 57 para o aluno resolver. Concluído em 21/02/12 com a sequência 318.

p45a - Programação em Maple

Os principais comandos e as estratégias adequadas para a programação em Maple. Ao final, ele pede 3 pequenos programas Maple. Concluído em 14/04/12 com a sequência 325.

p55a - Mais programação Maple

10 enunciados aleatórios para o aluno construir programas simples em Maple. Concluído em 09/05/12 com a sequência 330.

p65a - Lógica básica para Maple

Apresenta 10 exercícios triviais em que o aluno deve escolher uma de 3 alternativas que melhor respondem à questão. Não é específico para Maple, mas foi pensando nele e nas turmas que o aprendem que o exercício foi bolado. Concluído em 05/06/12 com a sequência 335.

p75a - Gráficos em Maple

Apresenta os principais comandos 2D e 3D para desenho de gráficos em Maple. Concluído em 21/07/12 com a sequência 341.

p76a - Leitura de dados em Maple

Apresenta 3 problemas nos quais o aluno deve ler um arquivo magnético e obter informações que estão lá, depois de processar os arquivos. Concluído em 25/09/12 com a sequência 354.

p91a - Maple em 1 dia

Resumo resumido do Maple para ser dado em uma única aula. Concluído em 14/05/15 com a sequência 471.

q05b - Arquitetura de computadores I

Descreve a arquitetura de computadores. Fala sobre instruções de máquina, tipos de memória, subsistemas do processador e suas conexões. Ao final pede valores de barramentos. Concluído em 05/05/24 com a sequência 692.

q05c - Arquitetura de computadores II

Continua a descrição de arquitetura de computadores. Descreve alguns processadores no estado da arte em 2024. Fala sobre CISC, RISC, pipeline e processamento paralelo. Ao final pede que o aluno resolva uma instância do exercício vindo do vivx772. Concluído em 05/05/24 com a sequência 693.

q06a - Um computador simples

Descreve tipo muito simples de computador. Dá um conjunto de 16 operações de máquina e resolve 2 exemplos de linguagem de máquina nesse computador. O aluno tem 2 exercícios: compilar uma expressão aritmética para tal linguagem de máquina e o caminho inverso: dado um programa em l.m. pergunta-se o que ele faz ? Concluído em 05/05/24 com a sequência 694.

q10a - Objetos binários grandes - BLOBs

Descreve os BLOBs (objetos binários grandes) como agregação de dados binários sujeitos a alguma estrutura pré-definida. Depois descreve o padrão BMP e seus concorrentes. Depois o padrão MP3 e seus concorrentes, os padrões de vídeo e o padrão PDF. Termina com um exercício de internals do arquivo PDF Concluído em 08/05/24 com a sequência 698.

q20b - Redes de computadores

Descreve redes e suas estruturas físicas. Aborda as categorias (LAN, MAN e WAN). Comenta sobre a suite de protocolos TCP/IP. Encerra com os principais serviços na Internet. O aluno deve projetar um conjunto de serviços para uma aplicação hipotética. Concluído em 06/05/24 com a sequência 695.

q20c - Modelo de camadas na Internet

Descreve as camadas de aplicação, transporte, rede, enlace de dados e física. Ao final pede que o aluno transporte estes conceitos para uma aplicação do mundo real Concluído em 06/05/24 com a sequência 696.

- Sistemas operacionais

Descreve as principais funcionalidades e o funcionamento de um sistema operacional. Manda investigar algumas questões de um sistema real em funcionamento. Concluído em com a sequência .

q32a - Representação de números em máquinas

Descreve a representação interna de inteiros e pontos flutuante. O aluno deve fazer interpretações de ambos. Concluído em 15/06/24 com a sequência 720.

q37a - Algoritmo A* - aplicações

Descreve o algoritmo A* e dá 3 aplicações: tarkin, problema das garrafas e busca em uma matriz numérica grande. Concluído em 14/06/24 com a sequência 716.

q43a - Alocação sequencial e encadeada

Descreve como a memória é alocada e depois mostra como o DOS-Windows e como o Unix faz alocações em disco, comparando ambas. Concluído em 22/06/24 com a sequência 723.

q66a - Aprendizado profundo

Mostra as ideias por trás do aprendizado profundo (redes neurais artificiais). Descreve o Keras e o problema de cães & gatos. Finalmente pede para o aluno calcular uma RNA binária de 2 camadas bem simples. Concluído em 10/06/24 com a sequência 715.

q80a - Tabelas hash

Descreve a importância e o funcionamento das tabelas hash e mais genericamente de funções hash. Pede que o aluno calcule 4 casos bem simples. Concluído em 06/08/24 com a sequência 743.

q81a - Grafos e caminho mínimo

Apresenta o conceito de grafo e depois o de caminho mínimo. Para este são 2 algoritmos: o de Dijkstra ($O(n^2)$) e o de Floyd-Warshall ($O(n^3)$). Ao final, o aluno deve interpretar um caso deste último. Concluído em 09/08/24 com a sequência 745.

q82a - Compressão de dados

Mostra o código Morse, depois o método do comprimento de fileira, depois a substituição de padrões, Huffman, codificação relativa, Shannon-Fano, código de vírgula, compressão auto-adaptável e finalmente o algoritmo LZW. No final, o aluno precisa descomprimir 3 mensagens em Morse e 3 em Huffman. Concluído em 10/08/24 com a sequência 746.

q83a - Caixeiro viajante

Apresenta o Problema do Caixeiro Viajante (ou Traveling Salesman Problem) e mostra as dificuldades na solução exata do problema. Apresenta a solução gulosa. Ao final, o aluno deve interpretar um caso deste último. Concluído em 11/08/24 com a sequência 747.

q84a - Criptografia

Um resumo do assunto, envolvendo vivxz35 (citale espartano), vivx709 (viginère), vivx708 (playfair), vivx715 (enigma), vivx710 (DES), vivx711 (TEA), vivx718 (El Gamal) e encerrando com o algoritmo RSA (vivx712 e vivx716) sobre o qual são pedidos vários exercícios. Concluído em 18/09/24 com a sequência 763.

q86a - Caminho mínimo: Metrô de Curitiba

Baseado no exercício 627, a ênfase aqui é a interpretação dos resultados do algoritmo (que são fornecidos em mídia magnética). As perguntas são parecidas às do exercício 627. Concluído em 02/09/24 com a sequência 756.

q91a - Árvores

Descreve a estrutura de dados conhecida como árvore. Usa uma árvore binária de pesquisa para exemplificar características e algoritmos de árvore. Concluído em 30/06/24 com a sequência 725.

q92a - Árvores-B

Descreve a estrutura de dados conhecida como árvore-B. Mostra seu funcionamento e porque ela é tão usada em praticamente todos os sistemas de gerenciamento de banco de dados. Exemplifica com uma descrição do SY-BASE. Concluído em 03/07/24 com a sequência 726.

r20a - Cartela de pseudo-código

Descreve os padrões de pseudo-código. Concluído em 08/03/13 com a sequência 375.

r24a - Cartela de Visual G

Descreve os padrões de Visual G. Concluído em 08/03/13 com a sequência 376.

r25a - Prática de VisualG

Este exercício sugere aos alunos que implementem seus algoritmos em VisualG. Uma lista de 12 comandos deve ser implementada, depois 3 comandos de SEs compostos e encadeados e finalmente o algoritmo dos dígitos verificados deve ser executado. Esta folha deve vir depois das folhas 011 (comandos aritméticos e lógicos) 017 (SEs compostos e encadeados) e 024 (dígitos verificados). Concluído em 11/04/08 com a sequência 214.

r26a - Linguagem C

Descreve a linguagem C e seus principais comandos e funções. Mostra como buscar e instalar um compilador C e um editor adequado a este compilador. Manda criar 3 programas C e pergunta o que eles responderiam para dados que são mostrados na folha. Concluído em 04/05/11 com a sequência 278.

r28a - Linguagem APL

Descreve a linguagem APL e seus principais comandos e funções. Mostra como buscar e instalar um interpretador APL. Concluído em 11/05/11 com a sequência 280.

r29a - xBase

Resumo das linguagens xBase (dbase, clipper, Fox,...) e dos comandos básicos de banco de dados destes ambientes. Concluído em 01/04/12 com a sequência 345.

r30a - Prolog

Resumo da linguagem Prolog Concluído em 01/04/12 com a sequência 346.

r32a - Cartela e introdução a LISP

Cartela de principais conceitos e funções de LISP. Concluído em 07/09/12 com a sequência 353.

r37a - Cartela de Python

Descreve algumas (poucas) características, comandos e dicas da linguagem Python. Concluído em 06/03/17 com a sequência 534.

r38a - Apresentação do Matplotlib

Apresenta o pacote Python para gerar gráficos e desenhos e pede que o aluno produza 3 gráficos lá especificados. Concluído em 21/05/18 com a sequência 579.

r39a - Apresentação do NumPy

Apresenta o pacote Python para manipulações matemáticas pesadas e pede que o aluno produza uma multiplicação matricial. Concluído em 22/05/18 com a sequência 580.

r41a - Apresentação do SymPy

Apresenta o pacote Python para manipulações simbólica de expressões matemáticas, (CAS=Computer Algebra System) e pede que o aluno calcule uma equação e ache uma integral definida. Concluído em 23/05/18 com a sequência 581.

r49a - Prática em LATEX

Mostra um pequeno conjunto de comandos LATEX, descreve o processo de sua instalação e uso e pede ao aluno que gere um relatório com alguns comandos latex que lá são dados. Concluído em 29/12/10 com a sequência 271.

r50a - Cartela de HTML

resumo da linguagem HTML Concluído em 20/12/15 com a sequência 478.

r51a - Linguagem J

Descreve uma pequena parte do mundo J, desde a sua origem como uma expansão de APL. Só arranha a complexidade e potência deste ambiente. Concluído em 13/01/20 com a sequência 640.

r55a - Cartela de CSS

resumo da linguagem CSS Concluído em 20/12/15 com a sequência 479.

r60a - Cartela de JS

resumo da linguagem Javascript Concluído em 20/12/15 com a sequência 480.

r70a - Cartela de Freemat/Octave

resumo da linguagem Freemat/Octave Concluído em 20/12/15 com a sequência 481.

r80a - Cartela de MySQL

Principais informações e comandos (resumidos) do SGDB MySQL. Concluído em 15/01/16 com a sequência 483.

t02a - Relações

Mostra a teoria associada e pede para o aluno resolver 4 exercícios sobre relações, propriedades reflexiva, simétrica e transitiva, além do conceito de partição. Concluído em 09/09/14 com a sequência 426.

t04a - Funções e somatórias

Mostra a teoria associada e pede para o aluno resolver 10 exercícios sobre funções e somatórias. Concluído em 10/09/14 com a sequência 426.

t07c - Ciclos de somatório em C++

Descreve 15 somatórios e pede que o aluno implemente programas que os calculem. Concluído em 21/02/19 com a sequência 607.

t07p - Ciclos de somatório em Python

Descreve 15 somatórios e pede que o aluno implemente programas que os calculem. Concluído em 05/05/17 com a sequência 536.

t10a - Progressões aritméticas e geométricas

Apresenta a teoria das progressões aritméticas e geométricas e depois pede que o aluno resolva 12 problemas aleatórios. Concluído em 27/08/14 com a sequência 419.

t13a - Demonstrações Algébricas

Apresenta diversas demonstrações, a saber: quadrado perfeito, a fórmula de Báskara, o Teorema de Pitágoras, a derivada de x^2 e a derivada da função $\sen(x)$. Pede que o aluno calcule a derivada de uma função *ad hoc*. Concluído em 02/08/21 com a sequência 663.

t16a - Matrizes

Apresenta a teoria das matrizes e depois pede que o aluno resolva 12 problemas aleatórios envolvendo somas matriciais multiplicações por escalar, multiplicações matriciais, solução de sistemas lineares e solução de polinômios matriciais. Concluído em 29/08/14 com a sequência 420.

t17a - Prática em matrizes (Python)

Oferece 2 matrizes grandes em mídia magnética para o aluno e pede que ele calcule algumas somatórias e o resultado da multiplicação matricial Concluído em 11/05/17 com a sequência 537.

t19a - Equações Lineares

Mostra a teoria associada e pede para o aluno resolver 4 sistemas lineares, além de desenhar a equação de uma reta. Concluído em 31/08/14 com a sequência 423.

t22a - Determinantes

Mostra a teoria associada e pede para o aluno resolver 6 exercícios sobre determinantes (2×2 e 3×3), além de resolver alguns sistemas de equações usando a teoria dos determinantes. Concluído em 01/09/14 com a sequência 424.

t25a - Coeficientes Binomiais

Mostra a teoria associada e pede para o aluno resolver 6 exercícios sobre coeficientes binomiais, além do Triângulo de Pascal. Concluído em 02/09/14 com a sequência 425.

t30a - Análise Combinatória

Descreve o princípio fundamental da contagem, permutações com e sem repetição, amostras ordenadas, combinações, partições e o diagrama de árvore. 10 problemas são apresentados. Concluído em 05/02/15 com a sequência 468.

t35a - Probabilidade

Descreve espaço amostral e eventos. Daí define eventos equiprováveis. Probabilidade condicional e processos estocásticos finitos. Independência. Ao final, 10 problemas são apresentados. Concluído em 06/02/15 com a sequência 469.

t36a - Prática em probabilidade: primos

Começa definindo números primos e sua importância na computação. Descreve o pequeno Teorema de Fermat e o algoritmo probabilístico de Rabin e como este trata probabilidades crescentes de primalidade até o limite desejado. O aluno deve detectar se 4 números grandes que são dados são primos ou não. Concluído em 01/02/15 com a sequência 466.

t45a - Variáveis aleatórias

Dá uma introdução a variáveis aleatórias e jogos de azar e pede 12 problemas envolvendo estes temas. Concluído em 12/09/14 com a sequência 430.

t50a - Estatística Descritiva

Mostra a teoria associada, dá as fórmulas da média, mediana, moda, desvio padrão e variância tanto para dados isolados quanto para dados agrupados e pede para o aluno resolver 10 exercícios. Concluído em 11/09/14 com a sequência 428.

t51a - Prática em estatística descritiva

Entrega um arquivo sequencial com 3 conjuntos de dados (alunos da UP, meteorológicos de Curitiba e de salário agregados de diversos estados brasileiros) e pede que o aluno elabore diversas medidas. Concluído em 10/06/17 com a sequência 542.

t60a - Distribuições binomial e normal

Define as duas distribuições e pede 7 exercícios para o aluno. Concluído em 30/09/14 com a sequência 433.

t70a - Aquecimento em geometria analítica

Dá uma introdução ao sistema de eixos cartesianos e pede 7 problemas envolvendo pontos e distâncias no plano. Concluído em 12/09/14 com a sequência 429.

t80a - Cônicas

Descreve elipses, hipérboles e parábolas e dá 8 exercícios para o aluno fazer. Concluído em 29/10/14 com a sequência 438.

t82a - Exercícios de Teoria dos Conjuntos

10 Exercícios de múltipla escolha sobre temas da Teoria dos Conjuntos. Concluído em 05/12/15 com a sequência 477.

t85a - Aquecimento em Cálculo

Apresenta fundamentos do cálculo (da álgebra na verdade) e pede 10 exercícios sobre problemas literais e funções. Baseado no livro “How to solve word problems in Calculus” de Eugene Don e Benay Don. Concluído em 15/09/20 com a sequência 641.

t87a - Taxas de Variação e Derivadas

Apresenta o conceito de derivada e pede 10 exercícios sobre problemas de derivadas. Baseado nos capítulos 1 e 2 do livro “How to solve word problems in Calculus” de Eugene Don e Benay Don. Concluído em 20/09/20 com a sequência 642.

t89a - Derivadas Relacionadas

Apresenta o conceito de derivadas relacionadas e pede 10 exercícios sobre problemas de derivadas relacionadas. Baseado no capítulo 3 do livro “How to solve word problems in Calculus” de Eugene Don e Benay Don. Concluído em 25/09/20 com a sequência 643.

t90a - Máximos e mínimos

Apresenta o conceito de otimização de funções e pede 10 exercícios sobre problemas deste tipo. Baseado no capítulo 4 do livro “How to solve word problems in Calculus” de Eugene Don e Benay Don. Concluído em 29/09/20 com a sequência 644.

t92a - Funções trigonométricas

Apresenta o conceito de derivação e otimização de funções trigonométricas e pede 6 exercícios sobre problemas deste tipo. Baseado no capítulo 5 do livro “How to solve word problems in Calculus” de Eugene Don e Benay Don. Concluído em 29/09/20 com a sequência 645.

t94a - Funções exponenciais

Apresenta o conceito de derivação e otimização de funções exponenciais (crescimento de populações, decaimento radioativo, juros compostos e similares). Pede 8 exercícios sobre problemas deste tipo. Baseado no capítulo 6 do livro “How to solve word problems in Calculus” de Eugene Don e Benay Don. Concluído em 30/09/20 com a sequência 646.

t96a - Problemas de área

Apresenta o conceito de integração, inclusive de sólidos de rotação. Pede 5 exercícios sobre problemas deste tipo. Baseado no capítulo 7 do livro “How to solve word problems in Calculus” de Eugene Don e Benay Don. Concluído em 05/10/20 com a sequência 647.

u02a - RPN para compiladores

Clone da folha 313 na sua versão para a disciplina de construção de compiladores. Os algoritmos estão em C. Concluído em 15/02/14 com a sequência 407.

u04a - Construção de compilador aritmético

Esta folha opera em sociedade com a 002f e implementa aquela linguagem. Concluído em 16/11/13 com a sequência 393.

u05a - Compilador com reais e ids longos

Uma extensão do exercício U04, no qual agora aparecem identificadores longos e números reais (grandes e podendo ter ponto flutuante). Concluído em 21/12/13 com a sequência 403.

u06a - Compilador Latex

Pede que o aluno construa um programa de nome PRELATEX que pré-compile um texto e inclua nele comandos L^AT_EX corretos. Concluído em 03/12/13 com a sequência 401.

u08a - Compilação do comando se..fimse

Esta folha opera em sociedade com a 020a e implementa aquela linguagem. Concluído em 19/11/13 com a sequência 394.

u10a - Compilação de um arquivo PDF

Esta folha está baseada na 680 e descreve um arquivo PDF. Concluído em 20/11/13 com a sequência 395.

u11a - Utilitário extrator de textos em PDF

Descreve o mecanismo para um utilitário (a ser feito pelo aluno) conseguir extrair o texto existente dentro de um arquivo PDF. Concluído em 14/05/14 com a sequência 411.

u12a - Compilador de expressões lógicas

Solicita a produção de uma calculadora lógica. Ela deve receber expressões lógicas e produzir a tabela verdade final. Opera em conjunto com a folha 095 e com o utilitário tv.exe. Concluído em 28/11/13 com a sequência 397.

u16a - Compilador do Jogo da Vida (Conway, 1970)

Apresenta as regras do jogo da vida e pede que o aluno construa um interpretador de tabuleiros, pedindo o que acontece na geração 50, 100 e 200. Concluído em 29/11/13 com a sequência 398.

u20a - Compilação de comandos de repetição

Esta folha opera em sociedade com a 023a e implementa aquela linguagem. Concluído em 29/11/13 com a sequência 399.

u24a - Compilador Maple → C

Pede que o aluno construa um programa de nome MAP2C que transforme um programa escrito em Maple no seu equivalente em C. Concluído em 03/12/13 com a sequência 402.

u28a - Compilação de tudo (u04+u08+u20)

Esta folha implementa comandos aritméticos, condicionais, de repetição e funções recursivas. É quase uma linguagem completa. Concluído em 02/12/13 com a sequência 400.

u42a - Clone do 719

Estuda o formato do arquivo BMP true color. Concluído em 30/08/14 com a sequência 421.

u44a - Compilador de imagens

Pede a construção de um compilador que receba alguns comandos textuais e construa uma imagem a partir deles, gravando-a em um arquivo BMP true color. Concluído em 30/08/14 com a sequência 422.

z02a - Garotos e seus brinquedos

Um exercício de lógica simples, extraído da Internet, que mostra 5 garotos, 5 cores, 5 idades, 5 presentes, 5 sucos e 5 profissões. São dadas 23 regras e uma pergunta é feita ao final. Concluído em 17/12/11 com a sequência 310.

z03a - Mulheres no supermercado

Um exercício de lógica simples, extraído da Internet, que mostra 5 mulheres, 5 blusas, 5 itens esquecidos, 5 tipos de pagamento e 5 acompanhantes. São dadas 19 regras e uma pergunta é feita ao final. Concluído em 17/12/11 com a sequência 309.

z04a - Meninas Estudiosas

Um exercício de lógica simples, extraído da Internet, que mostra 5 meninas, 5 cores, 5 sucos, 5 cidades, 5 animais e 5 matérias de estudo. São dadas 22 regras e uma pergunta é feita ao final. Concluído em 14/12/11 com a sequência 308.

z05a - Quebra cabeça: zebra e água

Este exercício tem mais de 50 anos, tendo sido retirado do livro do Russel. São dadas 15 pistas e ao final são feitas 2 perguntas. Para responder cada aluno deve construir uma árvore de possibilidades e navegar por ela até chegar a uma conclusão. As perguntas originais eram *Quem é o dono da zebra* e *Quem bebe água*. Agora, as perguntas variam, já que cada aluno recebe uma instância diferente do problema. Concluído em 26/07/04 com a sequência 122.

z07a - Reveillon das moças

Mais um exercício de lógica, onde há 5 moças, usando 5 cores, que bebem 5 bebidas, com 5 idades, que têm 5 desejos e com 5 profissões diferentes. São dadas 20 regras e feitas 4 perguntas. Concluído em 11/01/12 com a sequência 313.

z08a - Lutadores de jui-jitsu

5 lutadores com faixas, nomes, idades, peso, altura e estados diferentes. Das 21 regras ao final perguntam-se 2 nomes. Concluído em 21/04/12 com a sequência 328.

z09a - Sofrimento no dentista

Um exercício de lógica de alguma dificuldade, no qual 17 perguntas são feitas sobre um arranjo de pessoas. Concluído em 13/10/12 com a sequência 357.

z10a - Insanidade Instantânea

Baseado em um quebra cabeça que teve este nome comercial, este exercício tanto pode ser usado em IA (busca em espaço de estados) como em Estrutura de Dados (grafos). É um exercício bem divertido, já que o algoritmo de sua solução é elegante, simples e surpreendente. Concluído em 11/11/00 com a sequência 062.

z11a - Bloco dos cientistas famosos

Um exercício de lógica, no qual 23 perguntas são feitas sobre um arranjo de pessoas. Concluído em 13/10/12 com a sequência 358.

z15a - Dissecações geométricas

Um quebra-cabeça geométrico. Cada aluno receberá as peças de armar com as quais deverá montar 2 polígonos regulares distintos, embora de mesma área (já que compartilham as mesmas peças para serem montados). Após montar ambos, o aluno deve dizer qual aresta ficou adjacente a uma aresta dada, para cada uma das duas montagens. Concluído em 14/09/06 com a sequência 168.

z19a - O Planeta Terra recebe uma visita

Uma civilização extra-terrestre vem visitar nosso planeta. O aluno faz parte do comitê de recepção e descobre um texto matemático dos visitantes, contendo adições e subtrações de símbolos desconhecidos. Precisa descobrir características do sistema de numeração dos visitantes. Concluído em 31/10/17 com a sequência 554.

z20a - Sudoku para humanos

É um puzzle moderno (inventado em 1979 e popularizado fora do Japão apenas em 2004) que parece uma “palavra cruzada”, mas que usa os dígitos de 1 a 9 ao invés das letras. Popularizou-se no Japão pela imensa dificuldade de construir palavras cruzadas lá. Diariamente, a Folha de São Paulo publica um sudoku. Este exercício descreve o quebra-cabeças, indica as estratégias para sua solução e finalmente propõe uma instância ao aluno. Concluído em 05/09/06 com a sequência 166.

z25a - Quebra cabeça: Visão Lateral

Este quebra-cabeça é de invenção bastante recente. Ele pede que o leitor construa um desenho formado por pixels brancos e pretos, sabendo-se a quantidade e a aglomeração dos pixels pretos nas linhas e nas colunas do desenho. Parece trivial, mas pode ser extraordinário complexo. Os exercícios pedidos tem a dimensão 32 × 32, o que não é pouco. Concluído em 06/06/07 com a sequência 178.

z26a - Problemas aritméticos

Transforma uma vida em 1 dia e pede que o aluno calcule a que horas alguma coisa aconteceu. Define sequencias de Fibonacci genéricas e manda calcular três vezes o quinquagésimo termo e define aritmética modular (a dos ponteiros do relógio) e pede que o aluno faça alguns cálculos. Concluído em 19/09/16 com a sequência 515.

z35a - Citale Espartano

Descreve o citale e como foi usado com sucesso pela civilização espartana. Daí o aluno tem instruções de construir um citale usando 3 latas de refrigerante. Uma mensagem cifrada é entregue e o aluno deve decryptografá-la. Concluído em 03/10/14 com a sequência 434.

z41a - Construção de uma calculadora física

Descreve o uma régua de cálculo, mostra como os logaritmos foram usados por séculos para simplificar a álgebra e pede que o aluno faça uma multiplicação e uma divisão usando uma régua ad-hoc construída. Concluído em 23/09/19 com a sequência 634.

z55a - Um problema de Logística

Imagine uma empresa de distribuição de produtos adquiridos na web. Há 27 centros de consumo (capitais) e 20 fabricantes, além de 10 candidatas a sediar o centro de distribuição. Todo mundo têm as suas coordenadas. (latitude e longitude). O aluno tem que achar o local de menores custos associados. Concluído em 16/07/18 com a sequência 588.

z66a - Caminho mínimo de estradas e pontes

10 cidades, 25 estradas, contendo postos de pedágio e pontes com limite de peso. Um caminhão quer ir de A a B, pesando certo tanto. Há estradas de asfalto, pedras e cascalho. Deve-se descobrir o custo mínimo, indo só por asfalto, evitando as estradas de cascalho e sem nenhuma restrição. Concluído em 12/04/12 com a sequência 324.

z67a - Caminhos mínimos em 3 tipos de trens

Uma certa região geográfica é servida por 3 tipos de trens (super luxo, luxo e populares) cada um com seus preços. Há tabelas de locais e preços e pede-se um estudo de caminhos mínimos (mais baratos) entre todas as cidades da malha. Concluído em 21/11/12 com a sequência 366.

1 Índice por código dos exercícios

1 .

1.	001a	Programar e...	90.	061a	Grupos algébricos
2.	002b	Funcoes inventadas + mat basica	91.	062a	Engenharia reversa de algoritmos
3.	002c	Funcoes inventadas II + mat basica	92.	063a	Equacoes e inequacoes
4.	002d	Funcoes inventadas III + mat basica	93.	064a	3 Estruturas de repeticao
5.	002e	Funcoes inventadas IV + mat basica	94.	065a	Funcoes SE, ENQUANTO e REPITA
6.	002f	Funcoes inventadas V + mat basica	95.	066a	Geometrias não euclidianas
7.	002g	Funcoes inventadas VI + mat basica	96.	067a	Seguir o codigo: repeticoes
8.	003a	Algoritmo da raiz quadrada	97.	069a	Chines de 4 algoritmos aleatorios
9.	004a	Introducao a algoritmos	98.	071c	Programas C/C++ com parametros
10.	005a	Matematica Basica	99.	072b	Iniciando na programacao
11.	006a	Jogo da vida	100.	072d	Iniciando na programacao II
12.	007a	Descricao do GPS e simulacao	101.	072e	Iniciando na programacao III
13.	008a	Problemas matematicos	102.	072g	Iniciando na programacao IV
14.	009b	Achar o n-esimo em 30 sequencias	103.	072j	Iniciando na programacao V
15.	009d	Talento de D. Mariquinhas	104.	072k	Iniciando na programacao VI
16.	010a	Geografia terrestre: dist. e cap. Grant	105.	073a	Achar o decimo termo na sequencia
17.	011b	Matematica e logica basicas	106.	075a	Desenhando funcoes
18.	011c	Matematica e logica basicas (form C)	107.	076b	Leitura de dados preexistentes em C
19.	012a	Problemas envolvendo inteiros	108.	076c	Leitura de dados preexistentes Python
20.	013a	Mais problemas envolvendo inteiros	109.	077a	Chines de 4 algoritmos
21.	014a	Mais problemas de inteiros	110.	078a	Autovalores e autovetores
22.	015b	Comandos SE simples	111.	079a	Aplicacoes da Algebra Linear
23.	015d	Comandos SE (apenas codigo)	112.	080a	Introducao a Python
24.	015j	27 Comandos condicionais simples	113.	082a	Algoritmo de intercalacao
25.	016b	15 comandos condicionais	114.	084a	Balance Line
26.	016h	Comandos SEs aninhados	115.	086a	Algoritmo de quebra em relatorios
27.	017b	SEs compostos e encadeados	116.	088a	Sequencias e series
28.	017t	Working path em algoritmos	117.	089a	Otimizacao de funcoes
29.	018f	Pseudocodigo e fluxograma equivalentes	118.	090a	Se...entao: proposicao X comando
30.	018n	Pseudocodigo e nassi equivalentes	119.	093a	Logica proposicional
31.	019a	7 algoritmos para seguir (chines)	120.	094a	Tabelas verdade
32.	020a	Comandos SE com complexidade crescente	121.	095b	Construcao de tabela verdade
33.	021a	Seguir chines	122.	095e	Tabelas verdade aleatorias
34.	022a	Diversos SEs	123.	095f	Construcao de tabela verdade II
35.	023a	Enquanto e para basicos	124.	095h	Pratica em tabelas verdade
36.	024b	Digitos verificadores	125.	096a	Semantica em logica proposicional
37.	024e	Boleto bancario da FEBRABAN	126.	097a	Equivalencia e reducao ao absurdo
38.	025a	Aritmetica complexa	127.	098b	Algebra das proposicoes - Parte 1
39.	026a	Aritmetica inteira básica	128.	098f	Algebra das proposicoes - Parte 2
40.	027a	Bases maiores do que 10	129.	099b	Argumentos e validade
41.	028b	Condicionais aninhadas	130.	099h	Argumentos e validade ad-hoc
42.	028e	Condicionais aninhadas	131.	099j	Argumentos literais e validade
43.	028k	Condicionais aninhadas e compostas II	132.	100a	Demonstracoes de argumentos
44.	028n	Condicionais aninhadas e compostas III	133.	101b	Logica e diagramas de Venn
45.	029a	Condicionais literais	134.	101c	Teoria de conjuntos e diagramas de Ven
46.	030a	Integridade de funcoes C	135.	102a	Logica e quantificadores
47.	031a	Manipulação de datas	136.	103a	Demonstracao automatica de teoremas
48.	034a	Manipulação de datas: vetor hoj-ama	137.	104a	Dem. autom. teo. e PROLOG (clone 858)
49.	035b	Localizacao de oleoduto	138.	105a	Logica e Teoria dos conjuntos
50.	035c	Exponenciacao, diagonais e trelicas	139.	106a	Algebra de Boole
51.	036b	Aritmetica decimal, binaria e hexad	140.	108a	Logicas probabilistica e nebulosa
52.	036f	Fechamento de bases nao decimais	141.	110c	Operacoes com vetores e indirecoes
53.	037a	Base 64, funcionamento e conversoes	142.	110v	Operacoes com vetores e indirecoes
54.	038a	Sequencias de numeracao	143.	111a	Solicita o codigo de 9 funcoes
55.	039a	Um ambiente estimulante: Bitcoin	144.	112a	Pratica em vetores (Freemat/Octave)
56.	040a	Comandos de repeticao simples	145.	114a	Define vetor, registro e matriz
57.	041a	21 comandos de repeticao	146.	116b	Pesquisa em vetores - parte 1
58.	042a	14 comandos de repeticao	147.	116c	Pesquisa em vetores - parte 2
59.	043c	0 que faz o codigo em C++	148.	119a	Manuseio de tabelas
60.	043e	0 que faz o codigo em pseudocodigo	149.	120a	Pratica em conjuntos
61.	044c	Qual funcao faz o que - C++	150.	121p	Matrizes em Python
62.	044p	Qual funcao faz o que - Python	151.	125b	Certificar 5 algoritmos basicos
63.	044z	Escreva 10 programas C++	152.	125c	Certificar 5 algoritmos de vetor
64.	045a	Calculo de PI	153.	125d	Certificar 5 algoritmos vetor + difs
65.	046b	Condicionais em C++	154.	125e	Certificar 5 algoritmos parte 4
66.	046d	Condicionais em C++ II	155.	128a	Matrizes
67.	046p	Condicionais em Python	156.	129a	Pratica em matrizes
68.	046q	Condicionais em Python II	157.	131a	Conceito introdutorio de arvore
69.	047a	Mais repeticao	158.	132a	Ordenacao simples O(n**2)
70.	048c	Repeticao em C++	159.	135a	Manipulacoes no cubo Rubik
71.	048p	Repeticao em Python	160.	136b	Programacao I
72.	049b	Primeiros passos em programacao C++	161.	136c	Programacao II
73.	049d	Primeiros passos em programacao II-C++	162.	136d	Programacao III
74.	049p	Primeiros passos em programacao Python	163.	136e	Programacao IV
75.	049q	Primeiros passos programacao Python-II	164.	136f	Programacao V
76.	050c	Gerador de movimentos em xadrez-C++	165.	136g	Programacao VI
77.	050p	Gerador de movimentos em xadrez-Python	166.	140a	Pratica em eixos cartesianos
78.	051c	Pratica de programacao inicial C++	167.	144a	Numeracao romana
79.	051d	Pratica de programacao	168.	145a	Completando algoritmos
80.	051e	Pratica de programacao II	169.	146a	Compl. alg. II
81.	051p	Pratica de programacao inicial Python	170.	147a	Compl. alg. III
82.	052a	Pratica em programacao II	171.	148a	Compl. alg. IV - arvore Merkle
83.	053a	Pratica de programacao III	172.	149a	Compl. alg. V - sist lin e caixeiro
84.	054a	Kumon de algoritmos	173.	150a	Logica de programacao: ABP,m.q.,lombad
85.	055a	Programacao via arquivos	174.	157b	Pratica matrizes em C++
86.	056a	Introducao a programacao	175.	157c	Pratica matrizes em C++ - II
87.	057a	Continuar sequencias	176.	157p	Pratica matrizes em Python
88.	058a	Distribuicao de cadeiras no legislativ	177.	157q	Pratica matrizes em Python - II
89.	059a	Introducao a programacao - proj Euler	178.	158c	Matriz inversa em C++
			179.	158p	Matriz inversa em Python
			180.	159a	Matriz inversa - conceito
			181.	160a	Numeros primos
			182.	162c	Otimizacao de algoritmos em C++

183.	162p	Otimizacao de algoritmos em Python	278.	444a	Indices no padrao XBASE
184.	170c	Sao equivalentes as condicionais-C++	279.	445a	B-arvore com t=6
185.	170p	Sao equivalentes as condicionais-Pytho	280.	450a	Codigos: ASCII, Ortelino e Trocalettra
186.	175a	Perolas: complexos,pikachu e sierpinsk	281.	451a	Huffman - arvores iguais
187.	183c	Truques: convol., container e poker C+	282.	452a	Huffman - arvores diferentes
188.	183p	Truques: conv.container,poker: Python	283.	453a	Huffman arvore dinamica
189.	184c	Truques2: decomp. brinq e pitagoras-C+	284.	455a	Transformadas BWT e MTF
190.	184p	Tr2: decomp. brinq. e pitagoras Python	285.	456a	Caso pratico de huffman ad-hoc
191.	185c	Tr3: decomp. perfeito e superm. C++	286.	457a	Huffman sobre uma imagem BMP
192.	185p	Tr3: decomp. perfeito e superm. Python	287.	458a	Pratica em huffman: os lusiadas
193.	186c	Tr4: simetria, vulcao e fibonacci-C++	288.	461a	Heap
194.	186p	Tr4: simetria, vulcao fibonacci-Python	289.	480a	Unicode e UTF-8
195.	187c	Tr5: conjuntos, det e m.inv em C++	290.	511a	Sorts: primeira parte
196.	187p	Tr5: conjuntos, det e m.inv em Python	291.	511b	Juncao do 511 e 512: nao ha exercicio
197.	188c	Tr6: centro, histogr e xadrez - C++	292.	512a	Sorts: segunda parte
198.	188p	Tr6: centro, histogr e xadrez -Python	293.	513a	Redes de comparacao
199.	189c	Tr7: problemas da OBI	294.	521a	Hashing
200.	190a	10 problemas (papel-lapis) OBI2003	295.	531a	Busca: Boyer+Moore simplificado
201.	191b	5 problemas (lapis-papel) OBI2004	296.	532a	Compara buscas em tabelas
202.	191k	5 problemas (lapis-papel) OBI2004	297.	535a	Casamento aproximado de cadeias
203.	192a	10 problemas (lapis-papel) OBI2005	298.	536b	Bioinf. 1-maior subsequencia comum
204.	193b	7 problemas (lapis-papel) OBI2006	299.	536c	Biounf. 2-custos e busca padrao
205.	193c	6 problemas (lapis-papel) OBI2006	300.	538a	Expressoes regulares
206.	193d	6 problemas (lapis-papel) OBI2006	301.	555a	Representacao estruturas bidimensionai
207.	193e	7 problemas (lapis-papel) OBI2006	302.	560a	Analise estilistica de textos
208.	215a	Logaritmos	303.	570a	Codigos Reed Solomon
209.	220a	Complexidade, recursao e rel. recorr.	304.	571a	QR Code
210.	222a	Alto/baixo 1:conjuntos	305.	611a	Diametro de grafo com listas adjacente
211.	223a	Alto/baixo 2:pilha e fila	306.	612a	Caminhamento em lar e prof em grafos
212.	224a	Alto/baixo 3:ABP	307.	618a	Desenho de grafos (graphviz)
213.	225a	Alto/baixo 4:assembly e p. alto nivel	308.	621a	Caminho minimo em grafo (dijkstra)
214.	226a	Alto/baixo 5:matriz inversa	309.	622a	Caminhamento 1:1 (tennenbaum)
215.	231a	Complexidade: o problema da particao	310.	623a	Caminhamento n:n (floyd-warshall)
216.	232a	Complexidade: requisitos de automovel	311.	624a	Grafo: matriz de roteamentos
217.	233a	Complexidade: habilidades de booleiro	312.	627a	Caminho minimo no metro de curitiba
218.	240a	Problema da mochila 0-1 (knapsack)	313.	628a	Rotas e tempos minimos
219.	245a	Minimos quadrados	314.	632a	Fluxos ford-fulkerson/edmonds-karp
220.	246a	Regressao multipla	315.	633a	Sort topologico em grafos
221.	251a	Recursividade: funcoes A, B e C	316.	634a	Kruskal: arv de espalhamento minima
222.	252a	Recursividade: anagramas	317.	635a	Coloracao de grafos
223.	253a	Recursividade: passeio do cavalo	318.	636a	Caminho hamiltoniano: domino
224.	254a	Recursividade: 3 funcoes	319.	639a	Coloracao de grafos: aplicacoes
225.	255a	Arvore QUAD	320.	640a	PERT-CPM
226.	256a	Casamento estavel	321.	645a	Onde por o hipermercado ?
227.	257a	Recursividade: sudoku para computadore	322.	646a	Central 0800 com menores custos
228.	258a	Programacao literata e 3 f. recurs.	323.	650a	Algoritmo húngaro de assinalamento
229.	259b	TSP de pequeno porte: 6 cidades	324.	656a	Algoritmo Gale-Shapley de alocao
230.	259e	TSP em Python	325.	660a	Algoritmo do torneio de tenis
231.	259h	TSP em pseudo-codigo	326.	666b	Complexidade: busca binaria e recursao
232.	260b	Arvores com recursividade e apontadore	327.	666c	Tabela hash, largura, dijsktra guloso
233.	260c	Arvores com recursividade e cursores	328.	666d	Programacao dinamica, somas e k-vizinh
234.	261a	Arvores k-d	329.	666e	Trans. fourier, lg paralelos, f. bloom
235.	262a	Recursividade: Strassen e quickselect	330.	668a	Algoritmo dos k-vizinhos
236.	263a	Programacao linear: metodo simplex	331.	669a	Filtro de Bloom
237.	265a	Programacao dinamica	332.	671a	Multiplicacao de matrizes em serie
238.	266a	Selecao de atividades	333.	672a	Prog. dinamica: plano de producao
239.	271a	Alg. guloso/best/first/next/worst	334.	680a	Padrao PDF - construcao do PDF a mao
240.	272a	Encadeamento em um teatro	335.	690c	Compressao JPG em C
241.	273a	Arvore genealogica	336.	690p	Compressao JPG em Python
242.	291a	Simulador de FAT no dos/windows	337.	701a	Correcao de erros - alg. Hadamard
243.	292a	Simulador de alocao em UNIX	338.	705a	Certificacao digital pratica
244.	293a	Simulacao ambiente de alocao LISP	339.	708a	Cifra de Playfair
245.	311a	Exemplos de pilhas	340.	709a	Cifra de Viginere
246.	312a	Pilhas em assembler	341.	710a	Criptografia baseada no alg. DES
247.	313a	RPN-notacao polonesa reversa	342.	711a	Criptografia - algoritmo TEA
248.	314a	Pilhas: labirinto do ratinho	343.	712a	Criptografia de chave publica - RSA
249.	315a	Calculo da envoltoria convexa	344.	713a	Esteganografia: msg dentro de BMP
250.	318a	Compilador de pseudo codigo	345.	714a	Assinatura de arquivos
251.	319a	Engenharia reversa de compiladores	346.	715a	Maquina ENIGMA
252.	321a	Filas: implementacao e exemplos	347.	716a	Quebra de RSA (alg. rho pollard)
253.	322a	Filas circulares: impl. e exemplos	348.	717a	Codigo ADFGVX
254.	323a	Filas em assembler	349.	718a	Criptografia El Gamal
255.	324a	Ordenacao topologica usando filas	350.	719a	Esteganografia linear em BMP
256.	331a	Truco em assembler usando listas	351.	720a	Aplicativos comerciais: contabilidade
257.	332a	4 erros em listas encadeadas	352.	721a	Aplicativos comerciais: folha de pag.
258.	333a	Livros em biblioteca: exemplo de lista	353.	723a	Aplicativos comerciais: estoques
259.	341a	Listas dupl.encadeadas em assembler	354.	731a	LZW: compressao e descompressao
260.	342a	Listas duplamente encadeadas	355.	732a	Fractal de Mandelbroot
261.	343a	SGBDs antigos: inc/excl em LDEs	356.	735f	Calculus - sist. lineares - Octave
262.	351a	Equações diferenciais	357.	735p	Calculus - sist. lineares - Python
263.	352a	Transformada de Laplace	358.	750a	Reconstituicao de imagem com tomografi
264.	353a	Taxas de variacao relacionadas	359.	752a	Imagem BMP: descobrir linhas cinza
265.	411a	Arvores: caminhamentos em 3 arvores	360.	753a	Imagem BMP: descobrir linhas coloridas
266.	412a	Conversao de arvore n-aria em binaria	361.	754a	Conclucoes com mascaras em imagens
267.	421a	Arvores: redesenhar balanceando	362.	755a	Equalizacao de histograma em imagens
268.	431a	Arvore: criacao de ABP	363.	756a	Padrao GIF: buscar coisas em arquivos
269.	432a	ABPs: busca, criacao e altura	364.	760a	Padrao FITS
270.	433a	Arvores: inclusoes/exclusoes em ABPs	365.	763b	Petri nets: descobrir caracateristicas
271.	434a	Inc/Exc em 3 arvores binarias de pesq.	366.	763c	Petri nets: transacoes em um somador
272.	435a	Arvore binaria de partilha-ABPp	367.	763d	Petri nets: transacoes em uma ATM
273.	436a	Tries	368.	766a	Tecnicas de compressao - parte 1
274.	437a	Treinamento em ABPs	369.	767a	Tecnicas de compressao - parte 2
275.	441a	Aquecimento em arvores B	370.	768a	Compressao (obsoleta 766 e 767)
276.	442a	B arvore com t=3	371.	770a	Teoria de filas: SM e faminto de ouro
277.	443a	Indices em SYBASE (arvores)	372.	772a	Simulacao: banco quereomeu + sist. op.

373.	775a	Teoria de filas: simulacao ARENA	468.	b27a	Normalizacao em MER
374.	780a	Numeros pseudo-randomicos	469.	b30a	Pratica em MER: restaurante
375.	781a	Listas skip	470.	b40a	Pratica em MER: escolas do parana
376.	782a	Simulacao: porta-esper e questoes prov	471.	b45a	Engenharia reversa: construir o MER
377.	785a	0 problema da parada (3 descricoes)	472.	b50a	Pratica em MER: sist passagens aereas
378.	790a	Processamento fonetico (3 algs)	473.	b55a	Pratica em MER: locadora de veiculos
379.	798a	Roteiro para apresentacao da pos grad.	474.	b59a	Pratica em MER: Ultima tarefa
380.	799a	Perdidos na lua (decisoes coletivas)	475.	b60a	Criacao de um BD no ACCESS
381.	800a	Artigo e consideracoes de Turing	476.	b63a	Pratica em ACCESS
382.	802a	Maquina de Turing	477.	d05a	Questionario: introducao a BD
383.	805a	Questoes sobre o artigo de Turing	478.	d10a	Conceitos do modelo E-R
384.	807a	Computadores auto-replicantes	479.	d12a	MER: companhia aerea e revenda
385.	809a	Definicoes da Inteligencia Artificial	480.	d14a	Ferramentas CASE
386.	810a	Diagrama PAGE para sistemas de IA	481.	d30b	10 documentos para normalizar
387.	814a	Mundo dos aspiradores de po	482.	d30c	Mais 10 documentos para normalizar
388.	815a	Sistemas de producao	483.	d32a	Engenharia reversa
389.	817a	Pratica de EXSINTA	484.	d35b	Funcoes de string em MySQL
390.	819a	Sistemas especialistas	485.	d35c	Funcoes de data e hora em MySQL
391.	820a	Solucao do cubo RUBIK	486.	d60a	Introducao ao modelo relacional
392.	821a	Busca cega: 5 exercicios	487.	d62a	Normalizacao de relacoes
393.	824a	Minimax	488.	d63a	Integridade referencial em BD
394.	825a	Corte alfa-beta do minimax	489.	d64a	Restricoes de integridade em BD
395.	826b	A*: tarkin e jarras	490.	d66a	MySQL: resumo e exercicio
396.	826j	A*: jarras com implementacao	491.	d69b	Seguranca e direitos em MySQL
397.	826t	A*: tarkin com implementacao	492.	d70a	Um sistema
398.	830a	A*: trajeto de um robot	493.	d71b	Consultas MySQL e respostas: casar
399.	831a	A*: caminho mais barato	494.	d71c	Consultas MySQL: quais as certas
400.	835a	Perguntas sobre xadrez	495.	d75a	Construindo e consultando um DB MySQL
401.	841a	Uma RNA (rede neural) de 8e e 4s	496.	d76a	Estatistica de transporte intermunicip
402.	845a	Algoritmo de retropropagacao	497.	d77a	Avaliacao de banco de dados
403.	846a	Numero de neuronios em RNA	498.	d80a	DER: organizacao de congressos
404.	847a	Rede neural: implementacao em Python	499.	d81a	DER: sistema de almoxarifado
405.	848a	Projeto de uma rede neural	500.	d85a	Banco de dados e contabilidade
406.	851a	Tensorflow	501.	d86a	Banco de dados e folha de pagamento
407.	856a	Frases logicas	502.	d87a	Banco de dados e controle de estoque
408.	857a	Demonstracao automatica de teoremas	503.	d89a	Engenhos no MySQL
409.	860b	Prolog: introducao e exercicios	504.	d90a	Backup e restore no MySQL
410.	860c	Prolog: familia pirlimpimpim	505.	d92a	MySQL: stored routines
411.	860d	Prolog: juncao de 860b+860c	506.	d93a	MySQL: triggers
412.	861a	Prolog 1 - exercicios de Silvio Lago	507.	d94a	MySQL: cursores e variaveis
413.	862a	Prolog 2 - exercicios de Silvio Lago	508.	d95a	MySQL: replicacao
414.	863a	Prolog 3 - exercicios de Silvio Lago	509.	d97a	MySQL: otimizacao
415.	864a	Prolog 4 - exercicios de Silvio Lago	510.	d98a	MySQL: dados espaciais
416.	865a	Prolog 5 - exercicios de Silvio Lago	511.	d99a	MySQL e PHP
417.	866a	Prolog 6 - exercicios de Silvio Lago	512.	e01a	Faca programa que... (e01)
418.	869a	Linguagens classicas da IA	513.	e03a	Faca programa que... (e03)
419.	870a	Basico de LISP	514.	e05a	Faca programa que... (e05)
420.	871a	Definicoes de funcao em LISP	515.	e07a	Faca programa que... (e07)
421.	872a	Mais funcoes em LISP	516.	e09a	Faca programa que... (e09)
422.	877a	0 que faz esta funcao LISP	517.	e11a	Faca programa que... (e11)
423.	878a	Escreva funcao LISP que...	518.	e99a	Auto aprendizado em programacao
424.	879a	Escrevendo software em LISP	519.	h07a	Interface via agentes virtuais
425.	891a	Representacao do conhecimento	520.	h08a	Interface de menus e icones
426.	892a	Perguntas FASE	521.	h16a	Metodologia de avaliacao de sites
427.	900a	Operacoes sobre conjuntos	522.	h20a	Tutorial HTML - 1
428.	901a	Divisibilidade e numeros primos	523.	h22a	Tutorial HTML - 2
429.	902a	Geometria analitica	524.	h24a	Tutorial HTML - 3
430.	903a	Einstein e a teoria da relatividade	525.	h26a	Meu primeiro site
431.	911a	Teorema de Bayes	526.	h28a	Erros em documentos HTML
432.	912a	Pratica em probabilidade	527.	h29a	Pratica em textos: vamos escrever ?
433.	913a	Redes bayesianas de crenca	528.	h30a	Meu segundo site
434.	922a	Estereogramas	529.	h32a	Tutorial HTML - 4
435.	925b	Imagem: alg de Horowitz e Pavlidis	530.	h34a	Tutorial HTML - 5
436.	925f	Imagem: 925+928	531.	h35a	Tabelas em HTML
437.	926a	Esqueletizacao de imagens	532.	h36a	Tutorial HTML - 6
438.	928a	Imagem: bloquinhos de waltz	533.	h38a	Tutorial HTML - 7
439.	930a	Listas em Python	534.	h40a	Tutorial HTML - 8
440.	931a	Strings em Python	535.	h50a	Tutorial CSS - 1
441.	932a	Escreva uma funcao que... (em Python)	536.	h51a	Introducao a CSS
442.	933a	Arquivos em Python	537.	h52a	Tutorial CSS - 2
443.	934a	Tuplas, dicionarios conjuntos em Pytho	538.	h53a	Organizando o layout com CSS
444.	935a	Objetos em Python	539.	h54a	Tutorial CSS - 3
445.	936a	Pratica em Numpy (pacote Python)	540.	h55a	Uso de DIV em CSS
446.	937a	Pratica em Matplotlib (pacote Python)	541.	h56a	Uso de biblioteca padrao: bootstrap
447.	938a	Processamento simbolico: sympy/Maple	542.	h60a	Tutorial Javascript - 1
448.	939a	Blocos em Python	543.	h61a	JS como linguagem de desktop
449.	942a	Computacao evolutiva (fls iguais/difs)	544.	h63a	Tutorial Javascript - 2
450.	943a	Computacao evol.: operadores ad hoc	545.	h66a	Tutorial Javascript - 3
451.	945a	Programacao genetica	546.	h69a	Tutorial Javascript - 4
452.	947a	Algoritmos geneticos: uso do TUDEL	547.	h72a	Tutorial Javascript - 5
453.	950a	Exercicio genesereth e nilsson	548.	h77a	Um site
454.	952a	Imagem: transformada de Hough	549.	h81a	Imagem GIF
455.	953a	Transformada de Fourier	550.	h82a	Espaço de cores programado em JS
456.	954a	Espaços de cores	551.	h83a	Caça niqueis HTML+CSS+JS
457.	965a	Biorritmos	552.	h84a	Conversoes em HTML+CSS+JS
458.	975a	Processamento em linguagem natural	553.	h85a	Folha de Pagamento em JS
459.	980a	Guia de apreciacao do filme COLOSSUS	554.	h86a	Varejao em JS
460.	985a	Como escrever um artigo cientifico	555.	h87a	3 Calculos elementares em JS
461.	998b	Mnemonico para numeros irracionais	556.	h88a	Funcoes inventadas apoiadas em JS
462.	998c	Algoritmo guloso: o problema do troco	557.	h89a	Operacoes com datas
463.	998d	Forca bruta: critoaritmética	558.	h90a	Exame final de design: um site
464.	a01a	Questionario: sistemas distribuidos	559.	h91a	Maquina de troco
465.	a05a	Questionario: modelos de S.I.	560.	h92a	Criptograma letras e numeros
466.	b02a	Questionario: empresa Google	561.	h93a	Leitura de constantes em JS
467.	b20a	3 MER feitos e pede despacho de viat.	562.	h94a	Diagonais da matriz

563.	h95j	Rodada de poker em JS	658.	n40a	Scripts de transacao em bitcoin
564.	h95p	Rodada de poker em Python	659.	n55a	Arvores de Merkle
565.	h96a	Formatos internos em JS	660.	n60a	Mineracao e prova de trabalho (bitcoin)
566.	h97j	Conjuntos em JS	661.	n65a	Filtro de Bloom
567.	h97p	Conjuntos em Python	662.	n75a	Bifurcacao e disputa de blockchain
568.	h98a	Mais programas em JS	663.	o00a	Matematica retirado da OBMEP
569.	h99a	Teorema Fundamental da Aritmetica	664.	o01b	Aquecimento em matematica - 1
570.	ha1a	Sequencia de Collatz	665.	o01c	Aquecimento em matematica - 2
571.	ha2a	Primos pitagoreanos	666.	o01d	Aquecimento em matematica - 3
572.	ha3a	Calculo do determinante	667.	o01e	Aquecimento em matematica - 4
573.	ha4a	Maior palavra	668.	o01f	Aquecimento em matematica - 5
574.	ha5a	MMC e MDC em JS	669.	p05a	Maple: introducao
575.	ha6a	Algoritmo Addition Chaining em JS	670.	p15a	Maple: matematica basica
576.	ha7a	Numeros feios (UVA) em JS	671.	p25a	Maple: algebra basica
577.	ha8a	Qual o numero com mais divisores ?	672.	p45a	Maple: programacao
578.	ha9a	Fatorial em termos dos fatores primos	673.	p55a	Maple: mais programacao
579.	hb1a	UVa 105 - Perfil da skyline	674.	p65a	Maple: exercicios de logica
580.	hb2a	UVa 108 - Maior subsuma em matriz	675.	p75a	Maple: graficos
581.	hb4a	UVa 103 - Encaixando caixas	676.	p76a	Maple: leitura de arquivos
582.	j10a	Um jogo simples app Inventor: PALITO	677.	p91a	Maple em 1 dia
583.	j12a	Um jogo de Pong usando app inventor	678.	q05b	Arquitetura de computadores I
584.	j14a	Mini golfe em app inventor	679.	q05c	Arquitetura de computadores II
585.	j21a	Jogo de general	680.	q06a	Um computador simples
586.	j39a	Sudoku para computadores	681.	q10a	Objetos binarios grandes - BLOBs
587.	j42a	app inventor: onde deixei meu carro ?	682.	q11a	Mais um BLOB: Tomografia
588.	j45a	app inventor: xilofone	683.	q20b	Redes de computadores
589.	j70a	Um jogo	684.	q20c	Modelo de camadas da Internet
590.	j71a	Quebra cabeça 2048	685.	q30b	Sistemas operacionais
591.	j72a	Tetris	686.	q32a	Representacao de numeros em maquinas
592.	k01a	Introducao a metodos numericos	687.	q37a	Algoritmo A* - aplicacoes
593.	k04a	Sistemas de numeracao	688.	q43a	Alocacao sequencial e encadeada
594.	k07a	Representacao interna de numeros	689.	q66a	Aprendizado profundo
595.	k08b	Repres. interna numeros maq. binarias	690.	q80a	Tabelas hash
596.	k08g	Armazenamento de flutuantes	691.	q81a	Grafos e caminho minimo
597.	k13a	Calc num: solucao de equacoes	692.	q82a	Compressor de dados
598.	k16a	Iteracao linear, secante e misto	693.	q83a	Caixeiro viajante
599.	k19a	Revisao de calculo numerico	694.	q84a	Criptografia
600.	k22a	Metodo de Gauss para sistemas lineares	695.	q86a	Metrô de Curitiba
601.	k25a	Matriz LU para sistemas lineares	696.	q91a	Arvores
602.	k28f	Sistemas lineares-Jacobi (freemat)	697.	q92a	Arvores-B
603.	k28p	Sistemas lineares-Jacobi (python)	698.	r20a	Cartela de pseudo codigo
604.	k31a	Gauss-Seidl e relaxacao	699.	r24a	Cartela de VisualG
605.	k34a	Problemas praticos de sistemas lineare	700.	r25a	Pratica de VisualG
606.	k37a	Tomografia usando sist. lineares	701.	r26a	Cartela de C
607.	k40f	Interpolacao polinomial - Octave	702.	r28a	Cartela de APL
608.	k40p	Interpolacao polinomial - Python	703.	r29a	Cartela de xBase
609.	k43f	Int. polin. Met. Lagrange - Octave	704.	r30a	Cartela de Prolog
610.	k43p	Int. polin. Met. Lagrange - Python	705.	r32a	Cartela de Lisp
611.	k46a	Int. polin. Met. Lagrange - Sympy	706.	r37a	Cartela de Python
612.	k49a	Polinomio de Newton e met. Diferencas	707.	r38a	Cartela de Matplotlib
613.	k52a	Polinomio de Newton-Gregory	708.	r39a	Cartela de Numpy
614.	k55a	Exercicios de interpolacao	709.	r41a	Cartela de Sympy
615.	k61f	Ajuste de curvas - Octave	710.	r49a	Cartela de Latex
616.	k61p	Ajuste de curvas - Python	711.	r50a	Cartela de HTML
617.	k64a	Integracao numerica	712.	r51a	Cartela de J
618.	k67a	Aplicacoes de integracao numerica	713.	r55a	Cartela de CSS
619.	k80a	Resumo pratico de calculo numerico	714.	r60a	Cartela de JavaScript
620.	k90a	Jupyter e markdown com python	715.	r70a	Cartela de Freemat/Octave
621.	m02a	Maratona UP2011	716.	r80a	Cartela de MySQL
622.	m04a	Marat: Fazenda de ostras	717.	t02a	Matematica: relacoes
623.	m06a	Marat: Univ. Pinguinho e cubos color.	718.	t04a	Matematica: funcoes e somatorios
624.	m08a	Marat: magico e azulejamento piscina	719.	t07c	Mat: ciclos de somatorio em C++
625.	m10a	Marat: cientistas e iluminacao	720.	t07p	Mat: ciclos de somatorio em Python
626.	m12a	Marat: jogo do tabuleiro (IOI-2005)	721.	t10a	Mat: progressoes aritmeticas e geometr
627.	m14a	Marat: caminhos e vulcao	722.	t13a	Mat: demonstracoes algebricas
628.	m16a	Marat: genoma e palavras cruzadas	723.	t16a	Mat: matrizes
629.	m18a	Marat: trem, robot e restaurante	724.	t17a	Mat: pratica de matrizes em python
630.	m20a	Marat: seq. H, carga e rede otica	725.	t19a	Mat: equacoes e sistemas lineares
631.	m22a	Marat: Quermesse, notas e saldo	726.	t22a	Mat: determinantes
632.	m24a	Marat: macaco e visita ao MASP	727.	t25a	Mat: coeficientes binomiais
633.	m26a	Marat: aneis, balaies e meteoros	728.	t30a	Mat: analise combinatoria
634.	m28a	Marat: domino, sorvete e piramide	729.	t35a	Mat: probabilidade
635.	m30a	Marat: tesouro, aeroporto e pedagogio	730.	t36a	Mat: pratica em probabilidade: primos
636.	m32a	Marat: bicicleta, supermer. e erdos	731.	t45a	Mat: variaveis aleatorias
637.	m34a	Marat: furos, senhas e orkut	732.	t50a	Mat: estatistica descritiva
638.	m36a	Marat: minhocas, furos e taxis	733.	t51a	Mat: pratica em estatistica descritiva
639.	m38a	Marat: colheita, linhas e desculpas	734.	t60a	Mat: distribuicao binomial e normal
640.	m40a	Marat: poker, margarida e truco	735.	t70a	Mat: aquecimento geometria analitica
641.	m42a	Marat: 3n+1, viagem e editor grafico	736.	t80a	Mat: conicas
642.	m44a	Marat: interpretador, palindr. fibonac	737.	t82a	Mat: exercicios de teoria conjuntos
643.	m46a	Marat: blocos, lixo e caixas	738.	t85a	Mat: aquecimento em calculo
644.	m48a	Marat: fermat, cat-in-hat, subretang	739.	t87a	Mat: taxas de variacao e derivadas
645.	m50a	Marat: arvores, criptografia e TSP	740.	t89a	Mat: derivadas relacionadas
646.	m52a	Marat: panquecas, tubos e arvores	741.	t90a	Mat: maximos e minimos
647.	m54a	Marat: num feios, permutacoes e fat-fa	742.	t92a	Mat: funcoes trigonometricas
648.	m56a	Marat: circunfer. intersecao e ciclo	743.	t94a	Mat: funcoes exponenciais
649.	m58a	Marat: milhario e vaiqueemole	744.	t96a	Mat: problemas de area revolutiva
650.	m60b	Maratona UP 2012 p1	745.	u02a	Compil: Notacao Polonesa Reversa
651.	m60c	Maratona UP 2012 p2	746.	u04a	Compil: Compilador aritmetico
652.	n02a	Problema dos generais bizantinos	747.	u05a	Compil: Compilador com reais e id long
653.	n05a	Autenticacao de mensagens (toy domain)	748.	u06a	Compil: Compilador Latex
654.	n10a	Cript. assimetrica de c. eliptica	749.	u08a	Compil: Compilador de se...fimse
655.	n20a	Numeros pseudo-randomicos	750.	u10a	Compil: Compilador de arquivo PDF
656.	n25a	Numeracao de base 58	751.	u11a	Compil: Utilitario extrator de PDFs
657.	n35a	Carteiras em bitcoin	752.	u12a	Compil: compilando expressoes logicas

753.	u16a	Compil: compilando o jogo da vida	67.	Aritmetica complexa	025a	12/01/2024
754.	u20a	Compil: compilando comandos repeticao	68.	Aritmetica decimal, binaria e hexad	036b	13/07/2011
755.	u24a	Compil: Maple -> C	69.	Aritmetica inteira básica	026a	20/01/2024
756.	u28a	Compil: tudo junto - U04+U08+U20	70.	Armazenamento de flutuantes	k08g	12/02/2022
757.	u42a	Compil: clone do vivx719	71.	Arquitetura de computadores I	q05b	05/05/2024
758.	u44a	Compil: compilador de imagens BMP	72.	Arquitetura de computadores II	q05c	05/05/2024
759.	z02a	Quebra cab: garotos e seus brinquedos	73.	Arquivos em Python	933a	23/07/2018
760.	z03a	Quebra cab: reveillon das mocas	74.	Artigo e consideracoes de Turing	800a	16/07/2005
761.	z04a	Quebra cab: meninas estudiosas	75.	Arvore QUAD	255a	04/02/2002
762.	z05a	Quebra cab: quem bebe agua e tem zebra	76.	Arvore binaria de partilha-ABPp	435a	04/08/2003
763.	z07a	Quebra cab: reveillon das mocas	77.	Arvore genealogica	273a	18/02/2002
764.	z08a	Quebra cab: lutadores de jiu jitsu	78.	Arvore: criacao de ABP	431a	11/03/2000
765.	z09a	Quebra cab: visita ao dentista	79.	Arvores	q91a	30/06/2024
766.	z10a	Quebra cab: Insanidade Instantanea	80.	Arvores com recursividade e apontadore	260b	17/10/2020
767.	z11a	Quebra cab: Blocos cientistas famosos	81.	Arvores com recursividade e cursores	260c	17/10/2020
768.	z15a	Quebra cab: Dissecacoes geometricas	82.	Arvores de Merkle	n55a	20/11/2017
769.	z19a	Quebra cab: A Terra recebe uma visita	83.	Arvores k-d	261a	28/06/2021
770.	z20a	Quebra cab: Sudoku para humanos	84.	Arvores: caminhamentos em 3 arvores	411a	07/08/2000
771.	z25a	Quebra cab: Visao lateral	85.	Arvores: inclusoes/exclusoes em ABPs	433a	07/08/2000
772.	z26a	Problemas aritmeticos	86.	Arvores: redesenhar balanceando	421a	14/07/2000
773.	z35a	Citale espartano	87.	Arvores-B	q92a	03/07/2024
774.	z41a	Construcao de uma regua de calculo	88.	Assinatura de arquivos	714a	14/10/2000
775.	z55a	Um problema de logistica	89.	Autenticacao de mensagens (toy domain)	n05a	10/11/2017
776.	z66a	Menor caminho usando estradas e pontes	90.	Auto aprendizado em programacao	e99a	28/11/2013
777.	z67a	Custo minimo usando 3 tipos de trens	91.	Autovalores e autovetores	078a	08/02/2024

2 Índice por ordem alfabetica

1.	10 documentos para normalizar	d30b	21/03/2016	97.	Banco de dados e contabilidade	d85a	28/05/2016
2.	10 problemas (lapis-papel) OBI2005	192a	19/12/2007	98.	Banco de dados e controle de estoque	d87a	29/07/2016
3.	10 problemas (papel-lapis) OBI2003	190a	17/12/2007	99.	Banco de dados e folha de pagamento	d86a	03/08/2016
4.	14 comandos de repeticao	042a	27/03/2009	100.	Banc 64, funcionamento e conversoes	037a	07/11/2012
5.	15 comandos condicionais	016b	13/03/2009	101.	Bases maiores do que 10	027a	28/11/2001
6.	21 comandos de repeticao	041a	26/03/2009	102.	Basico de LISP	870a	04/08/2005
7.	27 Comandos condicionais simples	015j	12/03/2009	103.	Bifurcacao e disputa de blockchain	n75a	04/12/2017
8.	3 Calculos elementares em JS	h87a	02/08/2024	104.	Bioinf. 1-maior subsequencia comum	536b	09/10/2014
9.	3 Estruturas de repeticao	064a	26/06/2008	105.	Biorritmos	965a	10/11/2011
10.	3 MER feitos e pede despacho de viat.	b20a	23/08/2011	106.	Biounf. 2-custos e busca padrao	536c	09/10/2014
11.	4 erros em listas encadeadas	332a	23/02/2001	107.	Blocos em Python	939a	31/07/2024
12.	5 problemas (lapis-papel) OBI2004	191b	18/12/2007	108.	Boleto bancario da FEBRABAN	024e	05/01/2015
13.	5 problemas (lapis-papel) OBI2004	191k	18/12/2007	109.	Busca cega: 5 exercicios	821a	01/11/2004
14.	6 problemas (lapis-papel) OBI2006	193c	23/12/2007	110.	Busca: Boyer+Moore simplificado	531a	24/01/2002
15.	6 problemas (lapis-papel) OBI2006	193d	23/12/2007	111.	Caça niqueis HTML+CSS+JS	h83a	13/07/2024
16.	7 algoritmos para seguir (chines)	019a	06/05/2009	112.	Caixeiro viajante	q83a	11/08/2024
17.	7 problemas (lapis-papel) OBI2006	193b	22/12/2007	113.	Calc num: solucao de equacoes	k13a	21/08/2016
18.	7 problemas (lapis-papel) OBI2006	193e	23/12/2007	114.	Calcnm - sist. lineares - Octave	735f	27/08/2016
19.	A*: caminho mais barato	831a	18/08/2014	115.	Calcnm - sist. lineares - Python	735p	24/03/2018
20.	A*: jarras com implementacao	826j	17/03/2012	116.	Calculo da envoltoria convexa	315a	13/01/2003
21.	A*: tarkin com implementacao	826t	17/03/2012	117.	Calculo de PI	045a	21/01/2024
22.	A*: tarkin e jarras	826b	17/01/2007	118.	Calculo do determinante	ha3a	25/09/2024
23.	A*: trajeto de um robot	830a	27/02/2014	119.	Caminhamento 1:1 (tennenbaum)	622a	05/05/1998
24.	ABPs: busca, criacao e altura	432a	18/06/2002	120.	Caminhamento em lar e prof em grafos	612a	10/10/1997
25.	Achar o decimo termo na sequencia	073a	04/06/2006	121.	Caminhamento n:n (floyd-warshall)	623a	20/02/1997
26.	Achar o n-esimo em 30 sequencias	009b	12/12/2007	122.	Caminho hamiltoniano: domino	636a	29/10/2005
27.	Ajuste de curvas - Octave	k61f	02/11/2016	123.	Caminho minimo em grafo (dijkstra)	621a	11/10/1998
28.	Ajuste de curvas - Python	k61p	30/04/2018	124.	Caminho minimo no metro de curitiba	627a	09/10/2009
29.	Alg. guloso/best/first/next/worst	271a	26/10/2001	125.	Carteiras em bitcoin	n35a	07/12/2017
30.	Algebra das proposicoes - Parte 1	098b	01/05/2013	126.	Cartela de APL	r28a	11/05/2011
31.	Algebra das proposicoes - Parte 2	098f	21/04/2012	127.	Cartela de C	r26a	04/05/2011
32.	Algebra de Boole	106a	15/01/2013	128.	Cartela de CSS	r55a	20/12/2015
33.	Algoritmo A* - aplicacoes	q37a	14/06/2024	129.	Cartela de Freemath/Octave	r70a	20/12/2015
34.	Algoritmo Addition Chaining em JS	ha6a	26/09/2024	130.	Cartela de HTML	r50a	20/12/2015
35.	Algoritmo Gale-Shapley de alocao	656a	17/01/2009	131.	Cartela de J	r51a	13/01/2020
36.	Algoritmo da raiz quadrada	003a	12/11/2007	132.	Cartela de JavaScript	r60a	20/12/2015
37.	Algoritmo de intercalacao	082a	30/12/2009	133.	Cartela de Latex	r49a	29/12/2010
38.	Algoritmo de quebra em relatorios	086a	15/01/2010	134.	Cartela de Lisp	r32a	07/09/2012
39.	Algoritmo de retropropagacao	845a	21/12/2011	135.	Cartela de Matplotlib	r38a	21/05/2018
40.	Algoritmo do torneio de tenis	660a	07/08/2007	136.	Cartela de MySQL	r80a	15/01/2016
41.	Algoritmo dos k-vizinhos	668a	29/07/2017	137.	Cartela de Numpy	r39a	22/05/2018
42.	Algoritmo guloso: o problema do troco	998c	20/02/2011	138.	Cartela de Prolog	r30a	01/04/2012
43.	Algoritmo húngaro de assinalamento	650a	01/05/2007	139.	Cartela de Python	r37a	06/03/2017
44.	Algoritmos geneticos: uso do TUDEL	947a	17/07/2007	140.	Cartela de Sympy	r41a	23/05/2018
45.	Alocacao sequencial e encadeada	q43a	22/06/2024	141.	Cartela de VisualG	r24a	08/03/2013
46.	Alto/baixo 1:conjuntos	222a	22/05/2021	142.	Cartela de pseudo codigo	r20a	08/03/2013
47.	Alto/baixo 2:pilha e fila	223a	06/06/2021	143.	Cartela de xBase	r29a	01/04/2012
48.	Alto/baixo 3:ABP	224a	07/06/2021	144.	Casamento aproximado de cadeias	535a	05/06/2008
49.	Alto/baixo 4:assembly e p. alto nivel	225a	08/06/2021	145.	Casamento estavel	256a	11/04/2006
50.	Alto/baixo 5:matriz inversa	226a	10/06/2021	146.	Caso pratico de huffman ad-hoc	456a	11/07/2011
51.	Analise estilistica de textos	560a	24/06/2014	147.	Central 0800 com menores custos	646a	24/06/2013
52.	Aplicacoes da Algebra Linear	079a	10/02/2024	148.	Certificacao digital pratica	705a	01/09/2010
53.	Aplicacoes de integracao numerica	k67a	24/05/2018	149.	Certificar 5 algoritmos basicos	125b	11/07/2007
54.	Aplicativos comerciais: contabilidade	720a	19/12/2008	150.	Certificar 5 algoritmos de vetor	125c	11/07/2007
55.	Aplicativos comerciais: estoques	723a	30/12/2008	151.	Certificar 5 algoritmos parte 4	125e	27/07/2007
56.	Aplicativos comerciais: folha de pag.	721a	20/12/2008	152.	Certificar 5 algoritmos vetor + difs	125d	12/07/2007
57.	Aprendizado profundo	q66a	10/06/2024	153.	Chines de 4 algoritmos	077a	21/02/2000
58.	Aquecimento em arvores B	441a	02/07/2002	154.	Chines de 4 algoritmos aleatorios	069a	29/02/2000
59.	Aquecimento em matematica - 1	o01b	19/10/2016	155.	Cifra de Playfair	708a	23/07/2011
60.	Aquecimento em matematica - 2	o01c	21/10/2016	156.	Cifra de Viginere	709a	13/09/2010
61.	Aquecimento em matematica - 3	o01d	31/10/2016	157.	Citale espartano	z35a	03/10/2014
62.	Aquecimento em matematica - 4	o01e	09/11/2016	158.	Codigo ADFGVX	717a	03/09/2003
63.	Aquecimento em matematica - 5	o01f	13/11/2016	159.	Codigos Reed Solomon	570a	27/01/2015
64.	Argumentos e validade	099b	05/05/2012	160.	Codigos: ASCII, Ortelino e Trocalettra	450a	13/03/2001
65.	Argumentos e validade ad-hoc	099h	12/05/2012	161.	Coloracao de grafos	635a	19/08/2003
66.	Argumentos literais e validade	099j	09/03/2016				

162.	Coloracao de grafos: aplicacoes	639a	08/10/2019	257.	Engenharia reversa	d32a	08/02/2016
163.	Comandos SE (apenas codigo)	015d	10/02/2011	258.	Engenharia reversa de algoritmos	062a	31/01/2006
164.	Comandos SE com complexidade crescente	020a	17/04/2012	259.	Engenharia reversa de compiladores	319a	02/02/2006
165.	Comandos SE simples	015b	08/02/2011	260.	Engenharia reversa: construir o MER	b45a	29/08/2011
166.	Comandos SEs aninhados	016h	05/12/2014	261.	Engenhos no MySQL	d89a	18/08/2016
167.	Comandos de repeticao simples	040a	11/02/2011	262.	Enquanto e para basicos	023a	08/10/2012
168.	Como escrever um artigo cientifico	985a	17/09/2019	263.	Equacoes diferenciais	351a	24/03/2024
169.	Compara buscas em tabelas	532a	01/09/2003	264.	Equacoes e inequacoes	063a	03/02/2024
170.	Compil: Compilador Latex	u06a	03/12/2013	265.	Equalizacao de histograma em imagens	755a	10/05/2000
171.	Compil: Compilador aritmetico	u04a	16/11/2013	266.	Equivalencia e reducao ao absurdo	097a	24/03/2012
172.	Compil: Compilador com reais e id long	u05a	21/12/2013	267.	Erros em documentos HTML	h28a	11/11/2014
173.	Compil: Compilador de arquivo PDF	u10a	20/11/2013	268.	Escreva 10 programas C++	044z	11/05/2024
174.	Compil: Compilador de se...fimse	u08a	19/11/2013	269.	Escreva funcao LISP que...	878a	28/08/2004
175.	Compil: Maple -> C	u24a	03/12/2013	270.	Escreva uma funcao que... (em Python)	932a	20/07/2018
176.	Compil: Notacao Polonesa Reversa	u02a	15/02/2014	271.	Escrevendo software em LISP	879a	30/06/2005
177.	Compil: Utilitario extrator de PDFs	u11a	14/05/2014	272.	Espaco de cores programado em JS	h82a	17/06/2024
178.	Compil: clone do vivx719	u42a	30/08/2014	273.	Espagos de cores	954a	22/10/2021
179.	Compil: compilador de imagens BMP	u44a	30/08/2014	274.	Esqueletizacao de imagens	926a	15/07/2005
180.	Compil: compilando comandos repeticao	u20a	29/11/2013	275.	Estatistica de transporte intermunicip	d76a	30/04/2016
181.	Compil: compilando expressoes logicas	u12a	28/11/2013	276.	Esteganografia linear em BMP	719a	12/12/2010
182.	Compil: compilando o jogo da vida	u16a	29/11/2013	277.	Esteganografia: msg dentro de BMP	713a	03/04/2002
183.	Compil: tudo junto - U04+U08+U20	u28a	02/12/2013	278.	Estereogramas	922a	17/04/2005
184.	Compilador de pseudo codigo	318a	18/12/2005	279.	Exame final de design: um site	h90a	22/10/2015
185.	Compl. alg. II	146a	03/12/2018	280.	Exemplos de pilhas	311a	02/05/2000
186.	Compl. alg. III	147a	08/12/2018	281.	Exercicio genesereth e nilsson	950a	30/07/2004
187.	Compl. alg. IV - arvore Merkle	148a	26/12/2018	282.	Exercicios de interpolacao	k55a	08/05/2019
188.	Compl. alg. V - sist lin e caixeiro	149a	01/01/2019	283.	Exponenciacao, diagonais e trelicas	035c	29/04/2014
189.	Completando algoritmos	145a	30/11/2018	284.	Expressoes regulares	538a	02/12/2014
190.	Complexidade, recursao e rel. recorr.	220a	31/05/2021	285.	Faca programa que... (e01)	e01a	22/03/2013
191.	Complexidade: busca binaria e recursao	666b	27/07/2017	286.	Faca programa que... (e03)	e03a	24/03/2013
192.	Complexidade: habilidades de booleiro	233a	18/01/2001	287.	Faca programa que... (e05)	e05a	31/03/2013
193.	Complexidade: o problema da particao	231a	17/07/2002	288.	Faca programa que... (e07)	e07a	11/04/2013
194.	Complexidade: requisitos de automovel	232a	21/01/2001	289.	Faca programa que... (e09)	e09a	11/05/2013
195.	Compressao (obsoleta 766 e 767)	768a	26/08/2010	290.	Faca programa que... (e11)	e11a	18/06/2013
196.	Compressao JPG em C	690c	07/08/2013	291.	Fatorial em termos dos fatores primos	ha9a	05/10/2024
197.	Compressao JPG em Python	690p	07/08/2013	292.	Fechamento de bases nao decimais	036f	07/06/2013
198.	Compressao de dados	q82a	10/08/2024	293.	Ferramentas CASE	d14a	26/08/2011
199.	Computacao evol.: operadores ad hoc	943a	25/10/2003	294.	Filas circulares: impl. e exemplos	322a	12/06/2000
200.	Computacao evolutiva (fls iguais/difs)	942a	20/03/2001	295.	Filas em assembler	323a	03/05/2003
201.	Computadores auto-replicantes	807a	03/02/2015	296.	Filas: implementacao e exemplos	321a	12/06/2000
202.	Conceito introdutorio de arvore	131a	23/07/2008	297.	Filtro de Bloom	669a	31/08/2017
203.	Conceitos do modelo E-R	d10a	14/08/2011	298.	Filtro de Bloom	n65a	20/11/2017
204.	Concolucoes com mascaras em imagens	754a	30/03/2000	299.	Fluxos ford-fulkerson/edmonds-karp	632a	15/07/2002
205.	Condicionais aninhadas	028b	11/01/2018	300.	Folha de Pagamento em JS	h85a	18/07/2024
206.	Condicionais aninhadas	028e	12/01/2018	301.	Forca bruta: critoaritmetica	998d	15/10/2011
207.	Condicionais aninhadas e compostas II	028k	20/01/2018	302.	Formatos internos em JS	h96a	16/09/2024
208.	Condicionais aninhadas e compostas III	028n	22/01/2018	303.	Fractal de Mandelbroot	732a	03/04/1999
209.	Condicionais em C++	046b	15/03/2018	304.	Frases logicas	856a	27/07/2004
210.	Condicionais em C++ II	046d	17/03/2018	305.	Funcoes SE, ENQUANTO e REPITA	065a	20/04/2006
211.	Condicionais em Python	046p	16/03/2018	306.	Funcoes de data e hora em MySQL	d35c	15/07/2016
212.	Condicionais em Python II	046q	17/03/2018	307.	Funcoes de string em MySQL	d35b	14/07/2016
213.	Condicionais literais	029a	23/02/2013	308.	Funcoes inventadas + mat basica	002b	10/11/2012
214.	Conjuntos em JS	h97j	16/09/2024	309.	Funcoes inventadas II + mat basica	002c	10/11/2012
215.	Conjuntos em Python	h97p	25/10/2024	310.	Funcoes inventadas III + mat basica	002d	11/11/2012
216.	Construcao de tabela verdade	095b	04/01/2015	311.	Funcoes inventadas IV + mat basica	002e	12/11/2012
217.	Construcao de tabela verdade II	095f	27/02/2016	312.	Funcoes inventadas V + mat basica	002f	13/11/2012
218.	Construcao de uma regua de calculo	z41a	23/09/2019	313.	Funcoes inventadas VI + mat basica	002g	26/11/2012
219.	Construindo e consultando um DB MySQL	d75a	12/04/2016	314.	Funcoes inventadas apoiadas em JS	h88a	07/08/2024
220.	Consultas MySQL e respostas: casar	d71b	06/02/2016	315.	Gauss-Seidl e relaxacao	k31a	02/11/2021
221.	Consultas MySQL: quais as certas	d71c	06/02/2016	316.	Geografia terrestre: dist. e cap.Grant	010a	18/11/2008
222.	Continuar sequencias	057a	30/12/2005	317.	Geometria analitica	902a	06/01/2010
223.	Conversao de arvore n-aria em binaria	412a	01/07/2002	318.	Geometrias não euclidianas	066a	25/01/2024
224.	Conversoes em HTML+CSS+JS	h84a	15/07/2024	319.	Gerador de movimentos em xadrez-C++	050c	06/12/2019
225.	Correcao de erros - alg. Hadamard	701a	20/01/2002	320.	Gerador de movimentos em xadrez-Python	050p	06/12/2019
226.	Corte alfa-beta do minimax	825a	19/07/2005	321.	Grafo: matriz de roteamentos	624a	30/03/1997
227.	Criacao de um BD no ACCESS	b60a	23/08/2011	322.	Grafos e caminho minimo	q81a	09/08/2024
228.	Cript. assimetrica de c. eliptica	n10a	28/11/2017	323.	Grupos algébricos	061a	27/01/2024
229.	Criptografia	q84a	18/09/2024	324.	Guia de apreciacao do filme COLOSSUS	980a	23/03/2009
230.	Criptografia - algoritmo TEA	711a	20/08/2010	325.	Hashing	521a	11/09/1999
231.	Criptografia El Gamal	718a	21/05/2004	326.	Heap	461a	22/02/2002
232.	Criptografia baseada no alg. DES	710a	10/05/2001	327.	Huffman - arvores diferentes	452a	01/01/1994
233.	Criptografia de chave publica - RSA	712a	03/04/2001	328.	Huffman - arvores iguais	451a	01/01/1993
234.	Criptograma letras e numeros	h92a	18/08/2024	329.	Huffman arvore dinamica	453a	16/02/2004
235.	Custo minimo usando 3 tipos de trens	z67a	21/11/2012	330.	Huffman sobre uma imagem BMP	457a	21/06/2012
236.	DER: organizacao de congressos	d80a	08/02/2016	331.	Imagem BMP: descobrir linhas cinza	752a	03/03/1999
237.	DER: sistema de almoxarifado	d81a	08/02/2016	332.	Imagem BMP: descobrir linhas coloridas	753a	20/04/1999
238.	Define vetor, registro e matriz	114a	29/07/2008	333.	Imagem GIF	h81a	14/06/2024
239.	Definicoes da Inteligencia Artificial	809a	15/01/2003	334.	Imagem: 925+928	925f	13/07/2013
240.	Definicoes de funcao em LISP	871a	26/09/2014	335.	Imagem: alg de Horowitz e Pavlidis	925b	02/02/2005
241.	Dem. autom. teo. e PROLOG (clone 858)	104a	10/08/2012	336.	Imagem: bloquinhos de waltz	928a	23/08/2003
242.	Demonstracao automatica de teoremas	103a	19/07/2012	337.	Imagem: transformada de Hough	952a	25/10/2020
243.	Demonstracao automatica de teoremas	857a	21/07/2004	338.	Inc/Exc em 3 arvores binarias de pesq.	434a	17/07/2008
244.	Demonstracoes de argumentos	100a	19/05/2012	339.	Indices em SYBASE (arvores)	443a	22/11/2004
245.	Descricao do GPS e simulacao	007a	02/01/2006	340.	Indices no padrao XBASE	444a	10/08/2001
246.	Desenhando funcoes	075a	13/02/2024	341.	Iniciando na programacao	072b	30/05/2024
247.	Desenho de grafos (graphviz)	618a	07/05/2008	342.	Iniciando na programacao II	072d	30/05/2024
248.	Diagonais da matriz	h94a	21/08/2024	343.	Iniciando na programacao III	072e	14/06/2024
249.	Diagrama PAGE para sistemas de IA	810a	07/03/2013	344.	Iniciando na programacao IV	072g	21/06/2024
250.	Diametro de grafo com listas adjacente	611a	14/07/2002	345.	Iniciando na programacao V	072j	03/07/2024
251.	Digitos verificadores	024b	04/02/2001	346.	Iniciando na programacao VI	072k	06/07/2024
252.	Distribuicao de cadeiras no legislativ	058a	18/04/2014	347.	Int. polin. Met. Lagrange - Octave	k43f	11/10/2016
253.	Diversos SEs	022a	01/04/2012	348.	Int. polin. Met. Lagrange - Python	k43p	11/10/2016
254.	Divisibilidade e numeros primos	901a	14/07/2009	349.	Int. polin. Met. Lagrange - Sympy	k46a	13/11/2021
255.	Einstein e a teoria da relatividade	903a	16/07/2011	350.	Integracao numerica	k64a	26/10/2016
256.	Encadeamento em um teatro	272a	15/05/2001	351.	Integridade de funcoes C	030a	15/07/2013

352.	Integridade referencial em BD	d63a	29/03/2016	447.	Mat: analise combinatoria	t30a	05/02/2015
353.	Interface de menus e icones	h08a	21/11/2014	448.	Mat: aquecimento em calculo	t85a	15/09/2020
354.	Interface via agentes virtuais	h07a	13/11/2014	449.	Mat: aquecimento geometria analitica	t70a	12/09/2014
355.	Interpolacao polinomial - Octave	k40f	29/09/2016	450.	Mat: ciclos de somatorio em C++	t07c	21/02/2019
356.	Interpolacao polinomial - Python	k40p	16/04/2018	451.	Mat: ciclos de somatorio em Python	t07p	05/05/2017
357.	Introducao a CSS	h51a	28/04/2024	452.	Mat: coeficientes binomiais	t25a	02/09/2014
358.	Introducao a Python	080a	05/10/2022	453.	Mat: conicas	t80a	29/10/2014
359.	Introducao a algoritmos	004a	20/12/2005	454.	Mat: demonstracoes algebricas	t13a	02/08/2021
360.	Introducao a metodos numericos	k01a	06/12/2021	455.	Mat: derivadas relacionadas	t89a	25/09/2020
361.	Introducao a programacao	056a	13/07/2019	456.	Mat: determinantes	t22a	01/09/2014
362.	Introducao a programacao - proj Euler	059a	16/08/2019	457.	Mat: distribuicao binomial e normal	t60a	30/09/2014
363.	Introducao ao modelo relacional	d60a	25/12/2015	458.	Mat: equacoes e sistemas lineares	t19a	31/08/2014
364.	Iteracao linear, secante e misto	k16a	16/12/2021	459.	Mat: estatistica descritiva	t50a	11/09/2014
365.	JS como linguagem de desktop	h61a	22/05/2024	460.	Mat: exercicios de teoria conjuntos	t82a	05/12/2015
366.	Jogo da vida	006a	23/10/2007	461.	Mat: funcoes exponenciais	t94a	30/09/2020
367.	Jogo de general	j21a	01/06/2024	462.	Mat: funcoes trigonometricas	t92a	29/09/2020
368.	Juncao do 511 e 512: nao ha exercicio	511b	01/11/2009	463.	Mat: matrizes	t16a	29/08/2014
369.	Jupyter e markdown com python	k90a	12/02/2022	464.	Mat: maximos e minimos	t90a	29/09/2020
370.	Kruskal: arv de espalhamento minima	634a	18/01/2002	465.	Mat: pratica de matrizes em python	t17a	11/05/2017
371.	Kumon de algoritmos	054a	23/06/2006	466.	Mat: pratica em estatistica descritiva	t51a	10/06/2017
372.	LZW: compressao e descompressao	731a	20/10/2003	467.	Mat: pratica em probabilidade: primos	t36a	01/02/2015
373.	Leitura de constantes em JS	h93a	19/08/2024	468.	Mat: probabilidade	t35a	06/02/2015
374.	Leitura de dados preexistentes Python	076c	28/09/2017	469.	Mat: problemas de area revolutiva	t96a	05/10/2020
375.	Leitura de dados preexistentes em C	076b	12/08/2014	470.	Mat: progressoes aritmeticas e geometr	t10a	27/08/2014
376.	Linguagens classicas da IA	869a	29/08/2017	471.	Mat: taxas de variacao e derivadas	t87a	20/09/2020
377.	Listas dupl.encadeadas em assembler	341a	29/04/2001	472.	Mat: variaveis aleatorias	t45a	12/09/2014
378.	Listas duplamente encadeadas	342a	01/08/2000	473.	Matematica Basica	005a	29/09/2012
379.	Listas em Python	930a	18/07/2018	474.	Matematica e logica basicas	011b	01/09/2006
380.	Listas skip	781a	06/02/2003	475.	Matematica e logica basicas (form C)	011c	12/03/2018
381.	Livros em biblioteca: exemplo de lista	333a	01/08/2000	476.	Matematica retirado da OBMEP	o00a	18/10/2016
382.	Localizacao de oleoduto	035b	27/12/2002	477.	Matematica: funcoes e somatorios	t04a	10/09/2014
383.	Logaritmos	215a	06/01/2003	478.	Matematica: relacoes	t02a	09/09/2014
384.	Logica de programacao: ABP,m.q.,lombad	150a	02/11/2019	479.	Matriz LU para sistemas lineares	k25a	09/07/2018
385.	Logica e Teoria dos conjuntos	105a	27/12/2012	480.	Matriz inversa - conceito	159a	30/08/2018
386.	Logica e diagramas de Venn	101b	18/07/2012	481.	Matriz inversa em C++	158c	30/05/2018
387.	Logica e quantificadores	102a	19/07/2012	482.	Matriz inversa em Python	158p	30/05/2018
388.	Logica proposicional	093a	21/01/2012	483.	Matrizes	128a	22/08/2006
389.	Logicas probabilistica e nebulosa	108a	18/01/2013	484.	Matrizes em Python	121p	06/07/2016
390.	MER: companhia aerea e revenda	d12a	07/03/2016	485.	Menor caminho usando estradas e pontes	z66a	12/04/2012
391.	MMC e MDC em JS	ha5a	26/09/2024	486.	Metodo de Gauss para sistemas lineares	k22a	09/07/2018
392.	Maior palavra	ha4a	25/09/2024	487.	Metodologia de avaliacao de sites	h16a	12/11/2014
393.	Mais 10 documentos para normalizar	d30c	21/03/2016	488.	Metró de Curitiba	q86a	02/09/2024
394.	Mais funcoes em LISP	872a	26/09/2014	489.	Meu primeiro site	h26a	19/12/2014
395.	Mais problemas de inteiros	014a	01/07/2009	490.	Meu segundo site	h30a	19/12/2014
396.	Mais problemas envolvendo inteiros	013a	05/02/2009	491.	Mineracao e prova de trabalho (bitcoin)	n60a	18/11/2017
397.	Mais programas em JS	h98a	17/09/2024	492.	Mini golfe em app inventor	j14a	04/06/2024
398.	Mais repeticao	047a	19/04/2011	493.	Minimax	824a	27/01/2002
399.	Mais um BLOB: Tomografia	q11a	05/06/2024	494.	Minimos quadrados	245a	25/11/2010
400.	Manipulação de datas	031a	25/02/2002	495.	Mnemonico para numeros irracionais	998b	16/02/2011
401.	Manipulação de datas: vetor hoj-ama	034a	08/11/2012	496.	Modelo de camadas da Internet	q20c	06/05/2024
402.	Manipulacoes no cubo Rubik	135a	21/09/2006	497.	Multiplicacao de matrizes em serie	671a	25/09/2015
403.	Manuseio de tabelas	119a	10/08/2006	498.	Mundo dos aspiradores de po	814a	05/03/2003
404.	Maple em 1 dia	p91a	14/05/2015	499.	MySQL e PHP	d99a	20/09/2016
405.	Maple: algebra basica	p25a	21/02/2012	500.	MySQL: cursores e variaveis	d94a	30/08/2016
406.	Maple: exercicios de logica	p65a	05/06/2012	501.	MySQL: dados espaciais	d98a	20/09/2016
407.	Maple: graficos	p75a	21/07/2012	502.	MySQL: otimizacao	d97a	21/09/2016
408.	Maple: introducao	p05a	18/02/2012	503.	MySQL: replicacao	d95a	21/09/2016
409.	Maple: leitura de arquivos	p76a	25/09/2012	504.	MySQL: resumo e exercicio	d66a	18/10/2011
410.	Maple: mais programacao	p55a	09/05/2012	505.	MySQL: stored routines	d92a	15/08/2016
411.	Maple: matematica basica	p15a	20/07/2012	506.	MySQL: triggers	d93a	24/08/2016
412.	Maple: programacao	p45a	14/04/2012	507.	Normalizacao de relacoes	d62a	21/01/2016
413.	Maquina ENIGMA	715a	08/01/2003	508.	Normalizacao em MER	b27a	08/09/2011
414.	Maquina de Turing	802a	30/01/2003	509.	Numeracao de base 58	n25a	15/11/2017
415.	Maquina de troco	h91a	17/08/2024	510.	Numeracao romana	144a	16/09/2006
416.	Marat: 3n+1, viagem e editor grafico	m42a	18/06/2008	511.	Numero de neuronios em RNA	846a	23/12/2011
417.	Marat: Fazenda de ostras	m04a	24/09/2006	512.	Numeros feios (UVA) em JS	ha7a	26/09/2024
418.	Marat: Quermesse, notas e saldo	m22a	04/01/2008	513.	Numeros primos	160a	03/12/2015
419.	Marat: Univ. Pinguinho e cubos color.	m06a	07/10/2006	514.	Numeros pseudo-randomicos	780a	17/01/2002
420.	Marat: aneis, balaos e meteoros	m26a	08/01/2008	515.	Numeros pseudo-randomicos	n20a	16/11/2017
421.	Marat: arvores, criptografia e TSP	m50a	07/08/2009	516.	O problema da parada (3 descricoes)	785a	09/08/2021
422.	Marat: bicicleta, supermer. e erdos	m32a	18/01/2008	517.	O que faz esta funcao LISP	877a	08/06/2003
423.	Marat: blocos, lixo e caixas	m46a	03/08/2009	518.	O que faz o codigo em C++	043c	18/05/2024
424.	Marat: caminhos e vulcao	m14a	11/08/2007	519.	O que faz o codigo em pseudocodigo	043e	18/05/2024
425.	Marat: cientistas e iluminacao	m10a	03/08/2007	520.	Objetos binarios grandes - BLOBs	q10a	08/05/2024
426.	Marat: circunfer. interseccao e ciclo	m56a	12/08/2009	521.	Objetos em Python	935a	18/10/2019
427.	Marat: colheita, linhas e desculpas	m38a	23/01/2008	522.	Onde por o hipermercado ?	645a	14/12/2010
428.	Marat: domino, sorvete e piramide	m28a	09/01/2008	523.	Operacoes com datas	h89a	16/08/2024
429.	Marat: fermat, cat-in-hat, subretang	m48a	05/08/2009	524.	Operacoes com vetores e indirecoes	110c	18/02/2001
430.	Marat: furos, senhas e orkut	m34a	22/01/2008	525.	Operacoes com vetores e indirecoes	110v	18/02/2001
431.	Marat: genoma e palavras cruzadas	m16a	31/12/2007	526.	Operacoes sobre conjuntos	900a	11/07/2009
432.	Marat: interpretador, palindr. fibonac	m44a	22/06/2008	527.	Ordenacao simples O(n**2)	132a	13/06/2011
433.	Marat: jogo do tabuleiro (IOI-2005)	m12a	09/11/2007	528.	Ordenacao topologica usando filas	324a	30/05/2000
434.	Marat: macaco e visita ao MASP	m24a	08/01/2008	529.	Organizando o layout com CSS	h53a	29/04/2024
435.	Marat: magico e azulejamento piscina	m08a	28/10/2006	530.	Otimizacao de algoritmos em C++	162c	31/05/2018
436.	Marat: milhario e vaiqueemole	m58a	22/06/2008	531.	Otimizacao de algoritmos em Python	162p	08/06/2017
437.	Marat: minhocas, furos e taxis	m36a	22/01/2008	532.	Otimizacao de funcoes	089a	17/02/2024
438.	Marat: num feios, permutacoes e fat-fa	m54a	10/08/2009	533.	PERT-CPM	640a	15/02/2006
439.	Marat: panquecas, tubos e arvores	m52a	08/08/2009	534.	Padrao FITS	760a	31/03/2017
440.	Marat: poker, margarida e truco	m40a	23/01/2008	535.	Padrao GIF: buscar coisas em arquivos	756a	28/08/2000
441.	Marat: seq. H, carga e rede otica	m20a	04/01/2008	536.	Padrao PDF - construa do PDF a mao	680a	09/08/2013
442.	Marat: tesouro, aeroporto e pedagio	m30a	17/01/2008	537.	Perdidos na lua (decisoes coletivas)	799a	18/02/2006
443.	Marat: trem, robot e restaurante	m18a	03/01/2008	538.	Perguntas FASE	892a	20/06/2003
444.	Maratona UP 2012 p1	m60b	14/05/2012	539.	Perguntas sobre xadrez	835a	24/03/2006
445.	Maratona UP 2012 p2	m60c	17/05/2012	540.	Perolas: complexos,pikachu e sierpinsk	175a	09/07/2021
446.	Maratona UP2011	m02a	09/05/2011	541.	Pesquisa em vetores - parte 1	116b	07/08/2006

542.	Pesquisa em vetores - parte 2	116c	20/05/2011	637.	Quebra de RSA (alg. rho pollard)	716a	09/01/2003
543.	Petri nets: descobrir caracateristicas	763b	05/05/2000	638.	Questionario: empresa Google	b02a	26/05/2010
544.	Petri nets: transacoes em um somador	763c	05/10/2000	639.	Questionario: introducao a BD	d05a	08/08/2011
545.	Petri nets: transacoes em uma ATM	763d	06/07/2001	640.	Questionario: modelos de S.I.	a05a	09/06/2010
546.	Pilhas em assembler	312a	02/05/2003	641.	Questionario: sistemas distribuidos	a01a	18/05/2010
547.	Pilhas: laberinto do ratinho	314a	19/05/2000	642.	Questoes sobre o artigo de Turing	805a	23/11/2005
548.	Polinomio de Newton e met. Diferencas	k49a	14/11/2021	643.	RPN-notacao polonesa reversa	313a	02/05/2000
549.	Polinomio de Newton-Gregory	k52a	15/11/2021	644.	Reconstituicao de imagem com tomografi	750a	03/11/2000
550.	Pratica de EXSINTA	817a	24/05/2005	645.	Recursividade: 3 funcoes	254a	24/07/2003
551.	Pratica de VisualG	r25a	11/04/2008	646.	Recursividade: Strassen e quickselect	262a	16/10/2018
552.	Pratica de programacao	051d	16/05/2017	647.	Recursividade: anagramas	252a	18/04/2000
553.	Pratica de programacao II	051e	03/08/2018	648.	Recursividade: funcoes A, B e C	251a	18/04/2000
554.	Pratica de programacao III	053a	10/07/2019	649.	Recursividade: passeio do cavalo	253a	18/04/2000
555.	Pratica de programacao inicial C++	051c	24/03/2019	650.	Recursividade: sudoku para computadores	257a	08/09/2006
556.	Pratica de programacao inicial Python	051p	23/03/2019	651.	Rede neural: implementacao em Python	847a	29/10/2019
557.	Pratica em ACCESS	b63a	15/09/2011	652.	Redes bayesianas de crenca	913a	13/09/2017
558.	Pratica em MER: Ultima tarefa	b59a	29/10/2011	653.	Redes de comparacao	513a	02/01/2003
559.	Pratica em MER: escolas do parana	b40a	29/08/2011	654.	Redes de computadores	q20b	06/05/2024
560.	Pratica em MER: locadora de veiculos	b55a	29/08/2011	655.	Regressao multipla	246a	17/07/2014
561.	Pratica em MER: restaurante	b30a	22/10/2010	656.	Repeticao em C++	048c	27/03/2018
562.	Pratica em MER: sist passagens aereas	b50a	29/08/2011	657.	Repeticao em Python	048p	27/03/2018
563.	Pratica em Matplotlib (pacote Python)	937a	26/07/2018	658.	Repres. interna numeros maq. binarias	k08b	21/10/2022
564.	Pratica em Numpy (pacote Python)	936a	04/06/2019	659.	Representacao de numeros em maquinas	q32a	15/06/2024
565.	Pratica em conjuntos	120a	30/03/2015	660.	Representacao do conhecimento	891a	10/02/2005
566.	Pratica em eixos cartesianos	140a	28/06/2013	661.	Representacao estruturas bidimensionai	555a	23/06/2024
567.	Pratica em huffman: os lusiadas	458a	10/07/2017	662.	Representacao interna de numeros	k07a	13/12/2021
568.	Pratica em matrizes	129a	30/05/2013	663.	Restricoes de integridade em BD	d64a	24/01/2016
569.	Pratica em probabilidade	912a	31/12/2016	664.	Resumo pratico de calculo numerico	k80a	28/01/2022
570.	Pratica em programacao II	052a	26/05/2017	665.	Revisao de calculo numerico	k19a	15/09/2016
571.	Pratica em tabelas verdade	095h	13/02/2017	666.	Rodada de poker em JS	h95j	11/09/2024
572.	Pratica em textos: vamos escrever ?	h29a	03/01/2015	667.	Rodada de poker em Python	h95p	26/10/2024
573.	Pratica em vetores (Freemat/Octave)	112a	19/05/2016	668.	Rotas e tempos minimos	628a	31/08/2012
574.	Pratica matrizes em C++	157b	28/05/2018	669.	Roteiro para apresentacao da pos grad.	798a	25/07/2009
575.	Pratica matrizes em C++ - II	157c	28/05/2018	670.	SEs compostos e encadeados	017b	28/01/2006
576.	Pratica matrizes em Python	157p	28/05/2018	671.	SGBDs antigos: inc/excl em LDEs	343a	17/05/2010
577.	Pratica matrizes em Python - II	157q	28/05/2018	672.	Sao equivalentes as condicionais-C++	170c	06/03/2019
578.	Primeiros passos em programacao C++	049b	04/08/2019	673.	Sao equivalentes as condicionais-Pytho	170p	06/03/2019
579.	Primeiros passos em programacao II-C++	049d	04/08/2019	674.	Scripts de transacao em bitcoin	n40a	02/12/2017
580.	Primeiros passos em programacao Python	049p	01/08/2019	675.	Se...entao: proposicao X comando	090a	27/02/2017
581.	Primeiros passos programacao Python-II	049q	03/08/2019	676.	Seguir chinses	021a	03/04/2006
582.	Primos pitagoreanos	ha2a	24/09/2024	677.	Seguir o codigo: repeticoes	067a	27/05/2024
583.	Problema da mochila 0-1 (knapsack)	240a	27/01/2009	678.	Seguranca e direitos em MySQL	d69b	13/04/2016
584.	Problema dos generais bizantinos	n02a	09/11/2017	679.	Selecao de atividades	266a	30/09/2019
585.	Problemas aritmeticos	z26a	19/09/2016	680.	Semantica em logica proposicional	096a	21/03/2012
586.	Problemas envolvendo inteiros	012a	31/01/2009	681.	Sequencia de Collatz	ha1a	20/09/2024
587.	Problemas matematicos	008a	29/08/2006	682.	Sequencias de numeracao	038a	28/12/2012
588.	Problemas praticos de sistemas lineare	k34a	30/03/2018	683.	Sequencias e series	088a	24/02/2024
589.	Processamento em linguagem natural	975a	19/07/2004	684.	Simulacao ambiente de alocao LISP	293a	06/11/2001
590.	Processamento fonetico (3 algs)	790a	26/11/2008	685.	Simulacao: banco queromeu + sist. op.	772a	19/09/2005
591.	Processamento simbolico: sympy/Maple	938a	01/06/2019	686.	Simulacao: porta-esper e questoes prov	782a	01/11/2011
592.	Prog. dinamica: plano de producao	672a	06/10/2015	687.	Simulador de FAT no dos/windows	291a	04/05/2000
593.	Programacao I	136b	08/07/2024	688.	Simulador de alocao em UNIX	292a	04/05/2000
594.	Programacao II	136c	08/07/2024	689.	Sistemas de numeracao	k04a	02/01/2013
595.	Programacao III	136d	09/07/2024	690.	Sistemas de producao	815a	03/02/2005
596.	Programacao IV	136e	09/07/2024	691.	Sistemas especialistas	819a	28/03/2002
597.	Programacao V	136f	09/07/2024	692.	Sistemas lineares-Jacobi (freemat)	k28f	01/09/2016
598.	Programacao VI	136g	09/07/2024	693.	Sistemas lineares-Jacobi (python)	k28p	05/04/2018
599.	Programacao dinamica	265a	17/08/2014	694.	Sistemas operacionais	q30b	06/05/2024
600.	Programacao dinamica, somas e k-vizinh	666d	27/07/2017	695.	Solicita o codigo de 9 funcoes	111a	26/07/2008
601.	Programacao genetica	945a	02/06/2005	696.	Solucao do cubo RUBIK	820a	03/01/2005
602.	Programacao linear: metodo simplex	263a	16/10/2018	697.	Sort topologico em grafos	633a	22/10/2001
603.	Programacao literata e 3 f. recurs.	258a	29/11/2008	698.	Sorts: primeira parte	511a	03/01/2003
604.	Programacao via arquivos	055a	13/11/2018	699.	Sorts: segunda parte	512a	04/01/2003
605.	Programar e...	001a	11/10/2020	700.	Strings em Python	931a	19/07/2018
606.	Programas C/C++ com parametros	071c	20/05/2014	701.	Sudoku para computadores	j39a	03/06/2024
607.	Projeto de uma rede neural	848a	08/05/2006	702.	TSP de pequeno porte: 6 cidades	259b	06/02/2014
608.	Prolog 1 - exercicios de Silvio Lago	861a	06/09/2012	703.	TSP em Python	259e	25/05/2017
609.	Prolog 2 - exercicios de Silvio Lago	862a	06/09/2012	704.	TSP em pseudo-codigo	259h	03/02/2010
610.	Prolog 3 - exercicios de Silvio Lago	863a	07/09/2012	705.	Tabela hash, largura, dijsktra guloso	666c	27/07/2017
611.	Prolog 4 - exercicios de Silvio Lago	864a	07/09/2012	706.	Tabelas em HTML	h35a	29/07/2015
612.	Prolog 5 - exercicios de Silvio Lago	865a	07/09/2012	707.	Tabelas hash	q80a	06/08/2024
613.	Prolog 6 - exercicios de Silvio Lago	866a	07/09/2012	708.	Tabelas verdade	094a	24/01/2012
614.	Prolog: familia pirlimpimpim	860c	28/10/2011	709.	Tabelas verdade aleatorias	095e	27/01/2012
615.	Prolog: introducao e exercicios	860b	10/06/2006	710.	Talento de D. Mariquinhas	009d	02/03/2012
616.	Prolog: juncao de 860b+860c	860d	11/10/2013	711.	Taxas de variacao relacionadas	353a	17/03/2024
617.	Pseudocodigo e fluxograma equivalentes	018f	05/05/2007	712.	Tecnicas de compressao - parte 1	766a	14/07/2007
618.	Pseudocodigo e nassi equivalentes	018n	05/05/2007	713.	Tecnicas de compressao - parte 2	767a	17/07/2007
619.	QR Code	571a	31/01/2015	714.	Tensorflow	851a	25/07/2017
620.	Qual funcao faz o que - C++	044c	11/05/2024	715.	Teorema Fundamental da Aritmetica	h99a	20/09/2024
621.	Qual funcao faz o que - Python	044p	15/05/2024	716.	Teorema de Bayes	911a	11/01/2015
622.	Qual o numero com mais divisores ?	ha8a	04/10/2024	717.	Teoria de conjuntos e diagramas de Ven	101c	25/08/2014
623.	Quebra cab: A Terra recebe uma visita	z19a	31/10/2017	718.	Teoria de filas: SM e faminto de ouro	770a	18/09/2005
624.	Quebra cab: Blocos cientistas famosos	z11a	13/10/2012	719.	Teoria de filas: simulacao ARENA	775a	11/10/2005
625.	Quebra cab: Dissecacoes geometricas	z15a	14/09/2006	720.	Tetris	j72a	04/09/2024
626.	Quebra cab: Insanidade Instantanea	z10a	11/11/2000	721.	Tomografia usando sist. lineares	k37a	14/04/2018
627.	Quebra cab: Sudoku para humanos	z20a	05/09/2006	722.	Tr2: decomp. brinq. e pitagoras Python	184p	25/06/2019
628.	Quebra cab: Visao lateral	z25a	06/06/2007	723.	Tr3: decomp. perfeito e superm. C++	185c	14/06/2019
629.	Quebra cab: garotos e seus brinquedos	z02a	17/12/2011	724.	Tr3: decomp. perfeito e superm. Python	185p	25/06/2019
630.	Quebra cab: lutadores de jiu jitsu	z08a	21/04/2012	725.	Tr4: simetria, vulcao e fibonacci-C++	186c	14/06/2019
631.	Quebra cab: meninas estudiosas	z04a	14/12/2011	726.	Tr4: simetria, vulcao fibonacci-Python	186p	25/06/2019
632.	Quebra cab: mulheres no supermercado	z03a	17/12/2011	727.	Tr5: conjuntos, det e m.inv em C++	187c	15/06/2019
633.	Quebra cab: quem bebe agua e tem zebra	z05a	26/07/2004	728.	Tr5: conjuntos, det e m.inv em Python	187p	25/06/2019
634.	Quebra cab: reveillon das mocas	z07a	11/01/2012	729.	Tr6: centro, histogr e xadrez - C++	188c	16/06/2019
635.	Quebra cab: visita ao dentista	z09a	13/10/2012	730.	Tr6: centro, histogr e xadrez -Python	188p	27/06/2019
636.	Quebra cabeca 2048	j71a	03/09/2024	731.	Tr7: problemas da OBI	189c	09/07/2019

732.	Trans. fourier, lg paralelos, f. bloom	666e	27/07/2017	46.	18/02/2001	110v Operacoes com vetores e indirecoes	070
733.	Transformada de Fourier	953a	09/11/2020	47.	23/02/2001	332a 4 erros em listas encadeadas	065
734.	Transformada de Laplace	352a	14/03/2024	48.	13/03/2001	450a Codigos: ASCII, Ortelino e Trocaletra	071
735.	Transformadas BWT e MTF	455a	26/12/2008	49.	20/03/2001	942a Computacao evolutiva (fls iguais/difs)	072
736.	Treinamento em ABPs	437a	17/07/2021	50.	03/04/2001	712a Criptografia de chave publica - RSA	021
737.	Tries	436a	09/08/2003	51.	29/04/2001	341a Listas dupl.encadeadas em assembler	047
738.	Truco em assembler usando listas	331a	27/06/2000	52.	10/05/2001	710a Criptografia baseada no alg. DES	020
739.	Truques2: decomp. brinq e pitagoras-C+	184c	10/06/2019	53.	15/05/2001	272a Encadeamento em um teatro	073
740.	Truques: conv,container,poker: Python	183p	25/06/2019	54.	06/07/2001	763d Petri nets: transacoes em uma ATM	036
741.	Truques: convol., container e poker C+	183c	09/06/2019	55.	10/08/2001	444a Indices no padrao XBASE	074
742.	Tuplas, dicionarios conjuntos em Pytho	934a	24/07/2018	56.	22/10/2001	633a Sort topologico em grafos	075
743.	Tutorial CSS - 1	h50a	27/12/2014	57.	26/10/2001	271a Alg. guloso/best/first/next/worst	076
744.	Tutorial CSS - 2	h52a	28/12/2014	58.	06/11/2001	293a Simulacao ambiente de alocao LISP	078
745.	Tutorial CSS - 3	h54a	29/12/2014	59.	28/11/2001	027a Bases maiores do que 10	077
746.	Tutorial HTML - 1	h20a	22/12/2014	60.	17/01/2002	780a Numeros pseudo-randomicos	079
747.	Tutorial HTML - 2	h22a	22/12/2014	61.	18/01/2002	634a Kruskal: arv de espalhamento minima	080
748.	Tutorial HTML - 3	h24a	23/12/2014	62.	20/01/2002	701a Correcao de erros - alg. Hadamard	081
749.	Tutorial HTML - 4	h32a	24/12/2014	63.	24/01/2002	531a Busca: Boyer+Moore simplificado	082
750.	Tutorial HTML - 5	h34a	25/12/2014	64.	27/01/2002	824a Minimax	083
751.	Tutorial HTML - 6	h36a	26/12/2014	65.	04/02/2002	255a Arvore QUAD	084
752.	Tutorial HTML - 7	h38a	26/12/2014	66.	18/02/2002	273a Arvore genealogica	085
753.	Tutorial HTML - 8	h40a	26/12/2014	67.	22/02/2002	461a Heap	018
754.	Tutorial Javascript - 1	h60a	29/12/2014	68.	25/02/2002	031a Manipulação de datas	088
755.	Tutorial Javascript - 2	h63a	30/12/2014	69.	28/03/2002	819a Sistemas especialistas	086
756.	Tutorial Javascript - 3	h66a	31/12/2014	70.	03/04/2002	713a Esteganografia: msg dentro de BMP	038
757.	Tutorial Javascript - 4	h69a	01/01/2015	71.	18/06/2002	432a ABPs: busca, criacao e altura	089
758.	Tutorial Javascript - 5	h72a	01/01/2015	72.	01/07/2002	412a Conversao de arvore n-aria em binaria	090
759.	UVa 103 - Encaixando caixas	hb4a	11/10/2024	73.	02/07/2002	441a Aquecimento em arvores B	091
760.	UVa 105 - Perfil da skyline	hb1a	14/10/2024	74.	14/07/2002	611a Diametro de grafo com listas adjacente	092
761.	UVa 108 - Maior subsoma em matriz	hb2a	16/10/2024	75.	15/07/2002	632a Fluxos ford-fulkerson/edmonds-karp	093
762.	Um ambiente estimulante: Bitcoin	039a	10/05/2024	76.	17/07/2002	231a Complexidade: o problema da particao	094
763.	Um computador simples	q06a	05/05/2024	77.	27/12/2002	035b Localizacao de oleoduto	095
764.	Um jogo	j70a	23/08/2024	78.	02/01/2003	513a Redes de comparacao	096
765.	Um jogo de Pong usando app inventor	j12a	18/05/2024	79.	03/01/2003	511a Sorts: primeira parte	097
766.	Um jogo simples app Inventor: PALITO	j10a	13/05/2024	80.	04/01/2003	512a Sorts: segunda parte	098
767.	Um problema de logistica	z55a	16/07/2018	81.	06/01/2003	215a Logaritmos	099
768.	Um sistema	d70a	23/08/2024	82.	08/01/2003	715a Maquina ENIGMA	100
769.	Um site	h77a	23/08/2024	83.	09/01/2003	716a Quebra de RSA (alg. rho pollard)	101
770.	Uma RNA (rede neural) de 8e e 4s	841a	10/10/1996	84.	13/01/2003	315a Calculo da envoltoria convexa	102
771.	Unicode e UTF-8	480a	25/10/2014	85.	15/01/2003	809a Definicoes da Inteligencia Artificial	103
772.	Uso de DIV em CSS	h55a	01/05/2024	86.	30/01/2003	802a Maquina de Turing	104
773.	Uso de biblioteca padrao: bootstrap	h56a	12/05/2024	87.	06/02/2003	781a Listas skip	105
774.	Varejao em JS	h86a	31/07/2024	88.	05/03/2003	814a Mundo dos aspiradores de po	106
775.	Working path em algoritmos	017t	23/01/2014	89.	02/05/2003	312a Pilhas em assembler	107
776.	app inventor: onde deixei meu carro ?	j42a	14/07/2024	90.	03/05/2003	323a Filas em assembler	108
777.	app inventor: xilofone	j45a	14/05/2024	91.	08/06/2003	877a O que faz esta funcao LISP	109

3 Índice por data de liberação

1.	01/01/1993	451a Huffman - arvores iguais	001	100.	20/10/2003	731a LZW: compressao e descompressao	034
2.	01/01/1994	452a Huffman - arvores diferentes	002	101.	25/10/2003	943a Computacao evol.: operadores ad hoc	117
3.	10/10/1996	841a Uma RNA (rede neural) de 8e e 4s	050	102.	16/02/2004	453a Huffman arvore dinamica	118
4.	20/02/1997	623a Caminhamento n:n (floyd-warshall)	015	103.	21/05/2004	718a Criptografia El Gamal	119
5.	30/03/1997	624a Grafo: matriz de roteamentos	016	104.	19/07/2004	975a Processamento em linguagem natural	120
6.	10/10/1997	612a Caminhamento em lar e prof em grafos	012	105.	21/07/2004	857a Demonstracao automatica de teoremas	121
7.	05/05/1998	622a Caminhamento 1:1 (tennenbaum)	014	106.	26/07/2004	z05a Quebra cab: quem bebe agua e tem zebra	122
8.	11/10/1998	621a Caminho minimo em grafo (dijkstra)	013	107.	27/07/2004	856a Frases logicas	123
9.	03/03/1999	752a Imagem BMP: descobrir linhas cinza	030	108.	30/07/2004	950a Exercicio genesereth e nilsson	124
10.	03/04/1999	732a Fractal de Mandelbroot	029	109.	28/08/2004	878a Escreva funcao LISP que...	125
11.	20/04/1999	753a Imagem BMP: descobrir linhas coloridas	044	110.	26/09/2004	442a B arvore com t=3	027
12.	11/09/1999	521a Hashing	019	111.	01/10/2004	445a B-arvore com t=6	028
13.	21/02/2000	077a Chines de 4 algoritmos	004	112.	01/11/2004	821a Busca cega: 5 exercicios	126
14.	29/02/2000	069a Chines de 4 algoritmos aleatorios	005	113.	22/11/2004	443a Indices em SYBASE (arvores)	055
15.	02/03/2000	084a Balance Line	022	114.	03/01/2005	820a Solucao do cubo RUBIK	127
16.	11/03/2000	431a Arvore: criacao de ABP	010	115.	02/02/2005	925b Imagem: alg de Horowitz e Pavlidis	128
17.	30/03/2000	754a Concolucoes com mascaras em imagens	035	116.	03/02/2005	815a Sistemas de producao	129
18.	18/04/2000	251a Recursividade: funcoes A, B e C	006	117.	10/02/2005	891a Representacao do conhecimento	130
19.	18/04/2000	252a Recursividade: anagramas	023	118.	17/04/2005	922a Estereogramas	131
20.	18/04/2000	253a Recursividade: passeio do cavalo	024	119.	24/05/2005	817a Pratica de EXSINTA	132
21.	02/05/2000	311a Exemplos de pilhas	041	120.	02/06/2005	945a Programacao genetica	133
22.	02/05/2000	313a RPN-notacao polonesa reversa	009	121.	30/06/2005	879a Escrevendo software em LISP	134
23.	04/05/2000	291a Simulador de FAT no dos/windows	007	122.	15/07/2005	926a Esqueletizacao de imagens	135
24.	04/05/2000	292a Simulador de alocao em UNIX	008	123.	16/07/2005	800a Artigo e consideracoes de Turing	136
25.	05/05/2000	763b Petri nets: descobrir caracateristicas	031	124.	19/07/2005	825a Corte alfa-beta do minimax	137
26.	10/05/2000	755a Equalizacao de histograma em imagens	037	125.	04/08/2005	870a Basico de LISP	138
27.	19/05/2000	314a Pilhas: laberinto do ratinho	039	126.	18/09/2005	770a Teoria de filas: SM e faminto de ouro	139
28.	30/05/2000	324a Ordenacao topologica usando filas	045	127.	19/09/2005	772a Simulacao: banco quomeu + sist. op.	033
29.	12/06/2000	321a Filas: implementacao e exemplos	040	128.	11/10/2005	775a Teoria de filas: simulacao ARENA	140
30.	12/06/2000	322a Filas circulares: impl. e exemplos	043	129.	29/10/2005	636a Caminho hamiltoniano: domino	141
31.	27/06/2000	331a Truco em assembler usando listas	042	130.	23/11/2005	805a Questoes sobre o artigo de Turing	142
32.	14/07/2000	421a Arvores: redesenhar balanceando	048	131.	18/12/2005	318a Compilador de pseudo codigo	143
33.	01/08/2000	333a Livros em biblioteca: exemplo de lista	026	132.	20/12/2005	004a Introducao a algoritmos	144
34.	01/08/2000	342a Listas duplamente encadeadas	017	133.	30/12/2005	057a Continuar sequencias	145
35.	07/08/2000	411a Arvores: caminhamentos em 3 arvores	046	134.	02/01/2006	007a Descricao do GPS e simulacao	147
36.	07/08/2000	433a Arvores: inclusoes/exclusoes em ABPs	054	135.	28/01/2006	017b SEs compostos e encadeados	148
37.	28/08/2000	756a Padrao GIF: buscar coisas em arquivos	056	136.	31/01/2006	062a Engenharia reversa de algoritmos	149
38.	05/10/2000	763c Petri nets: transacoes em um somador	032	137.	02/02/2006	319a Engenharia reversa de compiladores	150
39.	14/10/2000	714a Assinatura de arquivos	059	138.	15/02/2006	640a PERT-CPM	151
40.	03/11/2000	750a Reconstituicao de imagem com tomografi	061	139.	18/02/2006	799a Perdidos na lua (decisoes coletivas)	152
41.	11/11/2000	z10a Quebra cab: Insanidade Instantanea	062	140.	24/03/2006	835a Perguntas sobre xadrez	153
42.	18/01/2001	233a Complexidade: habilidades de bolei	063				
43.	21/01/2001	232a Complexidade: requisitos de automovel	064				
44.	04/02/2001	024b Digitos verificadores	069				
45.	18/02/2001	110c Operacoes com vetores e indirecoes	070				

141.	03/04/2006	021a Seguir chinses	154	236.	08/08/2009	m52a Marat: panquecas, tubos e arvores	249
142.	11/04/2006	256a Casamento estavel	155	237.	10/08/2009	m54a Marat: num feios, permutacoes e fat-fa	250
143.	20/04/2006	065a Funcoes SE, ENQUANTO e REPITA	156	238.	12/08/2009	m56a Marat: circunfer. interseccao e ciclo	251
144.	08/05/2006	848a Projeto de uma rede neural	157	239.	09/10/2009	627a Caminho minimo no metro de curitiba	252
145.	04/06/2006	073a Achar o decimo termo na sequencia	158	240.	01/11/2009	511b Juncao do 511 e 512: nao ha exercicio	253
146.	10/06/2006	860b Prolog: introducao e exercicios	159	241.	30/12/2009	082a Algoritmo de intercalacao	254
147.	23/06/2006	054a Kumon de algoritmos	160	242.	06/01/2010	902a Geometria analitica	255
148.	07/08/2006	116b Pesquisa em vetores - parte 1	161	243.	15/01/2010	086a Algoritmo de quebra em relatorios	256
149.	10/08/2006	119a Manuseio de tabelas	162	244.	03/02/2010	259h TSP em pseudo-codigo	257
150.	22/08/2006	128a Matrizas	163	245.	17/05/2010	343a SGBDs antigos: inc/excl em LDEs	258
151.	29/08/2006	008a Problemas matematicos	164	246.	18/05/2010	a01a Questionario: sistemas distribuidos	259
152.	01/09/2006	011b Matematica e logica basicas	165	247.	26/05/2010	b02a Questionario: empresa Google	260
153.	05/09/2006	z20a Quebra cab: Sudoku para humanos	166	248.	09/06/2010	a05a Questionario: modelos de S.I.	261
154.	08/09/2006	257a Recursividade: sudoku para computadores	167	249.	20/08/2010	711a Criptografia - algoritmo TEA	263
155.	14/09/2006	z15a Quebra cab: Dissecacoes geometricas	168	250.	26/08/2010	768a Compressao (obsoleta 766 e 767)	264
156.	16/09/2006	144a Numeracao romana	169	251.	01/09/2010	705a Certificacao digital pratica	265
157.	21/09/2006	135a Manipulacoes no cubo Rubik	170	252.	13/09/2010	709a Cifra de Viginere	266
158.	24/09/2006	m04a Marat: Fazenda de ostras	171	253.	22/10/2010	b30a Pratica em MER: restaurante	267
159.	07/10/2006	m06a Marat: Univ. Pinguinho e cubos color.	172	254.	25/11/2010	245a Minimos quadrados	268
160.	28/10/2006	m08a Marat: magico e azulejamento piscina	173	255.	14/12/2010	645a Onde por o hipermercado ?	270
161.	17/01/2007	826b A*: tarkin e jarras	174	256.	14/12/2010	719a Esteganografia linear em BMP	269
162.	01/05/2007	650a Algoritmo húngaro de assinalamento	175	257.	29/12/2010	r49a Cartela de Latex	271
163.	05/05/2007	018f Pseudocodigo e fluxograma equivalentes	176	258.	08/02/2011	015b Comandos SE simples	272
164.	05/05/2007	018n Pseudocodigo e nassi equivalentes	177	259.	10/02/2011	015d Comandos SE (apenas codigo)	273
165.	06/06/2007	z25a Quebra cab: Visao lateral	178	260.	11/02/2011	040a Comandos de repeticao simples	274
166.	11/07/2007	125b Certificar 5 algoritmos basicos	179	261.	16/02/2011	998b Mnemonico para numeros irracionais	275
167.	11/07/2007	125c Certificar 5 algoritmos de vetor	180	262.	20/02/2011	998c Algoritmo guloso: o problema do troco	276
168.	12/07/2007	125d Certificar 5 algoritmos vetor + difs	181	263.	19/04/2011	047a Mais repeticao	277
169.	14/07/2007	766a Tecnicas de compressao - parte 1	182	264.	04/05/2011	r26a Cartela de C	278
170.	17/07/2007	767a Tecnicas de compressao - parte 2	183	265.	09/05/2011	m02a Maratona UP2011	279
171.	17/07/2007	947a Algoritmos geneticos: uso do TUDEL	184	266.	11/05/2011	r28a Cartela de APL	280
172.	27/07/2007	125e Certificar 5 algoritmos parte 4	185	267.	20/05/2011	116c Pesquisa em vetores - parte 2	281
173.	03/08/2007	m10a Marat: cientistas e iluminacao	186	268.	13/06/2011	132a Ordenacao simples 0(n**2)	282
174.	07/08/2007	660a Algoritmo do torneio de tenis	187	269.	11/07/2011	456a Caso pratico de Huffman ad-hoc	283
175.	11/08/2007	m14a Marat: caminhos e vulcao	188	270.	13/07/2011	036b Aritmetica decimal, binaria e hexad	284
176.	23/10/2007	006a Jogo da vida	189	271.	16/07/2011	903a Einstein e a teoria da relatividade	285
177.	09/11/2007	m12a Marat: jogo do tabuleiro (IOI-2005)	190	272.	23/07/2011	708a Cifra de Playfair	286
178.	12/11/2007	003a Algoritmo da raiz quadrada	191	273.	08/08/2011	d05a Questionario: introducao a BD	287
179.	12/12/2007	009b Achar o n-esimo em 30 sequencias	192	274.	14/08/2011	d10a Conceitos do modelo E-R	288
180.	17/12/2007	190a 10 problemas (papel-lapis) OBI2003	193	275.	23/08/2011	b20a 3 MER feitos e pede despacho de viat.	289
181.	18/12/2007	191b 5 problemas (lapis-papel) OBI2004	194	276.	23/08/2011	b60a Criacao de um BD no ACCESS	290
182.	18/12/2007	191k 5 problemas (lapis-papel) OBI2004	195	277.	26/08/2011	d14a Ferramentas CASE	291
183.	19/12/2007	192a 10 problemas (lapis-papel) OBI2005	196	278.	29/08/2011	b40a Pratica em MER: escolas do parana	295
184.	22/12/2007	193b 7 problemas (lapis-papel) OBI2006	197	279.	29/08/2011	b45a Engenharia reversa: construir o MER	292
185.	23/12/2007	193c 6 problemas (lapis-papel) OBI2006	198	280.	29/08/2011	b50a Pratica em MER: sist passagens aereas	293
186.	23/12/2007	193d 6 problemas (lapis-papel) OBI2006	199	281.	29/08/2011	b55a Pratica em MER: locadora de veiculos	294
187.	23/12/2007	193e 7 problemas (lapis-papel) OBI2006	200	282.	08/09/2011	b27a Normalizacao em MER	296
188.	31/12/2007	m16a Marat: genoma e palavras cruzadas	201	283.	15/09/2011	b63a Pratica em ACCESS	297
189.	03/01/2008	m18a Marat: trem, robot e restaurante	202	284.	15/10/2011	998d Forca bruta: critoaritmetica	302
190.	04/01/2008	m20a Marat: seq. H, carga e rede otica	203	285.	18/10/2011	d66a MySQL: resumo e exercicio	303
191.	04/01/2008	m22a Marat: Quermesse, notas e saldo	204	286.	28/10/2011	860c Prolog: familia pirlimpimp	304
192.	08/01/2008	m24a Marat: macaco e visita ao MASP	205	287.	29/10/2011	b59a Pratica em MER: Ultima tarefa	305
193.	08/01/2008	m26a Marat: aneis, balaies e meteoros	206	288.	01/11/2011	782a Simulacao: porta-esper e questoes prov	306
194.	09/01/2008	m28a Marat: domino, sorvete e piramide	207	289.	10/11/2011	965a Biorritmos	307
195.	17/01/2008	m30a Marat: tesouro, aeroporto e pedagogio	208	290.	14/12/2011	z04a Quebra cab: meninas estudiosas	308
196.	18/01/2008	m32a Marat: bicicleta, supermer. e erdos	209	291.	17/12/2011	z02a Quebra cab: garotos e seus brinquedos	310
197.	22/01/2008	m34a Marat: furos, senhas e orkut	210	292.	17/12/2011	z03a Quebra cab: mulheres no supermercado	309
198.	22/01/2008	m36a Marat: minhocas, furos e taxis	211	293.	21/12/2011	845a Algoritmo de retropropagacao	311
199.	23/01/2008	m38a Marat: colheita, linhas e desculpas	212	294.	23/12/2011	846a Numero de neuronios em RNA	312
200.	23/01/2008	m40a Marat: poker, margarida e truco	213	295.	11/01/2012	z07a Quebra cab: reveillon das mocas	313
201.	11/04/2008	r25a Pratica de VisualG	214	296.	21/01/2012	093a Logica proposicional	314
202.	07/05/2008	618a Desenho de grafos (graphviz)	215	297.	24/01/2012	094a Tabelas verdade	315
203.	05/06/2008	535a Casamento aproximado de cadeias	216	298.	27/01/2012	095e Tabelas verdade aleatorias	316
204.	18/06/2008	m42a Marat: 3n+1, viagem e editor grafico	217	299.	18/02/2012	p05a Maple: introducao	317
205.	22/06/2008	m44a Marat: interpretador, palindr. fibonac	218	300.	21/02/2012	p25a Maple: algebra basica	318
206.	22/06/2008	m58a Marat: milharie e vaiqueemole	219	301.	02/03/2012	009d Talento de D. Mariquinhas	319
207.	26/06/2008	064a 3 Estruturas de repeticao	220	302.	17/03/2012	826j A*: jarras com implementacao	321
208.	17/07/2008	434a Inc/Exc em 3 arvores binarias de pesq.	221	303.	17/03/2012	826t A*: tarkin com implementacao	320
209.	23/07/2008	131a Conceito introdutorio de arvore	222	304.	21/03/2012	096a Semantica em logica proposicional	322
210.	26/07/2008	111a Solicita o codigo de 9 funcoes	223	305.	24/03/2012	097a Equivalencia e reducao ao absurdo	323
211.	29/07/2008	114a Define vetor, registro e matriz	224	306.	01/04/2012	022a Diversos SEs	344
212.	18/11/2008	010a Geografia terrestre: dist. e cap. Grant	225	307.	01/04/2012	r29a Cartela de xBase	345
213.	26/11/2008	790a Processamento fonetico (3 algs)	226	308.	01/04/2012	r30a Cartela de Prolog	346
214.	29/11/2008	258a Programacao literata e 3 f. recurs.	227	309.	12/04/2012	z66a Menor caminho usando estradas e pontes	324
215.	19/12/2008	720a Aplicativos comerciais: contabilidade	228	310.	14/04/2012	p45a Maple: programacao	325
216.	20/12/2008	721a Aplicativos comerciais: folha de pag.	229	311.	17/04/2012	020a Comandos SE com complexidade crescente	326
217.	26/12/2008	455a Transformadas BWT e MTF	230	312.	21/04/2012	098f Algebra das proposicoes - Parte 2	327
218.	30/12/2008	723a Aplicativos comerciais: estoques	231	313.	21/04/2012	z08a Quebra cab: lutadores de jiu jitsu	328
219.	17/01/2009	656a Algoritmo Gale-Shapley de alocao	232	314.	05/05/2012	099b Argumentos e validade	329
220.	27/01/2009	240a Problema da mochila 0-1 (knapsack)	233	315.	09/05/2012	p55a Maple: mais programacao	330
221.	31/01/2009	012a Problemas envolvendo inteiros	234	316.	12/05/2012	099h Argumentos e validade ad-hoc	331
222.	05/02/2009	013a Mais problemas envolvendo inteiros	235	317.	14/05/2012	m60b Maratona UP 2012 p1	332
223.	12/03/2009	015j 27 Comandos condicionais simples	236	318.	17/05/2012	m60c Maratona UP 2012 p2	333
224.	13/03/2009	016b 15 comandos condicionais	237	319.	19/05/2012	100a Demonstracoes de argumentos	334
225.	23/03/2009	980a Guia de apreciacao do filme COLOSSUS	238	320.	05/06/2012	p65a Maple: exercicios de logica	335
226.	26/03/2009	041a 21 comandos de repeticao	239	321.	21/06/2012	457a Huffman sobre uma imagem BMP	336
227.	27/03/2009	042a 14 comandos de repeticao	240	322.	18/07/2012	101b Logica e diagramas de Venn	337
228.	06/05/2009	019a 7 algoritmos para seguir (chines)	241	323.	19/07/2012	102a Logica e quantificadores	338
229.	01/07/2009	014a Mais problemas de inteiros	242	324.	19/07/2012	103a Demonstracao automatica de teoremas	339
230.	11/07/2009	900a Operacoes sobre conjuntos	243	325.	20/07/2012	p15a Maple: matematica basica	340
231.	14/07/2009	901a Divisibilidade e numeros primos	244	326.	21/07/2012	p75a Maple: graficos	341
232.	25/07/2009	798a Roteiro para apresentacao da pos grad.	245	327.	10/08/2012	104a Dem. autom. teo. e PROLOG (clone 858)	342
233.	03/08/2009	m46a Marat: blocos, lixo e caixas	246	328.	31/08/2012	628a Rotas e tempos minimos	343
234.	05/08/2009	m48a Marat: fermat, cat-in-hat, subretang	247	329.	06/09/2012	861a Prolog 1 - exercicios de Silvio Lago	347
235.	07/08/2009	m50a Marat: arvores, criptografia e TSP	248	330.	06/09/2012	862a Prolog 2 - exercicios de Silvio Lago	348

331.	07/09/2012	863a	Prolog 3 - exercicios de Silvio Lago	349	426.	05/12/2014	016h	Comandos SEs aninhados	405
332.	07/09/2012	864a	Prolog 4 - exercicios de Silvio Lago	350	427.	19/12/2014	h26a	Meu primeiro site	444
333.	07/09/2012	865a	Prolog 5 - exercicios de Silvio Lago	351	428.	19/12/2014	h30a	Meu segundo site	445
334.	07/09/2012	866a	Prolog 6 - exercicios de Silvio Lago	352	429.	22/12/2014	h20a	Tutorial HTML - 1	446
335.	07/09/2012	r32a	Cartela de Lisp	353	430.	22/12/2014	h22a	Tutorial HTML - 2	447
336.	25/09/2012	p76a	Maple: leitura de arquivos	354	431.	23/12/2014	h24a	Tutorial HTML - 3	448
337.	29/09/2012	005a	Matematica Basica	355	432.	24/12/2014	h32a	Tutorial HTML - 4	449
338.	08/10/2012	023a	Enquanto e para basicos	356	433.	25/12/2014	h34a	Tutorial HTML - 5	449
339.	13/10/2012	z09a	Quebra cab: visita ao dentista	357	434.	26/12/2014	h36a	Tutorial HTML - 6	450
340.	13/10/2012	z11a	Quebra cab: Blocos cientistas famosos	358	435.	26/12/2014	h38a	Tutorial HTML - 7	450
341.	07/11/2012	037a	Base 64, funcionamento e conversoes	359	436.	26/12/2014	h40a	Tutorial HTML - 8	450
342.	08/11/2012	034a	Manipulacao de datas: vetor hoj-ama	360	437.	27/12/2014	h50a	Tutorial CSS - 1	451
343.	10/11/2012	002b	Funcoes inventadas + mat basica	361	438.	28/12/2014	h52a	Tutorial CSS - 2	452
344.	10/11/2012	002c	Funcoes inventadas II + mat basica	362	439.	29/12/2014	h54a	Tutorial CSS - 3	453
345.	11/11/2012	002d	Funcoes inventadas III + mat basica	363	440.	29/12/2014	h60a	Tutorial Javascript - 1	454
346.	12/11/2012	002e	Funcoes inventadas IV + mat basica	364	441.	30/12/2014	h63a	Tutorial Javascript - 2	455
347.	13/11/2012	002f	Funcoes inventadas V + mat basica	365	442.	31/12/2014	h66a	Tutorial Javascript - 3	456
348.	21/11/2012	z67a	Custo minimo usando 3 tipos de trens	366	443.	01/01/2015	h69a	Tutorial Javascript - 4	457
349.	26/11/2012	002g	Funcoes inventadas VI + mat basica	367	444.	01/01/2015	h72a	Tutorial Javascript - 5	458
350.	27/12/2012	105a	Logica e Teoria dos conjuntos	368	445.	03/01/2015	h29a	Pratica em textos: vamos escrever ?	460
351.	28/12/2012	038a	Sequencias de numeracao	369	446.	04/01/2015	095b	Construcao de tabela verdade	461
352.	02/01/2013	k04a	Sistemas de numeracao	370	447.	05/01/2015	024e	Boleto bancario da FEBRABAN	462
353.	15/01/2013	106a	Algebra de Boole	371	448.	11/01/2015	911a	Teorema de Bayes	463
354.	18/01/2013	108a	Logicas probabilistica e nebulosa	372	449.	27/01/2015	570a	Codigos Reed Solomon	464
355.	23/02/2013	029a	Condicionalis literais	373	450.	31/01/2015	571a	QR Code	465
356.	07/03/2013	810a	Diagrama PAGE para sistemas de IA	374	451.	01/02/2015	t36a	Mat: pratica em probabilidade: primos	466
357.	08/03/2013	r20a	Cartela de pseudo codigo	375	452.	03/02/2015	807a	Computadores auto-replicantes	467
358.	08/03/2013	r24a	Cartela de VisualG	376	453.	05/02/2015	t30a	Mat: analise combinatoria	468
359.	22/03/2013	e01a	Faca programa que... (e01)	377	454.	06/02/2015	t35a	Mat: probabilidade	469
360.	24/03/2013	e03a	Faca programa que... (e03)	378	455.	30/03/2015	120a	Pratica em conjuntos	470
361.	31/03/2013	e05a	Faca programa que... (e05)	379	456.	14/05/2015	p91a	Maple em 1 dia	471
362.	11/04/2013	e07a	Faca programa que... (e07)	380	457.	29/07/2015	h35a	Tabelas em HTML	472
363.	01/05/2013	098b	Algebra das proposicoes - Parte 1	381	458.	25/09/2015	671a	Multiplicacao de matrizes em serie	473
364.	11/05/2013	e09a	Faca programa que... (e09)	382	459.	06/10/2015	672a	Prog. dinamica: plano de producao	474
365.	30/05/2013	129a	Pratica em matrizes	383	460.	22/10/2015	h90a	Exame final de design: um site	475
366.	07/06/2013	036f	Fechamento de bases nao decimais	384	461.	03/12/2015	160a	Numeros primos	476
367.	18/06/2013	e11a	Faca programa que... (e11)	385	462.	05/12/2015	t82a	Mat: exercicios de teoria conjuntos	477
368.	24/06/2013	646a	Central 0800 com menores custos	386	463.	20/12/2015	r50a	Cartela de HTML	478
369.	28/06/2013	140a	Pratica em eixos cartesianos	387	464.	20/12/2015	r55a	Cartela de CSS	479
370.	13/07/2013	925f	Imagem: 925+928	388	465.	20/12/2015	r60a	Cartela de JavaScript	480
371.	15/07/2013	030a	Integridade de funcoes C	389	466.	20/12/2015	r70a	Cartela de Freemath/Octave	481
372.	07/08/2013	690c	Compressao JPG em C	390	467.	25/12/2015	d60a	Introducao ao modelo relacional	482
373.	07/08/2013	690p	Compressao JPG em Python	390	468.	15/01/2016	r80a	Cartela de MySQL	483
374.	09/08/2013	680a	Padrao PDF - construcao do PDF a mao	391	469.	21/01/2016	d62a	Normalizacao de relacoes	484
375.	11/10/2013	860d	Prolog: juncao de 860b+860c	392	470.	24/01/2016	d64a	Restricoes de integridade em BD	485
376.	16/11/2013	u04a	Compil: Compilador aritmetico	393	471.	06/02/2016	d71b	Consultas MySQL e respostas: casar	486
377.	19/11/2013	u08a	Compil: Compilador de se...fimse	394	472.	06/02/2016	d71c	Consultas MySQL: quais as certas	487
378.	20/11/2013	u10a	Compil: Compilador de arquivo PDF	395	473.	08/02/2016	d32a	Engenharia reversa	490
379.	28/11/2013	e99a	Auto aprendizado em programacao	396	474.	08/02/2016	d80a	DER: organizacao de congressos	488
380.	28/11/2013	u12a	Compil: compilando expressoes logicas	397	475.	08/02/2016	d81a	DER: sistema de almoxarifado	489
381.	29/11/2013	u16a	Compil: compilando o jogo da vida	398	476.	27/02/2016	095f	Construcao de tabela verdade II	491
382.	29/11/2013	u20a	Compil: compilando comandos repeticao	399	477.	07/03/2016	d12a	MER: companhia aerea e revenda	492
383.	02/12/2013	u28a	Compil: tudo junto - U04+U08+U20	400	478.	09/03/2016	099j	Argumentos literais e validade	492
384.	03/12/2013	u06a	Compil: Compilador Latex	401	479.	21/03/2016	d30b	10 documentos para normalizar	493
385.	03/12/2013	u24a	Compil: Maple -> C	402	480.	21/03/2016	d30c	Mais 10 documentos para normalizar	494
386.	21/12/2013	u05a	Compil: Compilador com reais e id long	403	481.	29/03/2016	d63a	Integridade referencial em BD	495
387.	23/01/2014	017t	Working path em algoritmos	404	482.	12/04/2016	d75a	Construindo e consultando um DB MySQL	496
388.	06/02/2014	259b	TSP de pequeno porte: 6 cidades	406	483.	13/04/2016	d69b	Seguranca e direitos em MySQL	497
389.	15/02/2014	u02a	Compil: Notacao Polonesa Reversa	407	484.	30/04/2016	d76a	Estatistica de transporte intermunicip	498
390.	27/02/2014	830a	A*: trajeto de um robot	408	485.	19/05/2016	112a	Pratica em vetores (Freemat/Octave)	499
391.	18/04/2014	058a	Distribuicao de cadeiras no legislativ	409	486.	28/05/2016	d85a	Banco de dados e contabilidade	500
392.	29/04/2014	035c	Exponenciacao, diagonais e trelicas	410	487.	06/07/2016	121p	Matrizes em Python	501
393.	14/05/2014	u11a	Compil: Utilitario extrator de PDFs	411	488.	14/07/2016	d35b	Funcoes de string em MySQL	502
394.	20/05/2014	071c	Programas C/C++ com parametros	412	489.	15/07/2016	d35c	Funcoes de data e hora em MySQL	503
395.	24/06/2014	560a	Analise estilistica de textos	413	490.	29/07/2016	d87a	Banco de dados e controle de estoque	504
396.	17/07/2014	246a	Regressao multipla	414	491.	03/08/2016	d86a	Banco de dados e folha de pagamento	505
397.	12/08/2014	076b	Leitura de dados preexistentes em C	415	492.	10/08/2016	d90a	Backup e restore no MySQL	506
398.	17/08/2014	265a	Programacao dinamica	416	493.	15/08/2016	d92a	MySQL: stored routines	507
399.	18/08/2014	831a	A*: caminho mais barato	417	494.	18/08/2016	d89a	Engenhos no MySQL	508
400.	25/08/2014	101c	Teoria de conjuntos e diagramas de Ven	418	495.	21/08/2016	k13a	Calc num: solucao de equacoes	509
401.	27/08/2014	t10a	Mat: progressoes aritmeticas e geometr	419	496.	24/08/2016	d93a	MySQL: triggers	510
402.	29/08/2014	t16a	Mat: matrizes	420	497.	27/08/2016	735f	Calcnun - sist. lineares - Octave	511
403.	30/08/2014	u42a	Compil: clone do vivx719	421	498.	30/08/2016	d94a	MySQL: cursores e variaveis	512
404.	30/08/2014	u44a	Compil: compilador de imagens BMP	422	499.	01/09/2016	k28f	Sistemas lineares-Jacobi (freemat)	513
405.	31/08/2014	t19a	Mat: equacoes e sistemas lineares	423	500.	15/09/2016	k19a	Revisao de calculo numerico	514
406.	01/09/2014	t22a	Mat: determinantes	424	501.	19/09/2016	z26a	Problemas aritmeticos	515
407.	02/09/2014	t25a	Mat: coeficientes binomiais	425	502.	20/09/2016	d98a	MySQL: dados espaciais	517
408.	09/09/2014	t02a	Matematica: relacoes	426	503.	20/09/2016	d99a	MySQL e PHP	516
409.	10/09/2014	t04a	Matematica: funcoes e somatorios	426	504.	21/09/2016	d95a	MySQL: replicacao	518
410.	11/09/2014	t50a	Mat: estatistica descritiva	428	505.	21/09/2016	d97a	MySQL: otimizacao	519
411.	12/09/2014	t45a	Mat: variaveis aleatorias	430	506.	29/09/2016	k40f	Interpolacao polinomial - Octave	520
412.	12/09/2014	t70a	Mat: aquecimento geometria analitica	429	507.	11/10/2016	k43f	Int. polin. Met. Lagrange - Octave	521
413.	26/09/2014	871a	Definicoes de funcao em LISP	431	508.	11/10/2016	k43p	Int. polin. Met. Lagrange - Python	521
414.	26/09/2014	872a	Mais funcoes em LISP	432	509.	18/10/2016	d77a	Avaliacao de banco de dados	522
415.	30/09/2014	t60a	Mat: distribuicao binomial e normal	433	510.	18/10/2016	o00a	Matematica retirado da OBMEP	523
416.	03/10/2014	z35a	Citale espartano	434	511.	19/10/2016	o01b	Aquecimento em matematica - 1	524
417.	09/10/2014	536b	Bioinf. 1-maior subsequencia comum	435	512.	21/10/2016	o01c	Aquecimento em matematica - 2	525
418.	09/10/2014	536c	Bioinf. 2-custos e busca padrao	436	513.	26/10/2016	k64a	Integracao numerica	526
419.	25/10/2014	480a	Unicode e UTF-8	437	514.	31/10/2016	o01d	Aquecimento em matematica - 3	527
420.	29/10/2014	t80a	Mat: conicas	438	515.	02/11/2016	k61f	Ajuste de curvas - Octave	528
421.	11/11/2014	h28a	Erros em documentos HTML	439	516.	09/11/2016	o01e	Aquecimento em matematica - 4	529
422.	12/11/2014	h16a	Metodologia de avaliacao de sites	440	517.	13/11/2016	o01f	Aquecimento em matematica - 5	530
423.	13/11/2014	h07a	Interface via agentes virtuais	441	518.	31/12/2016	912a	Pratica em probabilidade	531
424.	21/11/2014	h08a	Interface de menus e icones	442	519.	13/02/2017	095h	Pratica em tabelas verdade	532
425.	02/12/2014	538a	Expressoes regulares	443	520.	27/02/2017	090a	Se...entao: proposicao X comando	533

521.	06/03/2017	r37a	Cartela de Python	534	616.	25/06/2019	185p	Tr3: decomp. perfeito e superm. Python	623
522.	31/03/2017	760a	Padrao FITS	535	617.	25/06/2019	186p	Tr4: simetria, vulcao fibonacci-Python	624
523.	05/05/2017	t07p	Mat: ciclos de somatorio em Python	536	618.	25/06/2019	187p	Tr5: conjuntos, det e m.inv em Python	625
524.	11/05/2017	t17a	Mat: pratica de matrizes em python	537	619.	27/06/2019	188p	Tr6: centro, histogr e xadrez -Python	626
525.	16/05/2017	051d	Pratica de programacao	538	620.	09/07/2019	189c	Tr7: problemas da OBI	627
526.	25/05/2017	259e	TSP em Python	539	621.	10/07/2019	053a	Pratica de programacao III	628
527.	26/05/2017	052a	Pratica em programacao II	540	622.	13/07/2019	056a	Introducao a programacao	629
528.	08/06/2017	162p	Otimizacao de algoritmos em Python	541	623.	01/08/2019	049p	Primeiros passos em programacao Python	630
529.	10/06/2017	t51a	Mat: pratica em estatistica descritiva	542	624.	03/08/2019	049q	Primeiros passos programacao Python-II	631
530.	10/07/2017	458a	Pratica em huffman: os lusiadas	543	625.	04/08/2019	049b	Primeiros passos em programacao C++	630
531.	25/07/2017	851a	Tensorflow	544	626.	04/08/2019	049d	Primeiros passos em programacao II-C++	631
532.	27/07/2017	666b	Complexidade: busca binaria e recursao	545	627.	16/08/2019	059a	Introducao a programacao - proj Euler	632
533.	27/07/2017	666c	Tabela hash, largura, dijkstra guloso	546	628.	17/09/2019	985a	Como escrever um artigo cientifico	633
534.	27/07/2017	666d	Programacao dinamica, somas e k-vizinh	547	629.	23/09/2019	z41a	Construcao de uma regua de calculo	634
535.	27/07/2017	666e	Trans. fourier, lg paralelos, f. bloom	548	630.	30/09/2019	266a	Selecao de atividades	635
536.	29/07/2017	668a	Algoritmo dos k-vizinhos	549	631.	08/10/2019	639a	Coloracao de grafos: aplicacoes	636
537.	29/08/2017	869a	Linguagens classicas da IA	550	632.	18/10/2019	935a	Objetos em Python	637
538.	31/08/2017	669a	Filtro de Bloom	551	633.	29/10/2019	847a	Rede neural: implementacao em Python	638
539.	13/09/2017	913a	Redes bayesianas de crenca	552	634.	02/11/2019	150a	Logica de programacao: ABP,m.q.,lombad	639
540.	28/09/2017	076c	Leitura de dados preexistentes Python	553	635.	06/12/2019	050c	Gerador de movimentos em xadrez-C++	606
541.	31/10/2017	z19a	Quebra cab: A Terra recebe uma visita	554	636.	06/12/2019	050p	Gerador de movimentos em xadrez-Python	605
542.	09/11/2017	n02a	Problema dos generais bizantinos	555	637.	13/01/2020	r51a	Cartela de J	640
543.	10/11/2017	n05a	Autenticacao de mensagens (toy domain)	556	638.	15/09/2020	t85a	Mat: aquecimento em calculo	641
544.	15/11/2017	n25a	Numeracao de base 58	557	639.	20/09/2020	t87a	Mat: taxas de variacao e derivadas	642
545.	16/11/2017	n20a	Numeros pseudo-randomicos	558	640.	25/09/2020	t89a	Mat: derivadas relacionadas	643
546.	18/11/2017	n60a	Mineracao e prova de trabalho (bitcoin)	559	641.	29/09/2020	t90a	Mat: maximos e minimos	644
547.	20/11/2017	n55a	Arvores de Merkle	560	642.	29/09/2020	t92a	Mat: funcoes trigonometricas	645
548.	20/11/2017	n65a	Filtro de Bloom	561	643.	30/09/2020	t94a	Mat: funcoes exponenciais	646
549.	28/11/2017	n10a	Cript. assimetrica de c. eliptica	562	644.	05/10/2020	t96a	Mat: problemas de area revolutiva	647
550.	02/12/2017	n40a	Scripts de transacao em bitcoin	563	645.	11/10/2020	001a	Programar e...	648
551.	04/12/2017	n75a	Bifurcacao e disputa de blockchain	564	646.	17/10/2020	260b	Arvores com recursividade e apontadore	649
552.	07/12/2017	n35a	Carteiras em bitcoin	565	647.	17/10/2020	260c	Arvores com recursividade e cursores	650
553.	11/01/2018	028b	Condicionalis aninhadas	566	648.	25/10/2020	952a	Imagem: transformada de Hough	651
554.	12/01/2018	028e	Condicionalis aninhadas	567	649.	09/11/2020	953a	Transformada de Fourier	652
555.	20/01/2018	028k	Condicionalis aninhadas e compostas II	568	650.	23/04/2021	954a	Espacos de cores	653
556.	22/01/2018	028n	Condicionalis aninhadas e compostas III	569	651.	22/05/2021	222a	Alto/baixo 1:conjuntos	655
557.	12/03/2018	011c	Matematica e logica basicas (form C)	570	652.	31/05/2021	220a	Complexidade, recursao e rel. recorr.	656
558.	15/03/2018	046b	Condicionalis em C++	571	653.	06/06/2021	223a	Alto/baixo 2:pilha e fila	657
559.	16/03/2018	046p	Condicionalis em Python	572	654.	07/06/2021	224a	Alto/baixo 3:ABP	658
560.	17/03/2018	046d	Condicionalis em C++ II	573	655.	08/06/2021	225a	Alto/baixo 4:assembly e p. alto nivel	659
561.	17/03/2018	046q	Condicionalis em Python II	574	656.	10/06/2021	226a	Alto/baixo 5:matriz inversa	660
562.	24/03/2018	735p	Calcnm - sist. lineares - Python	511	657.	28/06/2021	261a	Arvores k-d	661
563.	27/03/2018	048c	Repeticao em C++	512	658.	09/07/2021	175a	Perolas: complexos,pikachu e sierpinsk	662
564.	27/03/2018	048p	Repeticao em Python	513	659.	17/07/2021	437a	Treinamento em ABPs	654
565.	30/03/2018	k34a	Problemas praticos de sistemas lineare	514	660.	02/08/2021	t13a	Mat: demonstracoes algebricas	663
566.	05/04/2018	k28p	Sistemas lineares-Jacobi (python)	575	661.	09/08/2021	785a	O problema da parada (3 descricoes)	664
567.	14/04/2018	k37a	Tomografia usando sist. lineares	576	662.	02/11/2021	k31a	Gauss-Seidl e relaxacao	665
568.	16/04/2018	k40p	Interpolacao polinomial - Python	577	663.	13/11/2021	k46a	Int. polin. Met. Lagrange - Sympy	666
569.	30/04/2018	k61p	Ajuste de curvas - Python	528	664.	14/11/2021	k49a	Polinomio de Newton e met. Diferencas	667
570.	21/05/2018	r38a	Cartela de Matplotlib	579	665.	15/11/2021	k52a	Polinomio de Newton-Gregory	668
571.	22/05/2018	r39a	Cartela de Numpy	580	666.	06/12/2021	k01a	Introducao a metodos numericos	669
572.	23/05/2018	r41a	Cartela de Sympy	581	667.	13/12/2021	k07a	Representacao interna de numeros	670
573.	24/05/2018	k67a	Aplicacoes de integracao numerica	582	668.	16/12/2021	k16a	Iteracao linear, secante e misto	671
574.	28/05/2018	157b	Pratica matrizes em C++	583	669.	28/01/2022	k80a	Resumo pratico de calculo numerico	672
575.	28/05/2018	157c	Pratica matrizes em C++ - II	584	670.	12/02/2022	k08g	Armazenamento de flutuantes	672
576.	28/05/2018	157p	Pratica matrizes em Python	583	671.	12/02/2022	k90a	Jupyter e markdown com python	674
577.	28/05/2018	157q	Pratica matrizes em Python - II	584	672.	05/10/2022	080a	Introducao a Python	676
578.	30/05/2018	158c	Matriz inversa em C++	585	673.	21/10/2022	k08b	Repres. interna numeros maq. binarias	678
579.	30/05/2018	158p	Matriz inversa em Python	585	674.	12/01/2024	025a	Aritmetica complexa	673
580.	31/05/2018	162c	Otimizacao de algoritmos em C++	586	675.	20/01/2024	026a	Aritmetica inteira basica	675
581.	09/07/2018	k22a	Metodo de Gauss para sistemas lineares	596	676.	21/01/2024	045a	Calculo de PI	677
582.	09/07/2018	k25a	Matriz LU para sistemas lineares	587	677.	25/01/2024	066a	Geometrias não euclidianas	679
583.	16/07/2018	z55a	Um problema de logistica	588	678.	27/01/2024	061a	Grupos algébricos	680
584.	18/07/2018	930a	Listas em Python	589	679.	03/02/2024	063a	Equacoes e inequacoes	681
585.	19/07/2018	931a	Strings em Python	590	680.	08/02/2024	078a	Autovalores e autovetores	682
586.	20/07/2018	932a	Escreva uma funcao que... (em Python)	591	681.	10/02/2024	079a	Aplicacoes da Algebra Linear	683
587.	23/07/2018	933a	Arquivos em Python	592	682.	13/02/2024	075a	Desenhando funcoes	684
588.	24/07/2018	934a	Tuplas, dicionarios conjuntos em Pytho	593	683.	17/02/2024	089a	Otimizacao de funcoes	685
589.	26/07/2018	937a	Pratica em Matplotlib (pacote Python)	594	684.	24/02/2024	088a	Sequencias e series	686
590.	03/08/2018	051e	Pratica de programacao II	595	685.	14/03/2024	352a	Transformada de Laplace	687
591.	30/08/2018	159a	Matriz inversa - conceito	596	686.	17/03/2024	353a	Taxas de variacao relacionadas	688
592.	16/10/2018	262a	Recursividade: Strassen e quickselect	598	687.	24/03/2024	351a	Equacoes diferenciais	689
593.	16/10/2018	263a	Programacao linear: metodo simplex	597	688.	28/04/2024	h51a	Introducao a CSS	690
594.	13/11/2018	055a	Programacao via arquivos	599	689.	29/04/2024	h53a	Organizando o layout com CSS	691
595.	30/11/2018	145a	Completando algoritmos	600	690.	01/05/2024	h55a	Uso de DIV em CSS	692
596.	03/12/2018	146a	Compl. alg. II	601	691.	05/05/2024	q05b	Arquitetura de computadores I	692
597.	08/12/2018	147a	Compl. alg. III	602	692.	05/05/2024	q05c	Arquitetura de computadores II	693
598.	26/12/2018	148a	Compl. alg. IV - arvore Merkle	603	693.	05/05/2024	q06a	Um computador simples	694
599.	01/01/2019	149a	Compl. alg. V - sist lin e caixeiro	604	694.	06/05/2024	q20b	Redes de computadores	695
600.	21/02/2019	t07c	Mat: ciclos de somatorio em C++	607	695.	06/05/2024	q20c	Modelo de camadas da Internet	696
601.	06/03/2019	170c	Sao equivalentes as condicionais-C++	608	696.	06/05/2024	q30b	Sistemas operacionais	697
602.	06/03/2019	170p	Sao equivalentes as condicionais-Pytho	609	697.	08/05/2024	q10a	Objetos binarios grandes - BLOBs	698
603.	23/03/2019	051p	Pratica de programacao inicial Python	610	698.	10/05/2024	039a	Um ambiente estimulante: Bitcoin	699
604.	24/03/2019	051c	Pratica de programacao inicial C++	611	699.	11/05/2024	044c	Qual funcao faz o que - C++	700
605.	08/05/2019	k55a	Exercicios de interpolacao	612	700.	11/05/2024	044z	Escreva 10 programas C++	701
606.	01/06/2019	938a	Processamento simbolico: sympy/Maple	613	701.	12/05/2024	h56a	Uso de biblioteca padrao: bootstrap	702
607.	04/06/2019	936a	Pratica em Numpy (pacote Python)	614	702.	13/05/2024	j10a	Um jogo simples app Inventor: PALITO	703
608.	09/06/2019	183c	Truques: convol., container e poker C+	615	703.	14/05/2024	j45a	app inventor: xilofone	737
609.	10/06/2019	184c	Truques2: decomp. brinq e pitagoras-C+	616	704.	15/05/2024	044p	Qual funcao faz o que - Python	704
610.	14/06/2019	185c	Tr3: decomp. perfeito e superm. C++	617	705.	18/05/2024	043c	O que faz o codigo em C++	706
611.	14/06/2019	186c	Tr4: simetria, vulcao e fibonacci-C++	618	706.	18/05/2024	043e	O que faz o codigo em pseudocodigo	705
612.	15/06/2019	187c	Tr5: conjuntos, det e m.inv em C++	619	707.	18/05/2024	j12a	Um jogo de Pong usando app inventor	707
613.	16/06/2019	188c	Tr6: centro, histogr e xadrez - C++	620	708.	22/05/2024	h61a	JS como linguagem de desktop	708
614.	25/06/2019	183p	Truques: conv, container,poker: Python	621	709.	27/05/2024	067a	Seguir o codigo: repeticoes	709
615.	25/06/2019	184p	Tr2: decomp. brinq. e pitagoras Python	622	710.	30/05/2024	072b	Iniciando na programacao	710

711.	30/05/2024	072d Iniciando na programacao II	718
712.	01/06/2024	j21a Jogo de general	711
713.	03/06/2024	j39a Sudoku para computadores	712
714.	04/06/2024	j14a Mini golfe em app inventor	713
715.	05/06/2024	q11a Mais um BLOB: Tomografia	714
716.	10/06/2024	q66a Aprendizado profundo	715
717.	14/06/2024	072e Iniciando na programacao III	719
718.	14/06/2024	h81a Imagem GIF	717
719.	14/06/2024	q37a Algoritmo A* - aplicacoes	716
720.	15/06/2024	q32a Representacao de numeros em maquinas	720
721.	17/06/2024	h82a Espaço de cores programado em JS	721
722.	21/06/2024	072g Iniciando na programacao IV	722
723.	22/06/2024	q43a Alocacao sequencial e encadeada	723
724.	23/06/2024	555a Representacao estruturas bidimensionais	724
725.	30/06/2024	q91a Arvores	725
726.	03/07/2024	072j Iniciando na programacao V	727
727.	03/07/2024	q92a Arvores-B	726
728.	06/07/2024	072k Iniciando na programacao VI	728
729.	08/07/2024	136b Programacao I	729
730.	08/07/2024	136c Programacao II	730
731.	09/07/2024	136d Programacao III	731
732.	09/07/2024	136e Programacao IV	732
733.	09/07/2024	136f Programacao V	733
734.	09/07/2024	136g Programacao VI	734
735.	13/07/2024	h83a Caça niqueis HTML+CSS+JS	735
736.	14/07/2024	j42a app inventor: onde deixei meu carro ?	736
737.	15/07/2024	h84a Conversoes em HTML+CSS+JS	738
738.	18/07/2024	h85a Folha de Pagamento em JS	739
739.	31/07/2024	939a Blocos em Python	740
740.	31/07/2024	h86a Varejao em JS	741
741.	02/08/2024	h87a 3 Calculos elementares em JS	742
742.	06/08/2024	q80a Tabelas hash	743
743.	07/08/2024	h88a Funcoes inventadas apoiadas em JS	744
744.	09/08/2024	q81a Grafos e caminho minimo	745
745.	10/08/2024	q82a Compressao de dados	746
746.	11/08/2024	q83a Caixeiro viajante	747
747.	16/08/2024	h89a Operacoes com datas	748
748.	17/08/2024	h91a Maquina de troco	749
749.	18/08/2024	h92a Criptograma letras e numeros	750
750.	19/08/2024	h93a Leitura de constantes em JS	751
751.	21/08/2024	h94a Diagonais da matriz	752
752.	23/08/2024	d70a Um sistema	753
753.	23/08/2024	h77a Um site	754
754.	23/08/2024	j70a Um jogo	755
755.	02/09/2024	q86a Metrô de Curitiba	756
756.	03/09/2024	j71a Quebra cabeça 2048	757
757.	04/09/2024	j72a Tetris	758
758.	11/09/2024	h95j Rodada de poker em JS	759
759.	16/09/2024	h96a Formatos internos em JS	760
760.	16/09/2024	h97j Conjuntos em JS	761
761.	17/09/2024	h98a Mais programas em JS	762
762.	18/09/2024	q84a Criptografia	763
763.	20/09/2024	h99a Teorema Fundamental da Aritmetica	764
764.	20/09/2024	ha1a Sequencia de Collatz	765
765.	24/09/2024	ha2a Primos pitagoreanos	766
766.	25/09/2024	ha3a Calculo do determinante	767
767.	25/09/2024	ha4a Maior palavra	768
768.	26/09/2024	ha5a MMC e MDC em JS	769
769.	26/09/2024	ha6a Algoritmo Addition Chaining em JS	770
770.	26/09/2024	ha7a Numeros feios (UVA) em JS	771
771.	04/10/2024	ha8a Qual o numero com mais divisores ?	772
772.	05/10/2024	ha9a Fatorial em termos dos fatores primos	773
773.	14/10/2024	hb1a UVA 105 - Perfil da skyline	774
774.	16/10/2024	hb2a UVA 108 - Maior soma em matriz	775
775.	21/10/2024	hb4a UVA 103 - Encaixando caixas	776
776.	25/10/2024	h97p Conjuntos em Python	777
777.	26/10/2024	h95p Rodada de poker em Python	778

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais “nobre” da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a ideia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquinas são transferidas para um programa de com-

putador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- ★ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- ★ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte “arte”. O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}*. Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9:   TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com

o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas “coisas”. É como aprender um idioma novo. Mas antes de aprender a expressar ideias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 9
2: Y ← 36
3: ALFA ← 3
4: enquanto X < Y
5:   X ← X + ALFA
6:   se ((ALFA mod 3) ≠ 0)
7:     ALFA ← ALFA + 1
8:   fim{se}
9:   ALFA ← ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 2
2: B ← 22
3: enquanto A ≤ B
4:   se ((B > 7) ∨ (A ≤ 14))
5:     B ← B - 1
6:   fim{se}
7:   A ← A + 3
8:   B ← B - 2
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
1: X ← 3
2: enquanto X < 9
3:   Y ← 3
4:   enquanto Y < 15
5:     X ← X + 3
6:     Y ← X + 1
7:   fim{enquanto}
8:   Y ← Y + 3
9:   X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 9
2: Y ← 4
3: se X < 6 ∧ Y > 3
4:   X ← X - 3
5: senão
6:   Y ← Y + 2
7:   se X mod 4 ≠ 0
8:     Y ← Y + 2
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4



Expressões aritméticas

Nesta folha vão aparecer diversos conjuntos de 6 expressões aritméticas simples, envolvendo as operações de adição, subtração, multiplicação e divisão, todas conhecidas nossas.

Elas vão ser hierarquizadas por parênteses, que devem ser respeitados. A novidade é o surgimento de duas funções novas, chamadas teto ($\lceil$) e chão ($\lfloor$). Estas operações foram definidas pelo matemático canadense Kenneth Iverson, no final da década de 50. (Curiosamente, ele comentou que o símbolo foi obtido raspando um pedaço do caracter “abre colchete” da máquina de escrever).

O funcionamento dessas operações é simples e pode ser definido assim:

Chão O chão devolve o próximo inteiro igual ou menor ao operando da função.

Teto O teto devolve o próximo inteiro igual ou maior ao operando da função

Eis alguns exemplos:		
$\lfloor 3$	3	o chão de 3 é 3
$\lfloor 3.5$	3	o chão de 3.5 é 3
$\lfloor 1.9$	1	o chão de 1.9 é 1
$\lfloor 5.01$	5	o chão de 5.01 é 5
$\lfloor -2.5$	-3	o chão de -2.5 é -3
$\lceil 3$	3	o teto de 3 é 3
$\lceil 3.5$	4	o teto de 3.5 é 4
$\lceil 1.9$	2	o teto de 1.9 é 2
$\lceil 5.01$	6	o teto de 5.01 é 6
$\lceil -2.5$	-2	o teto de -2.5 é -2

Uma questão importante envolvendo essas duas operações é que elas transformam um número real (eventualmente com casas decimais) em um número inteiro.

Seja agora um exemplo com 6 operações aritméticas:

$A \leftarrow 9$
 $B \leftarrow 2$
 $C \leftarrow 8$
 $D \leftarrow 5$
 $F \leftarrow 2$
 $(6 \times F) - ((2 - F) \times (F + F))$
 $((\lceil(4 \div F)\rceil) \times (3 + F)) - (\lceil(6 \div F)\rceil)$
 $((F - F) - (5 - 6)) - (F + F)$
 $((5 - F) \times (D \times F)) - (7 + 4)$
 $((\lfloor(C \div 2)\rfloor) \times (\lceil(A \div 7)\rceil)) \times (9 - F)$
 $(C - 4) - ((5 + 6) - (6 \times 5))$

Resolvendo as 6 expressões, obtem-se os seguintes 6 valores: 12, 7, -3, 19, 56 e 23.

Mais um exemplo a fazer:

$A \leftarrow 5$
 $B \leftarrow 5$
 $C \leftarrow 3$
 $D \leftarrow 4$
 $F \leftarrow 5$
 $(5 \times F) + ((A - F) - (\lceil F \div 7 \rceil))$
 $\lfloor(((8 - F) \times (D \times 9)) \div (B + F))\rfloor$
 $(\lfloor(A \div 8)\rfloor) - ((9 \times 6) - (F - 6))$
 $((F - 6) - (\lceil(C \div 6)\rceil)) \times (\lfloor(D \div F)\rfloor)$
 $\lceil(((B - 4) \times (3 + F)) \div (8 + F))\rceil$
 $((A + 8) - (5 - F)) - (5 + 4)$

Cujas respostas vão ser:

Para você fazer

1. Resolva as 6 linhas a seguir

$A \leftarrow 5$
 $B \leftarrow 3$
 $C \leftarrow 4$
 $D \leftarrow 4$
 $F \leftarrow 9$
 $(\lfloor(2 \div F)\rfloor) + ((6 \times 5) - (7 \times F))$
 $(C + 7) \times ((\lceil(A \div 8)\rceil) - (\lfloor(A \div F)\rfloor))$
 $(6 - 3) - ((3 \times 2) - (\lceil(6 \div F)\rceil))$
 $((7 \times F) - (D \times 5)) + (8 - 8)$
 $(5 + F) \times ((B \times 6) \times (B \times 2))$
 $(B \times 2) + ((A \times F) \times (A - F))$

E informe abaixo, quantas linhas retornaram valores ímpares.

2. Resolva as 6 linhas a seguir

$A \leftarrow 3$
 $B \leftarrow 4$
 $C \leftarrow 9$
 $D \leftarrow 2$
 $F \leftarrow 6$
 $(\lceil(6 \div F)\rceil) - ((\lfloor(3 \div F)\rfloor) + (6 - 9))$
 $((F \times 6) \times (6 + 4)) - (6 - 2)$
 $(6 + 9) + ((C \times 2) - (\lceil(C \div 6)\rceil))$
 $(A \times F) - ((7 \times F) \times (\lceil(D \div F)\rceil))$
 $((F \times F) \times (A + 9)) + (9 + 7)$
 $((\lceil(C \div 5)\rceil) - (5 \times 3)) \times (9 \times F)$

E informe abaixo, quantas linhas retornaram valores maiores que 50.

3. Resolva as 6 linhas a seguir

$A \leftarrow 3$
 $B \leftarrow 4$
 $C \leftarrow 8$
 $D \leftarrow 4$
 $F \leftarrow 9$
 $(\lceil(B \div 3)\rceil) + ((2 \times 4) + (\lceil(B \div 9)\rceil))$
 $(2 - F) - ((5 - 2) \times (8 + 7))$
 $(F + F) - ((\lfloor(9 \div F)\rfloor) + (C + 4))$
 $\lfloor((C + F) \div ((2 - 6) \times (B + 5)))\rfloor$
 $((\lceil(F \div 4)\rceil) - (\lfloor(D \div 9)\rfloor)) - (6 + F)$
 $\lceil((C - F) \div ((3 - F) - (F \times F)))\rceil$

E informe abaixo, quantas linhas retornaram diferentes de 0.

4. Resolva as 6 linhas a seguir

$A \leftarrow 7$
 $B \leftarrow 3$
 $C \leftarrow 9$
 $D \leftarrow 5$
 $F \leftarrow 6$
 $(6 \times 2) + ((8 - 9) + (B \times 6))$
 $\lfloor((2 - F) \div ((5 + 6) + (4 - 5)))\rfloor$
 $(B \times 8) - ((B \times 7) + (\lceil(F \div F)\rceil))$

$((A \times F) - (3 - 3)) + (D \times 2)$
 $((D - F) \times (6 + 6)) - (\lceil(A \div 4)\rceil)$
 $((A + F) + (\lceil(3 \div F)\rceil)) - (\lfloor(F \div F)\rfloor)$

E informe abaixo, quantas linhas retornaram valores menores que 75.

5. Resolva as 6 linhas a seguir

$A \leftarrow 9$
 $B \leftarrow 4$
 $C \leftarrow 2$
 $D \leftarrow 5$
 $F \leftarrow 6$
 $((7 - F) \times (\lceil(4 \div F)\rceil)) + (D + F)$
 $(F - 8) \times ((B + F) - (\lfloor(C \div F)\rfloor))$
 $((5 + 8) \times (\lfloor(3 \div 5)\rfloor)) \times (F \times 2)$
 $\lceil(((D - F) \times (9 - 3)) \div (9 \times 5))\rceil$
 $\lfloor((\lceil(D \div 2)\rceil) \div ((B + F) \times (4 + 8)))\rfloor$
 $(A \times 7) + ((\lfloor(F \div F)\rfloor) \times (3 \times 8))$

E informe abaixo, quantas linhas retornaram valores pares.

6. Resolva as 6 linhas a seguir

$A \leftarrow 3$
 $B \leftarrow 9$
 $C \leftarrow 3$
 $D \leftarrow 3$
 $F \leftarrow 2$
 $(\lfloor(2 \div 8)\rfloor) \times ((6 \times 8) \times (A + 4))$
 $((\lfloor(4 \div 8)\rfloor) - (3 + F)) \times (C + F)$
 $(6 - F) + ((A - F) \times (D + F))$
 $(9 - 3) + ((4 \times 6) \times (4 \times 9))$
 $((\lfloor(4 \div 7)\rfloor) - (C + 2)) - (6 - 6)$
 $(\lceil(4 \div 9)\rceil) \times ((8 + 4) \times (F - 6))$

E informe abaixo, quantas linhas cujos resultados foram positivos eram números primos

Responda aqui . Para seu controle a soma dos 6 resultados e: 22 .

ímpares em 1	maiores que 50 em 2	nao zero em 3
menores que 75 em 4	pares em 5	positivos e primos

Funções inventadas

Nesta folha vão aparecer diversos conjuntos de 6 expressões aritméticas simples, envolvendo as operações de adição, subtração, multiplicação e divisão, todas conhecidas nossas, além de 3 funções novas, inventadas apenas para serem usadas nesta folha.

Elas vão ser hierarquizadas por parênteses, que devem ser respeitados. Há duas funções, chamadas teto (⌈) e chão (⌊). Estas operações foram definidas pelo matemático canadense Kenneth Iverson, no final da década de 50. (Curiosamente, ele comentou que o símbolo foi obtido raspando um pedaço do caracter “abre colchete” da máquina de escrever).

O funcionamento dessas operações é simples e pode ser definido assim:

Chão O chão devolve o próximo inteiro igual ou menor ao operando da função.

Teto O teto devolve o próximo inteiro igual ou maior ao operando da função

Eis alguns exemplos:

⌈3	3	o chão de 3 é 3
⌈5.01	5	o chão de 5.01 é 5
⌈-2.5	-3	o chão de -2.5 é -3
⌊3	3	o teto de 3 é 3
⌊3.5	4	o teto de 3.5 é 4
⌊1.9	2	o teto de 1.9 é 2

Uma questão importante envolvendo essas duas operações é que elas transformam um número real (eventualmente com casas decimais) em um número inteiro.

Funções novas As funções novas são:

- ⊛ Em $a ⊛ b$, o resultado é 1 se $a > b$ e é -1 se $a ≤ b$.
- ⊖ Em $a ⊖ b$, o resultado é 10 se a é divisor de b . O resultado é 5 se b é divisor de a e o resultado é 1 se nenhum dos dois casos acima ocorre. Note que se a ou b são iguais a zero, ou ambos, o resultado é 1. A ordem de investigação é exatamente a mostrada.
- ⊞ Em $a ⊞ b$ o resultado é:
3 se a é par e b é impar.
6 se a e b são pares.
9 se a é impar e b é par.
12 se ambos são impares.

Seja agora um exemplo com 6 operações aritméticas:

$A ← 2$
 $B ← 4$
 $C ← 5$
 $D ← 2$
 $F ← 7$
 $(B × 6) + (⌊(C ÷ F))$
 $(4 ⊖ F) + (7 ⊞ F)$
 $(⌊(2 ÷ F)) ⊞ (D + F)$
 $(D × F) × (D ⊞ F)$
 $(A + 3) + (B ⊖ 7)$
 $(A ⊖ F) ⊞ (F ⊞ 2)$

Cujas respostas foram: 24, 13, 3, 42, 6 e 12.

Mais um exemplo a fazer:

$A ← 5$
 $B ← 9$
 $C ← 2$
 $D ← 9$
 $F ← 4$
 $(5 + F) × (D ⊛ 3)$
 $(⌊(D ÷ F)) × (3 + F)$
 $(B ⊖ F) ⊛ (5 ⊞ 3)$
 $(7 - F) + (F + F)$
 $(C ⊖ F) ⊞ (3 × 2)$
 $(D - F) ⊖ ((6 ⊖ 6) + (⌊(8 ÷ 6)))$

Cujas respostas vão ser:

Para você fazer

1. Resolva as 6 linhas a seguir

$A ← 7$
 $B ← 7$
 $C ← 7$
 $D ← 2$
 $F ← 3$
 $(8 - F) × (2 ⊖ F)$
 $(2 × F) × (5 + F)$
 $(6 ⊛ 8) ⊖ (8 - 8)$
 $((C ⊖ F) × (2 ⊞ 4)) - (6 + F)$
 $(B × F) ⊛ (C - 5)$
 $(2 × F) + ((4 + 9) - (⌊(C ÷ F)))$

E informe abaixo, quantas linhas retornaram valores ímpares.

2. Resolva as 6 linhas a seguir

$A ← 4$
 $B ← 8$
 $C ← 9$
 $D ← 9$
 $F ← 3$
 $(3 × F) ⊖ (⌊(C ÷ 5))$
 $(8 + F) + (3 ⊛ 7)$
 $(⌈(8 ÷ F)) ⊖ (6 ⊞ F)$
 $(7 ⊞ 4) × (8 × F)$
 $(B ⊞ F) ⊛ (3 ⊖ F)$
 $(8 ⊞ F) ⊞ (⌊(B ÷ F))$

E informe abaixo, quantas linhas retornaram valores maiores que 50.

3. Resolva as 6 linhas a seguir

$A ← 3$
 $B ← 8$
 $C ← 2$
 $D ← 7$
 $F ← 3$
 $(D ⊞ 9) ⊛ (C - F)$
 $(F - F) ⊞ ((8 ⊖ F) + (⌈(F ÷ F)))$
 $(A ⊖ F) ⊛ ((⌊(4 ÷ F)) + (9 - F))$
 $((2 ⊖ F) × (C - F)) ⊛ (C + F)$
 $(8 ⊖ F) + (5 × F)$
 $(9 + 2) + (⌈(3 ÷ F))$

E informe abaixo, quantas linhas retornaram diferentes de 0.

4. Resolva as 6 linhas a seguir

$A ← 6$
 $B ← 5$
 $C ← 9$
 $D ← 4$
 $F ← 9$
 $(⌈((2 × F) ÷ (F ⊞ 5)))$
 $(D ⊛ 8) × (⌈(6 ÷ F))$
 $(6 - F) - (D ⊛ F)$
 $((4 × 6) ⊖ (A - 4)) ⊛ (7 + 6)$
 $(B ⊛ 2) × (C ⊞ 9)$
 $(⌊((5 ⊖ F) ÷ (⌈(8 ÷ F)))$

E informe abaixo, quantas linhas retornaram valores menores que 50.

5. Resolva as 6 linhas a seguir

$A ← 2$
 $B ← 5$
 $C ← 2$
 $D ← 2$
 $F ← 4$
 $(B - F) - (A + 9)$
 $(⌈((D - F) ÷ (D × 4)))$
 $(D ⊞ 3) ⊛ (C ⊖ F)$
 $(C ⊛ F) ⊞ (B ⊖ F)$
 $(A - F) ⊞ (9 ⊖ 8)$
 $(8 + F) ⊛ (8 ⊞ 3)$

E informe abaixo, quantas linhas retornaram valores pares.

6. Resolva as 6 linhas a seguir

$A ← 6$
 $B ← 6$
 $C ← 6$
 $D ← 2$
 $F ← 5$
 $(3 + F) + ((A + 9) ⊛ (C + 8))$
 $(B ⊖ 9) ⊛ (F - 5)$
 $(⌊(D ÷ 2)) - (F + 8)$
 $(2 ⊖ F) + (7 × 9)$
 $(⌊(7 ÷ 6)) - ((D × 2) + (⌈(A ÷ 3)))$
 $(⌈((C ⊛ F) ÷ (4 × F)))$

E informe abaixo, quantas linhas cujos resultados foram positivos eram números primos

ímpares em 1	maiores que 50 em 2	nao zero em 3
menores que 50 em 4	pares em 5	positivos e primos



Funções inventadas-2

Nesta folha vão aparecer diversos conjuntos de 6 expressões aritméticas simples, envolvendo as operações de adição, subtração, multiplicação e divisão, todas conhecidas nossas, além de 3 funções novas, inventadas apenas para serem usadas nesta folha.

Elas vão ser hierarquizadas por parênteses, que devem ser respeitados. Há duas funções, chamadas teto ($\lceil$) e chão ($\lfloor$). Estas operações foram definidas pelo matemático canadense Kenneth Iverson, no final da década de 50. (Curiosamente, ele comentou que o símbolo foi obtido raspando um pedaço do caracter “abre colchete” da máquina de escrever).

O funcionamento dessas operações é simples e pode ser definido assim: **Chão** O chão devolve o próximo inteiro igual ou menor ao operando da função. **Teto** O teto devolve o próximo inteiro igual ou maior ao operando da função.

Eis alguns exemplos:

3	3	o chão de 3 é 3
5.01	5	o chão de 5.01 é 5
-2.5	-3	o chão de -2.5 é -3
3	3	o teto de 3 é 3
3.5	4	o teto de 3.5 é 4
1.9	2	o teto de 1.9 é 2

Uma questão importante envolvendo essas duas operações é que elas transformam um número real (eventualmente com casas decimais) em um número inteiro.

Funções novas As funções novas são:

ρ Em $a \rho b$, se a é par, o resultado é o dobro de b . Se a é ímpar, o resultado é o triplo de b .

$\diamond$ Em $a \diamond b$, o resultado é 3 se a e b têm o mesmo sinal. Caso contrário, $a \diamond b$ é igual a 7.

∇ Em $a \nabla b$ o resultado é $20 - (a + b)$

Eis alguns exemplos: $5 \nabla 3$ é 12; $22 \nabla 2$ é -4; $5 \diamond 6$ é 3; $2 \rho 3$ é 6; $3 \diamond -2$ é 7; $0 \diamond 0$ é 3; $7 \rho 7$ é 21.

No caso de ρ , deve-se lembrar que $-3\%2=-1$ e não 1 como seria de se esperar. Então o teste completo de ímpar deve ser $1==\text{Math.abs}(a\%2)$.

No caso da função $\diamond$, considera-se o sinal do zero como sendo indeterminado, logo diferente de positivo e de negativo.

Seja agora um exemplo com 6 operações aritméticas:

$A \leftarrow 7$
 $B \leftarrow 2$
 $C \leftarrow 2$
 $D \leftarrow 3$
 $F \leftarrow 7$
 $(D \diamond F) + (\lfloor (4 \div 5) \rfloor)$
 $(B \rho F) \nabla (8 \rho 2)$
 $(8 - F) \diamond ((B \nabla F) + (4 \times F))$
 $\lfloor ((F \nabla F) \div (\lceil (4 \div 9) \rceil)) \rfloor$
 $(6 \nabla F) \diamond (8 \times 5)$
 $(9 \rho 5) \rho (B + 2)$

Cujas respostas foram: 3 2 3 6 3 12

Mais um exemplo a fazer:

$A \leftarrow 9$
 $B \leftarrow 6$
 $C \leftarrow 4$

$D \leftarrow 3$
 $F \leftarrow 7$
 $(4 \times 9) \diamond (\lceil (3 \div 5) \rceil)$
 $(F + 7) - (\lfloor (2 \div F) \rfloor)$
 $((C \rho 9) + (2 \times F)) \rho (B \rho F)$
 $(C + F) \rho (4 \diamond F)$
 $(6 - F) \rho ((F \nabla F) \rho (3 \nabla 8))$
 $((3 \nabla 2) \diamond (3 \diamond F)) + (4 \times 2)$

Cujas respostas vão ser:

Para você fazer

1. Resolva as 6 linhas a seguir

$A \leftarrow 3$
 $B \leftarrow 9$
 $C \leftarrow 9$
 $D \leftarrow 4$
 $F \leftarrow 2$
 $(B \times 6) \rho (7 - F)$
 $(6 \nabla F) \times (3 \diamond 2)$
 $\lfloor (((\lfloor (8 \div 3) \rfloor) - (9 \times F)) \div (C + F)) \rfloor$
 $(6 \nabla F) \diamond (B \nabla F)$
 $(D - 5) \nabla (8 \diamond 2)$
 $(C \times 6) + ((4 - 7) \nabla (7 - F))$

E informe abaixo, a soma dos resultados obtidos.

2. Resolva as 6 linhas a seguir

$A \leftarrow 8$
 $B \leftarrow 4$
 $C \leftarrow 4$
 $D \leftarrow 9$
 $F \leftarrow 7$
 $(7 - F) - (6 + 9)$
 $((5 \times 8) + (7 + F)) - (\lceil (C \div 9) \rceil)$
 $(3 \rho F) - (3 + F)$
 $((F \rho 7) - (F - 9)) \times (2 \rho F)$
 $(D \times F) - (\lceil (D \div F) \rceil)$
 $((\lfloor (5 \div F) \rfloor) \rho (C + 9)) \rho (8 \diamond F)$

E informe abaixo, a soma dos resultados obtidos

3. Resolva as 6 linhas a seguir

$A \leftarrow 4$
 $B \leftarrow 5$
 $C \leftarrow 2$
 $D \leftarrow 4$
 $F \leftarrow 7$
 $(5 \nabla F) \nabla (D + 7)$
 $(D \nabla 7) \nabla (9 - 3)$
 $(A \nabla 5) + (5 - F)$
 $(F \diamond 5) - (5 - F)$
 $(\lfloor (B \div 6) \rfloor) + (D \times 3)$
 $(6 - F) \rho (B \diamond 5)$

E informe abaixo, a soma dos resultados obtidos.

4. Resolva as 6 linhas a seguir

$A \leftarrow 3$
 $B \leftarrow 5$
 $C \leftarrow 2$
 $D \leftarrow 5$
 $F \leftarrow 3$
 $(7 \diamond 4) + (D \times 8)$
 $(7 + F) \nabla (7 + 3)$
 $\lceil ((A + 4) \div (F \diamond F)) \rceil$
 $(D \times 4) \diamond (3 + 3)$
 $(4 \times F) \diamond ((A \nabla F) \rho (5 \nabla F))$
 $(9 + 6) + (A + 5)$

E informe abaixo, a soma dos resultados obtidos.

5. Resolva as 6 linhas a seguir

$A \leftarrow 2$
 $B \leftarrow 8$
 $C \leftarrow 4$
 $D \leftarrow 4$
 $F \leftarrow 2$
 $(F \times 3) + (9 - 4)$
 $\lceil ((D \rho 4) \div (2 \nabla F)) \rceil$
 $(5 \diamond 6) \diamond (F \nabla 6)$
 $(6 \times 3) \times (\lceil (D \div F) \rceil)$
 $(C \rho 5) \nabla (C \times 6)$
 $((D \rho F) \times (8 - F)) \rho (\lfloor (6 \div 5) \rfloor)$

E informe abaixo, a soma dos resultados obtidos.

6. Resolva as 6 linhas a seguir

$A \leftarrow 4$
 $B \leftarrow 9$
 $C \leftarrow 7$
 $D \leftarrow 6$
 $F \leftarrow 3$
 $(9 + 6) \rho (D \nabla 7)$
 $(8 + F) \nabla ((4 + 8) \times (6 \rho F))$
 $(B - 7) + (\lceil (B \div 4) \rceil)$
 $(A \diamond 6) \rho (F \nabla 8)$
 $(8 \diamond F) \diamond ((9 + F) \diamond (7 \times F))$
 $(3 \times 2) + (F + 4)$

E informe abaixo, a soma dos resultados obtidos

Responda aqui

Para seu controle a soma dos 6 resultados foi: 736

1	2	3
4	5	6



Funções inventadas-3

Nesta folha vão aparecer diversos conjuntos de 6 expressões aritméticas simples, envolvendo as operações de adição, subtração, multiplicação e divisão, todas conhecidas nossas, além de 3 funções novas, inventadas apenas para serem usadas nesta folha.

Elas vão ser hierarquizadas por parênteses, que devem ser respeitados. Há duas funções, chamadas teto (↑) e chão (↓). Estas operações foram definidas pelo matemático canadense Kenneth Iverson, no final da década de 50. (Curiosamente, ele comentou que o símbolo foi obtido raspando um pedaço do caracter “abre colchete” da máquina de escrever).

O funcionamento dessas operações é simples e pode ser definido assim:

Chão O chão devolve o próximo inteiro igual ou menor ao operando da função.

Teto O teto devolve o próximo inteiro igual ou maior ao operando da função

Eis alguns exemplos:		
3	3	o chão de 3 é 3
5.01	5	o chão de 5.01 é 5
-2.5	-3	o chão de -2.5 é -3
3	3	o teto de 3 é 3
3.5	4	o teto de 3.5 é 4
1.9	2	o teto de 1.9 é 2

Uma questão importante envolvendo essas duas operações é que elas transformam um número real (eventualmente com casas decimais) em um número inteiro.

Funções novas As funções novas são:

↑ Em $a \uparrow b$, se $a \geq b$ ou se $b = 0$, o resultado é $a \times b$. Se $a < b$, o resultado é $a - b$.

↓ Em $a \downarrow b$, o resultado é $a + b + 3$.

↗ Em $a \nwarrow b$ o resultado é $a - b$ se $a > b$ e é $b - a$ se $b > a$. Em outras palavras, é sempre o maior – o menor. Claramente, se eles sao iguais, o resultado é 0.

Seja agora um exemplo com 6 operações aritméticas:

$A \leftarrow 3$
 $B \leftarrow 6$
 $C \leftarrow 6$
 $D \leftarrow 7$
 $F \leftarrow 3$
 $(5 - F) \uparrow ((\uparrow(F \div 4)) - (7 \nwarrow F))$
 $(3 - F) - (6 \uparrow F)$
 $(2 \uparrow 6) - ((6 - F) \nwarrow (A \downarrow F))$
 $(9 \times 8) - (B \downarrow 6)$
 $((2 + F) + (C - 8)) - (5 \uparrow 8)$
 $(5 + 9) \downarrow (4 + F)$
Cujas respostas foram: -6, -18, -10, 57, 6 e 24.

Mais um exemplo a fazer:

$A \leftarrow 5$
 $B \leftarrow 9$
 $C \leftarrow 7$
 $D \leftarrow 8$
 $F \leftarrow 9$

$(9 \times F) \downarrow (D + F)$
 $(C \nwarrow F) \times (6 \uparrow 4)$
 $(D \nwarrow F) \nwarrow (\downarrow(9 \div 5))$
 $(F - 4) + (A \nwarrow 6)$
 $((F \times F) + (\downarrow(6 \div F))) - (A \downarrow F)$
 $(F + F) \downarrow ((2 + F) \times (7 \times F))$
Cujas respostas vão ser:

Para você fazer

1. Resolva as 6 linhas a seguir

$A \leftarrow 8$
 $B \leftarrow 2$
 $C \leftarrow 7$
 $D \leftarrow 8$
 $F \leftarrow 3$
 $(D \downarrow 7) \downarrow (3 \uparrow 7)$
 $((8 \nwarrow F) + (7 - 5)) + (B \downarrow 6)$
 $(A \uparrow 4) \nwarrow (D \downarrow 9)$
 $((A + F) \uparrow (\downarrow(6 \div F))) \uparrow (F + 2)$
 $(D + 3) \uparrow (5 \times F)$
 $(5 \times F) \nwarrow (F - F)$

E informe abaixo, a soma dos resultados obtidos.

2. Resolva as 6 linhas a seguir

$A \leftarrow 5$
 $B \leftarrow 4$
 $C \leftarrow 8$
 $D \leftarrow 4$
 $F \leftarrow 6$
 $(A \downarrow 9) \downarrow (F - 2)$
 $(B \uparrow 4) \downarrow (A \nwarrow 8)$
 $(8 \downarrow 8) + ((2 - F) \downarrow (5 - F))$
 $(\downarrow(6 \div F)) + (A \downarrow F)$
 $(B - 8) \times ((B + F) \nwarrow (3 \times 3))$
 $(\downarrow(2 \div 7)) \uparrow (5 + F)$

E informe abaixo, a soma dos resultados obtidos

3. Resolva as 6 linhas a seguir

$A \leftarrow 2$
 $B \leftarrow 4$
 $C \leftarrow 6$
 $D \leftarrow 7$
 $F \leftarrow 3$
 $(9 + 2) - (8 \nwarrow 4)$
 $(A \uparrow F) \times (\downarrow(9 \div F))$
 $\downarrow((9 \nwarrow F) \div (\uparrow(6 \div 5)))$
 $(F \times 3) \downarrow (D - F)$
 $((F \downarrow F) - (B \times 7)) + (B \times 5)$
 $(9 - 6) \downarrow (\uparrow(9 \div 7))$

E informe abaixo, a soma dos resultados obtidos.

4. Resolva as 6 linhas a seguir

$A \leftarrow 7$
 $B \leftarrow 5$

$C \leftarrow 8$
 $D \leftarrow 8$
 $F \leftarrow 3$
 $(2 \downarrow 3) \nwarrow (D \uparrow 6)$
 $(5 \nwarrow 9) - (2 \uparrow F)$
 $(9 \uparrow F) \downarrow (C - 9)$
 $(C \uparrow F) \nwarrow (F \nwarrow 3)$
 $(F - F) + (8 \uparrow 9)$
 $(7 + F) + (9 \nwarrow F)$

E informe abaixo, a soma dos resultados obtidos.

5. Resolva as 6 linhas a seguir

$A \leftarrow 2$
 $B \leftarrow 5$
 $C \leftarrow 7$
 $D \leftarrow 3$
 $F \leftarrow 2$
 $(7 - 6) - ((B - 9) + (F - F))$
 $(D \downarrow F) - (4 \downarrow 6)$
 $(\downarrow(D \div F)) - ((D \times 7) + (7 + 3))$
 $(2 \nwarrow F) \nwarrow (7 \nwarrow F)$
 $((\uparrow(B \div F)) + (D \times F)) \nwarrow (8 + 3)$
 $((3 \downarrow 3) + (A - 6)) \uparrow (B - 2)$

E informe abaixo, a soma dos resultados obtidos.

6. Resolva as 6 linhas a seguir

$A \leftarrow 7$
 $B \leftarrow 3$
 $C \leftarrow 8$
 $D \leftarrow 2$
 $F \leftarrow 8$
 $(9 + 3) \downarrow (C \nwarrow F)$
 $(3 \nwarrow 3) \nwarrow (\uparrow(6 \div 2))$
 $(A \times 3) \downarrow (F + 2)$
 $(D \uparrow F) \uparrow (\downarrow(6 \div F))$
 $(C \nwarrow 8) \downarrow (2 - 9)$
 $(4 - F) \downarrow ((A \nwarrow F) \uparrow (B + 4))$

E informe abaixo, a soma dos resultados obtidos

Responda aqui

Para seu controle a soma dos 6 resultados foi: 409

1	2	3
4	5	6



- 1 - /

Funções inventadas-4

Nesta folha vão aparecer diversos conjuntos de 6 expressões aritméticas simples, envolvendo as operações de adição, subtração, multiplicação e divisão, todas conhecidas nossas, além de 3 funções novas, inventadas apenas para serem usadas nesta folha.

Elas vão ser hierarquizadas por parênteses, que devem ser respeitados. Há duas funções, chamadas teto ($\lceil$) e chão ($\lfloor$). Estas operações foram definidas pelo matemático canadense Kenneth Iverson, no final da década de 50. (Curiosamente, ele comentou que o símbolo foi obtido raspando um pedaço do caracter “abre colchete” da máquina de escrever).

O funcionamento dessas operações é simples e pode ser definido assim:

Chão O chão devolve o próximo inteiro igual ou menor ao operando da função.

Teto O teto devolve o próximo inteiro igual ou maior ao operando da função

Eis alguns exemplos:

$\lceil 3$	3	o chão de 3 é 3
$\lfloor 5.01$	5	o chão de 5.01 é 5
$\lfloor -2.5$	-3	o chão de -2.5 é -3
$\lceil 3$	3	o teto de 3 é 3
$\lceil 3.5$	4	o teto de 3.5 é 4
$\lceil 1.9$	2	o teto de 1.9 é 2

Uma questão importante envolvendo essas duas operações é que elas transformam um número real (eventualmente com casas decimais) em um número inteiro.

Funções novas As funções novas são:

$\wr$ Em $a \wr b$, se $a + b > 3$, o resultado é 1, senão é 2.

$\sim$ Em $a \sim b$, o resultado é 5 se $a \wr b = 1$. O resultado é igual a 10 senão.

$\top$ Em $a \top b$ o resultado é b se $a > 5$ e é a senão.

Seja agora um exemplo com 6 operações aritméticas:

$A \leftarrow 9$
 $B \leftarrow 7$
 $C \leftarrow 7$
 $D \leftarrow 3$
 $F \leftarrow 2$
 $(C + F) - (A \times F)$
 $(B \top 6) \sim (A \sim F)$
 $(C + 3) \sim (F \times F)$
 $(5 \times 6) - ((F - F) \times (C \top F))$
 $(8 - 7) - (\lfloor (C \div F) \rfloor)$
 $(5 \times 3) - (B \top 6)$
Cujas respostas foram: -9, 5, 5, 30, -2 e 9.
Mais um exemplo a fazer:

$A \leftarrow 4$
 $B \leftarrow 6$
 $C \leftarrow 8$
 $D \leftarrow 7$
 $F \leftarrow 9$
 $\lceil (((B \sim F) - (C \times F)) \div (3 \times 3)) \rceil$
 $\lceil ((2 \sim F) \div (F \sim 9)) \rceil$
 $((3 - 5) \sim (5 \times F)) - (C + 5)$
 $(B \wr 5) \top (4 \sim F)$

$(F \wr F) \wr (8 - 9)$
 $(F - F) \top ((9 \sim 2) + (4 - F))$
Cujas respostas vão ser:

Para você fazer

1. Resolva as 6 linhas a seguir

$A \leftarrow 7$
 $B \leftarrow 7$
 $C \leftarrow 8$
 $D \leftarrow 7$
 $F \leftarrow 7$
 $(8 - 6) - ((A + F) + (A \wr F))$
 $(F \top 2) \wr (B - 2)$
 $\lceil ((\lceil (A \div F) \rceil) \div (F \top 9)) \rceil$
 $(6 + F) \wr (F + 4)$
 $((\lfloor (6 \div F) \rfloor) \times (F - 8)) \times (D \top 6)$
 $\lfloor ((5 \top F) \div (\lceil (C \div F) \rceil)) \rfloor$

E informe abaixo, a soma dos resultados obtidos.

2. Resolva as 6 linhas a seguir

$A \leftarrow 9$
 $B \leftarrow 5$
 $C \leftarrow 8$
 $D \leftarrow 6$
 $F \leftarrow 3$
 $(7 \wr F) \times (7 \wr F)$
 $(2 \times F) \times ((D - F) \wr (3 \sim F))$
 $((F + 3) + (C \times F)) \times (\lfloor (2 \div 6) \rfloor)$
 $\lfloor ((F \sim 3) \div (4 \wr F)) \rfloor$
 $(9 \top 7) \top (7 \wr 2)$
 $(4 \wr 3) + (3 \wr 7)$

E informe abaixo, a soma dos resultados obtidos

3. Resolva as 6 linhas a seguir

$A \leftarrow 4$
 $B \leftarrow 4$
 $C \leftarrow 7$
 $D \leftarrow 2$
 $F \leftarrow 8$
 $(C + 3) \wr (C \wr F)$
 $(6 \wr 8) \times (6 \times F)$
 $\lceil ((D \times F) \div (C \top F)) \rceil$
 $\lceil (((B - F) \sim (4 + 5)) \div (F \sim F)) \rceil$
 $(8 \top F) \times (\lfloor (C \div F) \rfloor)$
 $(6 - 8) \sim ((A + 7) \wr (D + 9))$

E informe abaixo, a soma dos resultados obtidos.

4. Resolva as 6 linhas a seguir

$A \leftarrow 3$
 $B \leftarrow 3$
 $C \leftarrow 8$
 $D \leftarrow 9$
 $F \leftarrow 3$

$(9 \top 9) - (3 \sim F)$
 $(F \sim F) \wr ((9 - F) \wr (C \sim 9))$
 $(\lceil (2 \div F) \rceil) \times ((8 \times 3) \top (4 - F))$
 $(8 \times 7) \top (2 + 4)$
 $((5 \wr F) \top (9 \sim 3)) + (3 - 9)$
 $(B \wr 3) + (3 \times F)$

E informe abaixo, a soma dos resultados obtidos.

5. Resolva as 6 linhas a seguir

$A \leftarrow 9$
 $B \leftarrow 2$
 $C \leftarrow 3$
 $D \leftarrow 8$
 $F \leftarrow 7$
 $(B \times F) - (3 \sim F)$
 $(D \times 3) \top ((C - F) \sim (\lfloor (C \div 7) \rfloor))$
 $((D \wr 2) - (\lfloor (9 \div F) \rfloor)) \top (3 \wr 2)$
 $(\lceil (3 \div 8) \rceil) \top (B \wr F)$
 $(F + 5) \sim (7 - F)$
 $(C \times 2) - (C - F)$

E informe abaixo, a soma dos resultados obtidos.

6. Resolva as 6 linhas a seguir

$A \leftarrow 3$
 $B \leftarrow 5$
 $C \leftarrow 2$
 $D \leftarrow 4$
 $F \leftarrow 9$
 $(7 \times F) \wr (8 \top 6)$
 $(6 + F) + (8 + F)$
 $(8 \wr F) - (B + F)$
 $(F - F) - (3 - F)$
 $(6 \sim 3) + ((D \wr 8) - (F \top F))$
 $\lfloor (((D \wr 7) - (6 + 5)) \div (3 \top F)) \rfloor$

E informe abaixo, a soma dos resultados obtidos

Responda aqui
Para seu controle a soma dos 6 resultados é: 140 .

1	2	3
4	5	6



Funções inventadas-5

Nesta folha vão aparecer diversos conjuntos de 6 expressões aritméticas simples, envolvendo as operações de adição, subtração, multiplicação e divisão, todas conhecidas nossas, além de 3 funções novas, inventadas apenas para serem usadas nesta folha.

Elas vão ser hierarquizadas por parênteses, que devem ser respeitados. Há duas funções, chamadas teto (⌈) e chão (⌊). Estas operações foram definidas pelo matemático canadense Kenneth Iverson, no final da década de 50. (Curiosamente, ele comentou que o símbolo foi obtido raspando um pedaço do caracter “abre colchete” da máquina de escrever).

O funcionamento dessas operações é simples e pode ser definido assim:

Chão O chão devolve o próximo inteiro igual ou menor ao operando da função.

Teto O teto devolve o próximo inteiro igual ou maior ao operando da função

Eis alguns exemplos:		
5.01	5	o chão de 5.01 é 5
-2.5	-3	o chão de -2.5 é -3
3	3	o teto de 3 é 3
1.9	2	o teto de 1.9 é 2

Uma questão importante envolvendo essas duas operações é que elas transformam um número real (eventualmente com casas decimais) em um número inteiro.

Funções novas As funções novas são:

⌊ Em $a \text{ ⌊ } b$, se o resto da divisão de b por a é 1, o resultado é $a \times b$; se o resto da divisão de b por a é 2, o resultado é $a \times b \times 2$; se o resto da divisão de b por a é 3, o resultado é $a \times b \times 4$; se nenhum dos casos anteriores é verdadeiro, então o resultado é $a \times b \times 3$.

⌋ Em $a \text{ ⌋ } b$, o resultado é $\lfloor a \times \frac{1}{3}b \rfloor$ se $a > b$. O resultado é $\lceil a \times \frac{1}{2}b \rceil$ se $a < b$. Se $a = b$ então o resultado é igual a $a + b$.

⊕ Em $a \text{ ⊕ } b$ o resultado é $a + b + 1$.

Seja agora um exemplo com 6 operações aritméticas:

$$\begin{aligned} A &\leftarrow 9 \\ B &\leftarrow 2 \\ C &\leftarrow 2 \\ D &\leftarrow 6 \\ F &\leftarrow 2 \\ \lceil ((F + F) \div (7 \text{ ⌊ } F)) \\ (B \text{ ⊕ } F) \text{ ⌋ } (8 \times 5) \\ (A \text{ ⌋ } F) \text{ ⌊ } (A \text{ ⊕ } 5) \\ \lceil ((4 \text{ ⌊ } F) \div (D \text{ ⊕ } 5)) \\ ((3 \times F) \text{ ⌊ } (5 \text{ ⌊ } F)) - (B \times 7) \\ (\lceil (2 \div 6)) - (B \text{ ⌋ } F) \end{aligned}$$

Respostas: 1, 100, 225, 2, 226 e -3.

Mais um exemplo a fazer:

$$\begin{aligned} A &\leftarrow 7 \\ B &\leftarrow 2 \\ C &\leftarrow 5 \\ D &\leftarrow 2 \end{aligned}$$

$$\begin{aligned} F &\leftarrow 3 \\ (D \text{ ⌋ } 4) + ((A - F) + (F \text{ ⊕ } 7)) \\ \lfloor ((6 \text{ ⌊ } F) \div (B \text{ ⌋ } F)) \\ (C \text{ ⌋ } 6) \text{ ⌊ } (A + F) \\ (F + F) \text{ ⌋ } (D - F) \\ (C \text{ ⌋ } 9) - (F \text{ ⌊ } F) \\ (3 \text{ ⌋ } 8) \text{ ⊕ } (\lfloor (5 \div F)) \end{aligned}$$

Para você fazer

1. Resolva as 6 linhas a seguir

$$\begin{aligned} A &\leftarrow 4 \\ B &\leftarrow 3 \\ C &\leftarrow 6 \\ D &\leftarrow 5 \\ F &\leftarrow 3 \\ (6 \text{ ⌊ } F) \times (D + F) \\ (D - 5) \text{ ⊕ } ((F \text{ ⊕ } F) + (7 + 2)) \\ (9 \text{ ⌋ } F) - (9 \text{ ⌋ } F) \\ (5 \text{ ⊕ } 5) + (4 - 2) \\ (A \times F) + (4 - 3) \\ (\lceil (9 \div F)) + ((F \times 6) \text{ ⊕ } (F + F)) \end{aligned}$$

E informe abaixo, a soma dos resultados obtidos.

2. Resolva as 6 linhas a seguir

$$\begin{aligned} A &\leftarrow 2 \\ B &\leftarrow 2 \\ C &\leftarrow 3 \\ D &\leftarrow 6 \\ F &\leftarrow 3 \\ (\lceil (A \div F)) + (7 \text{ ⌋ } 4) \\ (\lfloor (8 \div F)) + (C - 9) \\ (4 \times 3) \times (B \text{ ⌊ } 3) \\ (C + 9) \text{ ⌊ } (D + F) \\ (B \text{ ⌊ } F) \text{ ⌋ } (2 \times 7) \\ (A \text{ ⌊ } 7) \text{ ⌋ } (C \text{ ⊕ } F) \end{aligned}$$

E informe abaixo, a soma dos resultados obtidos

3. Resolva as 6 linhas a seguir

$$\begin{aligned} A &\leftarrow 4 \\ B &\leftarrow 7 \\ C &\leftarrow 2 \\ D &\leftarrow 6 \\ F &\leftarrow 3 \\ (F \text{ ⌋ } F) \text{ ⌊ } (9 \text{ ⊕ } 7) \\ (C \text{ ⊕ } 4) \times (D \text{ ⊕ } 6) \\ (\lfloor (7 \div 6)) - (2 \times 7) \\ (A - F) \text{ ⊕ } (4 \times 6) \\ (D - F) - (D + 8) \\ (8 - 8) - (A \times F) \end{aligned}$$

E informe abaixo, a soma dos resultados obtidos.

4. Resolva as 6 linhas a seguir

$$A \leftarrow 4$$

$$\begin{aligned} B &\leftarrow 3 \\ C &\leftarrow 5 \\ D &\leftarrow 6 \\ F &\leftarrow 8 \\ \lceil ((9 \text{ ⊕ } 5) \div (5 \text{ ⊕ } F)) \\ (6 \text{ ⌋ } 5) \text{ ⌋ } (\lceil (C \div 4)) \\ (C - F) + (5 \times 9) \\ (6 + F) - (6 \text{ ⊕ } F) \\ (3 \text{ ⊕ } F) - (4 + F) \\ (5 \text{ ⊕ } 5) - (B \text{ ⌊ } 7) \end{aligned}$$

E informe abaixo, a soma dos resultados obtidos.

5. Resolva as 6 linhas a seguir

$$\begin{aligned} A &\leftarrow 9 \\ B &\leftarrow 9 \\ C &\leftarrow 6 \\ D &\leftarrow 4 \\ F &\leftarrow 6 \\ (\lceil (A \div 3)) \times (A \text{ ⌋ } 5) \\ (\lfloor (C \div 5)) + (5 \text{ ⌊ } 8) \\ (6 \times 4) \text{ ⊕ } (A \text{ ⊕ } 6) \\ \lceil (((3 - F) \times (3 + 3)) \div (B \text{ ⌊ } F)) \\ \lfloor (\lceil (3 \div F)) \div (C \text{ ⌋ } 5)) \\ (7 \text{ ⌋ } F) \text{ ⌊ } (9 - F) \end{aligned}$$

E informe abaixo, a soma dos resultados obtidos.

6. Resolva as 6 linhas a seguir

$$\begin{aligned} A &\leftarrow 4 \\ B &\leftarrow 8 \\ C &\leftarrow 6 \\ D &\leftarrow 3 \\ F &\leftarrow 6 \\ (C \times F) \text{ ⌊ } (C \text{ ⌊ } F) \\ (8 - 8) + (B + 6) \\ (B \text{ ⌊ } F) \text{ ⌋ } (8 - F) \\ (8 \times 7) \text{ ⌊ } (2 \text{ ⊕ } F) \\ (9 \text{ ⌋ } 6) - (7 \text{ ⌊ } F) \\ (4 \text{ ⊕ } 4) \times (6 \text{ ⌊ } 8) \end{aligned}$$

E informe abaixo, a soma dos resultados obtidos

Responda aqui

Para seu controle, a soma das 6 respostas é: 16006

1	2	3
4	5	6



Algoritmos

Um conceito

Um algoritmo é um método finito, escrito em um vocabulário simbólico fixo, regido por instruções precisas, que se movem em passos discretos, cuja execução não requer *insight*, esperteza, intuição, inteligência, ou clareza e lucidez e que mais cedo ou mais tarde chega a um fim. (David Berlinski)

O algoritmo é o que está por trás de todos os programas de computadores. Fazendo uma analogia com a Rose (a empregada robot da família Jetson).



Quando ela apresenta uma bela torta de maçãs, a torta é o produto final, a Rose é o computador e a receita da torta é o algoritmo.

Algoritmos são bem antigos na história do mundo. Provavelmente o mais antigo que é conhecido é devido a Euclides. Para poder observá-lo, vamos relembrar uma operação da aritmética básica. Trata-se da divisão inteira. Quando os números inteiros A e B são divididos em divisão inteira, dois resultados aparecem: o quociente e o resto.

Por exemplo, na divisão inteira $10 \div 7$, o quociente (resultado) é 1, enquanto o resto é 3.

Nos nossos algoritmos, o quociente da divisão inteira vai ser representado pela palavra *div*, já que o símbolo $\div$ e mesmo a barra simples $/$ ficarão reservados para a divisão real. Portanto poderemos afirmar que

$$10 \text{ div } 7 = 1$$

Da mesma maneira, o resto da divisão inteira, será representado pela palavra *mod*, e aqui podemos afirmar que

$$10 \text{ mod } 7 = 3$$

Para treinar

Procure calcular e responder:

1. $10 \text{ div } 2 =$ _____.
2. $10 \text{ mod } 2 =$ _____.
3. $100 \text{ div } 33 =$ _____.
4. $100 \text{ mod } 33 =$ _____.
5. $8 \text{ div } 2 =$ _____.
6. $8 \text{ mod } 2 =$ _____.
7. $45 \text{ div } 11 =$ _____.
8. $45 \text{ mod } 11 =$ _____.
9. $2 \text{ div } 13 =$ _____.
10. $2 \text{ mod } 13 =$ _____.

Máximo Divisor Comum

Define-se o máximo divisor comum entre dois números inteiros A e B como sendo o maior número inteiro que divide simultaneamente tanto A como B sem deixar resto.

Por exemplo, se quisermos achar o MDC entre 10 e 14, teremos que achar os divisores de 10 (que são: 1, 2, 5 e 10) e de 14 (que são 1, 2, 7, 14) e localizar o maior número que é comum às duas listas, no caso 2. Assim, 2 é o MDC entre 10 e 14.

Para treinar

Informe qual o MDC entre

1. $MDC(15, 18) =$ _____.
2. $MDC(20, 40) =$ _____.
3. $MDC(17, 31) =$ _____.
4. $MDC(2, 11) =$ _____.

Do exercício anterior percebeu-se que quando um dos números é primo, o MDC é sempre 1. Isto decorre da definição dos números primos: Um número inteiro é primo, quando seus únicos divisores inteiros são a unidade e ele próprio.

Algoritmo do Máximo Divisor Comum

Este algoritmo sempre é chamado com dois inteiros. Note que o primeiro número deve ser maior ou igual do que o segundo. Se esta condição não estiver satisfeita, os números devem ser INVERTIDOS antes de serem entregues ao algoritmo. Este algoritmo recursivo é devido a Euclides

```
1: inteiro função MDC(inteiro a, b) {obrigatoriamente:  $a \geq b$ }
2: se  $b = 0$  então
3:   devolva  $a$ 
4:   esta é outra maneira de identificar comentários
5: senão
6:   devolva  $MDC(b, (a \text{ mod } b))$ 
7: fim se
8: fim {função}
```

Suponha que far-se-á a chamada ao algoritmo acima, como segue:

MDC(1200, 1119)
 0 que aconteceria ?
 Chamando com 1200 1119, Chamando com 1119 81, Chamando com 81 66,
 Chamando com 66 15, Chamando com 15 6, Chamando com 6 3 e
 Chamando com 3 0
 Resposta é 3

Para você fazer

Algoritmo máximo divisor comum Chame o algoritmo mdc com os valores

1963 1540

Responda no quadro próprio, qual o MDC pedido

Algoritmos no dia a dia Na nossa vida, também usamos muitos algoritmos. Aqui eles têm uma característica tão restrita como os algoritmos computacionais, porque supostamente eles serão seguidos por seres inteligentes. Mas, vale a experiência: acompanhe o algoritmo que um de nós poderia usar para ir do Unicenp até a Praça Tiradentes

```
1: Algoritmo para ir do Unicenp até a Praça Tiradentes
2: se tenho dinheiro ? então
3:   se tenho pressa ? então
4:     chamo um taxi
5:     mando ele ir para a Praça
6:   senão
7:     procuro uma carona OU
8:     vou para o ponto do ônibus
9:   ao desembarcar, pergunto qual a direção da Praça
10:   sigo andando até chegar nela
11:   senão
12:     saio do Unicenp pela JK, pegando à direita
13:     pego a BR277 à direita, sigo até a Pe Agostinho
14:     pego a Pe Agostinho à esquerda e sigo até a Al. Cabral
15:     viro à direita até a Carlos de Carvalho
16:     viro à esquerda e sigo até a Praça Tiradentes
17:   fim se
18: fim se
```

Para você fazer

Construa um algoritmo que permita que um colega seu possa

Descobrir em qual dia da semana será o Natal do próximo ano

Continue a resposta no verso, se necessário



- 1 - /

Matemática Básica

Este exercício é para relembrar conceitos da matemática básica (regra de sinais, parênteses, prioridades, ...) junto com alguns conceitos novos: teto e chão.

Cada exercício vai apresentar 5 expressões aritméticas que sempre darão como resposta um número inteiro. Você deve calculá-las todas e depois deve ordenar (do menor para o maior) os 5 resultados achados. Finalmente, deve responder qual a letra correspondente à expressão que ficou exatamente no meio da lista ordenada.

Por exemplo, se a expressão A deu 8, a B deu -4, a C deu 0, a D deu 22 e a E deu 12, a ordenação levará a -4, 0, 8, 12 e 22. Ou usando a identificação das expressões: B, C, A, E, D. A expressão do meio (e a resposta deste exercício) é A.

Operações A seguir, a lista de operações usadas neste exercício:

nome	definição	exemplo
+	adição convencional	$3 + 7 = 10$
-	subtração convencional	$5 - 8 = -3$
×	amultiplicação convencional	$4 \times 3 = 12$
÷	divisão convencional	$9 \div 3 = 3$
	resto da divisão inteira	$16 3 = 1$
⌊	chão	$\lfloor 3.8 = 3$
⌈	teto	$\lceil 3.8 = 4$

Exemplo 1

- A $((\lceil(3 \div 2) - (3 - 2)) | (\lceil(3 \div 3)))$
- B $((\lfloor(9 \div 4) + (6|9)) \times (2 - 9))$
- C $(\lfloor(8 \div 3) + (4 \times 7))$
- D $(\lceil((2 + 3) \div (6 \times 8)))$
- E $((2 + 2) \times (5|8)) + (7 \times 9)$

Ao executar estes 5 comandos, obtiveram-se os valores 0, -56, 30, 1, 83. Colocando-os em ordem tem-se: -56(B), 0(A), 1(D), 30(C) e 83(E). Como se pode ver o elemento do meio é o 1, consequentemente a resposta correta é D.

Exemplo 2

- A $(2 - 3) \times (6 - 4)$
- B $(\lceil((3 - 9) \div (5 + 5)))$
- C $(\lfloor(9 \div 2)) | (\lceil(5 \div 2))$
- D $((6 - 9) | (\lfloor(7 \div 4))) \times (7 - 3)$
- E $(\lfloor((4|4) \div (5|6)))$

Neste caso, as respostas são -2, 0, 1, 0, 0. Pondo em ordem: -2(A), 0(B), 0(D), 0(E) e 1(C). A resposta correta é D.

Exemplo 3

- A $((3 \times 6) \times (4 + 6)) \times (4|3)$
- B $((5 - 7) \times (7 \times 3)) | (\lceil(6 \div 4))$
- C $(9 \times 5) \times (4 \times 9)$
- D $(\lfloor((9 - 7) \div (4 - 9)))$
- E $(\lfloor((\lceil(2 \div 9)) \div (\lfloor(6 \div 2))))$

Respostas: 180, 0, 1620, -1, 0. Em ordem: -1(D), 0(B), 0(E), 180(A) e 1620(C). Resposta é E.

👉 Para você fazer

Exercício 1

- A $(7 - 2) - (\lfloor(6 \div 2))$
- B $(4 - 3) + (5|4)$
- C $(6 - 5) | (\lceil(5 \div 6))$
- D $((7 + 4) | (8 + 5)) + (9 + 4)$
- E $(\lfloor((5|2) \div (6|5)))$

Exercício 2

- A $(6 \times 5) + (5|6)$
- B $((6 \times 3) | (6 + 3)) + (4 - 6)$
- C $(\lfloor(5 \div 6)) | (5|2)$
- D $(9 + 3) | (5 - 4)$
- E $(9 + 5) \times (7|4)$

Exercício 3

- A $(\lfloor((7 - 6) \div (9 - 2)))$
- B $(9 \times 3) \times (7 + 4)$
- C $(4 - 5) + (5|3)$
- D $(9 \times 5) - (9|2)$
- E $((5 \times 5) - (7 + 2)) \times (\lceil(8 \div 4))$

Exercício 4

- A $((6 - 2) | (6 \times 6)) - (8|2)$
- B $(5 \times 4) | (7 - 6)$
- C $(7 + 2) | (4 + 3)$
- D $(4 - 5) + (7|5)$
- E $(5 \times 3) \times (9 - 3)$

Exercício 5

- A $(5 + 2) \times (6 - 3)$
- B $(9 \times 3) \times (8|4)$
- C $((\lfloor(6 \div 5)) | (\lfloor(7 \div 4))) \times (7 \times 5)$
- D $(9 + 6) + (\lfloor(7 \div 3))$
- E $(6 + 6) | (9 \times 2)$

Exercício 6

- A $(\lceil(7 \div 6)) - (7 + 6)$
- B $(\lceil((\lceil(6 \div 2)) \div (4 - 3)))$
- C $(8|3) + (\lceil(8 \div 2))$
- D $(9 - 6) \times (4 + 2)$
- E $(8|6) - (\lfloor(9 \div 6))$

Exercício 7

- A $(6 - 4) - (4 \times 4)$
- B $(6 - 3) + (7|4)$
- C $(\lfloor(7 \div 4)) + (\lceil(5 \div 3))$
- D $((5 \times 5) | (8 - 2)) - (7|6)$
- E $((8 - 2) + (6 \times 2)) - (8 + 4)$

Exercício 8

- A $(\lfloor(6 \div 4)) \times (4|6)$
- B $((5 \times 6) \times (\lceil(4 \div 3))) + (8 + 2)$
- C $(\lceil((6 - 5) \div (\lfloor(4 \div 4))))$
- D $(4 - 6) + (7 + 5)$
- E $((\lceil(6 \div 5)) - (5|3)) + (\lceil(9 \div 4))$

Exercício 9

- A $((9 - 4) + (\lceil(4 \div 6))) - (5 \times 3)$
- B $(\lceil((9|2) \div (5 + 6)))$
- C $(8|3) \times (7|2)$
- D $(6 + 3) | (8 + 5)$
- E $(\lfloor((6|2) \div (6|5)))$

Exercício 10

- A $(\lceil((9 \times 6) \div (4 \times 6)))$
- B $(\lceil((6|5) \div (\lfloor(8 \div 2))))$
- C $(8 + 2) | (6 \times 2)$
- D $(6 \times 6) - (\lceil(7 \div 3))$
- E $(6 - 2) \times (4|6)$

Para todos os 10 exercícios, calcule o valor de cada expressão, coloque-as em ordem crescente (respeite a ordem lexicográfica em caso de empates) e escreva a letra correspondente ao resultado que ocupa a posição central na lista (ou seja, o terceiro valor).

1	2	3	4	5
6	7	8	9	10

- 1 - /

Jogo da Vida

O Jogo da vida foi desenvolvido pelo matemático britânico John Horton Conway em 1970. Começou com uma brincadeira mas depois virou coisa séria. Esta pesquisa vem na sequência dos últimos trabalhos de Alan Turing, que um pouco antes de morrer trabalhava no tratamento matemático da herança. O que Watson e Creek estudavam (a molécula do DNA do ponto de vista biológico e químico), Turing estudava do ponto de vista matemático. Permanece como um imenso "SE" o que teria sido descoberto se ele não tivesse morrido. Para saber mais, veja "Computadores Auto Replicantes", VIVO807.

O jogo foi criado de modo a reproduzir, através de regras simples, as alterações e mudanças em grupos de seres vivos, tendo aplicações em diversas áreas da ciência.

As regras definidas são aplicadas a cada nova "geração"; assim, a partir de uma imagem em um tabuleiro bi-dimensional definida pelo jogador, percebem-se mudanças muitas vezes inesperadas e belas a cada nova geração, variando de padrões fixos a caóticos.

Origem Um dos problemas matemáticos mais famosos dos anos 40 era o de achar uma máquina que fosse capaz de construir cópias de si mesma, que teve uma solução baseada em um autômato celular extremamente engenhoso e complicado inventado pelo renomado matemático John von Neumann. Conway inventou o Jogo da Vida (ou Game of Life) ao utilizar suas descobertas anteriores relacionadas com o problema de encontrar um grupo simétrico de esferas em 24 dimensões, proposto por John Leech para simplificar a solução de von Neumann.

O jogo fez sua primeira aparição na edição de Outubro de 1970 da Scientific American, na coluna de jogos matemáticos de Martin Gardner. De um ponto de vista teórico, ele é interessante, pois tem o poder de uma máquina de Turing universal: tudo pode ser computado através de algoritmos no Jogo da Vida de Conway. Também era dito desde 1970 que foi destinado mais tempo de computação ao Jogo da Vida do que a qualquer outra atividade.

Desde sua publicação, ele tem atraído muito interesse devido aos caminhos surpreendentes que pode tomar. Life é um exemplo de auto-organização. Ele é interessante para biólogos, matemáticos, economistas, filósofos e outros a observar o modo como imagens complexas podem surgir de implementações de regras muito simples.

O Jogo da Vida tem um número de imagens reconhecidas que emergem de posições iniciais particulares. Antes da publicação, muitas imagens interessantes já haviam sido descobertas, incluindo o sempre envolvente Repentominó, o auto propulsionado "glider", e várias "guns" (armas) que geravam um fluxo sem fim de novas imagens, todas aumentando o interesse no jogo. Sua popularidade foi ajudada pelo fato de ele ter vindo justo no momento em que uma nova geração de minicomputadores de baixo custo estavam sendo disponibilizados no mercado, significando que o jogo poderia ser rodado por horas nessa máquinas que não eram utilizadas de noite. Isto escondeu a popularidade posterior dos fractais gerados por computador. Para muitos aficionados, Life era simplesmente um desafio de programação, um método divertido de gastar os ciclos da CPU. Para alguns, no entanto, Life tinha conotações mais filosóficas. Ele desenvolveu um culto através dos anos 70 e no meio dos 80; os desenvolvimentos atuais foram tão longe a ponto de criar emulações teóricas de sistemas de computadores com as regras de um tabuleiro de Life.

Regras do Jogo da Vida As regras são simples e elegantes:

- Qualquer célula viva com menos de dois vizinhos vivos morre de solidão.
- Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação.
- Qualquer célula com exatamente três vizinhos vivos se torna uma célula viva.
- Qualquer célula com dois vizinhos vivos continua no mesmo estado para a próxima geração.

É importante entender que todos os nascimentos e mortes ocorrem simultaneamente. Juntos eles constituem uma geração ou, como podemos chamá-los, um "instante" na história da vida completa da configuração inicial.

Descrição Este "jogo" é na realidade um jogo sem jogador, o que quer dizer que sua evolução é determinada pelo seu estado inicial, não necessitando de nenhuma entrada de jogadores humanos. Ele é jogado em um conjunto de células quadradas que seguem ao infinito em todas as direções. Cada célula tem oito "vizinhos", que são as células adjacentes, incluindo as diagonais. Cada célula pode estar em dois estados: "viva" ou "morta". (Também são usados os termos "ligado" e "desligado".) O estado do tabuleiro evolui e se modifica em pequenas passagens de tempo. Os estado de todas as células em um instante são considerados para calcular o estado de todas as células no instante seguinte. Todas as células são atualizadas simultaneamente. As transições dependem apenas do número de vizinhos vivos (ver as regras acima).

A idéia básica do "jogo" é começar com uma configuração simples de células vivas (organismos) que são colocadas em um tabuleiro 2D de vários métodos. Isto constitui a primeira geração. As "leis genéticas" de Conway para nascimentos, mortes e sobrevivência (as quatro regras acima) são então aplicadas e a nova geração é então colocada de acordo. Geração a geração os "jogador(es)" observam as várias imagens que surgem.

A vida sempre segue De uma imagem inicial de células vivas no tabuleiro, percebe-se que, conforme passam as gerações, a população anda constantemente de modo não usual, algumas vezes belo e quase sempre inesperado, porém muda. Em muitos poucos casos a sociedade eventualmente morre (todas as células se tornam células mortas), apesar de que isso pode não acontecer até que ocorram uma série de novas gerações. A maioria das imagens iniciais chegam a figuras estáveis - Conway chama-as de "vida parada" - que não podem mudar ou imagens que oscilam para sempre. Imagens sem simetria inicial tendem a se tornar simétricas. Uma vez que isto ocorre, a simetria não pode mais ser perdida, apesar de poder crescer em riqueza.

Conway originalmente supôs que nenhuma imagem poderia crescer ilimitadamente. Em outras palavras, qualquer configuração com um número finito de células não pode crescer além de um limite finito superior ao número de células do campo. Esta foi provavelmente a mais profunda e mais difícil questão colocada pelo jogo atualmente. Conway ofereceu um prêmio de \$50 à primeira pessoa que pudesse provar a afirmação ou seu contrário antes do fim de 1970. Um caminho para provar seu contrário era ser capaz de descobrir imagens que continuassem adicionando contadores ao campo: Uma "gun" (uma configuração que constantemente atira objetos que se movimentam, como o "glider") ou um "puffer train" (uma configuração que se move mas deixa atrás uma trilha de "fumaça"). O prêmio foi ganho em novembro do mesmo ano por um time do MIT, a configuração inicial cresce como um gun, emitindo seu primeiro glider na 40ª geração. A gun emite então um novo glider a cada 30 gerações.

Imagens Existe uma série de diferentes imagens que podem ocorrer no Jogo da Vida, incluindo imagens paradas ("vidas paradas"), imagens repetitivas ("osciladores" - o conjunto de vidas paradas), e imagens que se movimentam através do tabuleiro ("naves espaciais").

Imagens chamadas "Matusalens" podem evoluir por longos períodos de tempo antes de se repetir. "Diehard" é uma imagem que eventualmente desaparece após 130 gerações, ou passos. "Acorn" leva 5206 gerações para criar 13 gliders depois se estabiliza como vários osciladores.

Desde então, uma série de construções complicadas têm sido feitas, incluindo portas lógicas de gliders, um somador, um gerador de números primos e uma unidade de célula que emula o Jogo da Vida em uma escala muito maior e a passos lentos.

É possível construir portas lógicas AND, OR e NOT usando gliders. É possível construir uma imagem que aja como uma máquina de estado finito conectada a dois contadores. Isto possui o mesmo poder computacional de uma máquina de Turing universal (veja contador para a prova). Jogo da Vida é tão poderoso quanto qualquer computador com memória ilimitada: é o Turing completo. Além disso, uma imagem pode conter um conjunto de guns que se combina para construir novos objetos, incluindo cópias da imagem original. Um "construtor universal" pode ser construído que contenha um computador Turing completo, e que pode construir uma série de objetos complexos, incluindo mais cópias de si mesma.

Para este exercício considere que o tabuleiro está rodeado por células mortas.

Filosofia O Jogo da Vida de Conway cria um universo complicado a partir de poucas regras. É imaginável que todo o universo real seja um jogo similar, jogado em alguma dimensão mais alta. Esta possibilidade pode influenciar o debate de vivermos em um universo perfeitamente ajustado que requer um projeto inteligente. (texto fortemente baseado em pt.wikipedia.org)

Para você fazer

Seja o seguinte tabuleiro inicial, onde '.' significa célula morta e 'X' significa célula viva. Este tabuleiro corresponde à **geração ZERO**, portanto você deve calcular 4 gerações a seguir.

1-	.	.	.	.	.	.	.	.	X
2-	X	.	.	.	.	X	X	.	X
3-	.	.	.	X	X	.	X	.	.
4-	.	X	.	X	.	.	.	.	.
5-	.	.	.	X	.	.	.	.	.
6-	X	.	.	.	.	.	.	.	.
7-X	X	X	X	.	X	.	.	X	.
8-	.	.	.	.	X	.	.	.	X
9-	.	.	X	X	.	X	X	.	.
10-X	X	X	.	.	.	.	X	.	.

Informe a seguir se a posição pedida, após 4 gerações, está viva ou morta. Indique que ela está viva escrevendo **1** e que está morta escrevendo **0**.

L = 2 , C = 6	L = 2 , C = 8	L = 3 , C = 5	L = 3 , C = 6



- 1 - /

Dia 3 de setembro de 1989 Maracanã lotado para assistir ao Brasil X Chile, pelas eliminatórias da Copa do Mundo de 90. Longe dali, em algum ponto a princípio entre Marabá e Belém, Cezar Augusto Garcez comanda um voo cego. Perdido em pleno ar, tenta se posicionar. Localiza, em vão, uma rádio que transmitia a partida que entraria para a história como “o jogo da fogueteira”. Aquele voo também estaria nos jornais no dia seguinte: “Avião desaparece na Amazônia”, publicou O Globo.

Herói e vilão, ao mesmo tempo, o comandante Garcez é a principal personagem do RG-254 que caiu na selva amazônica em 1989. O que deveria ser um voo rotineiro se transformou numa tragédia. Desorientado, Garcez permaneceu durante três horas em voo cego. Temendo que o erro fosse descoberto, passou diversas informações truncadas para a base, afirmou estar onde não estava. Sem combustível, arriscou o aparentemente impossível: um pouso na copa das árvores, em plena noite, com visibilidade praticamente nula. Garcez foi acusado de negligenciar rotinas básicas da aviação. Por outro lado, salvou a vida de muitos passageiros ao conseguir aterrissar a aeronave e cuidar dos feridos. Foi condenado a 12 anos em 1991, e a instância superior reafirmou a sentença. Ficou 2 anos preso.

A desorientação completa em voo, embora rara (todo o treinamento dos pilotos é para impedir isso), de vez em quando dá as caras. Há um episódio famoso, durante a crise dos mísseis em Cuba, em que um piloto de U2, Charles Maultsby, resolveu se perder no Polo Norte. Desorientado, acabou entrando mais de 500 Km em território da antiga União Soviética.

Teve uma sorte dos diabos, se achou e retornou, mas o dia escolhido por ele para se perder não podia ser pior: Foi dia 27 de outubro de 1962, o dia em que o mundo esteve mais próximo da IV Guerra Mundial, que seria também a última: a humanidade teria sido aniquilada. Aliás, os russos juravam que a missão do U2 era preparar a invasão dos mísseis.

GPS Na década de 70, os EUA começaram a projetar e implementar o sistema conhecido como GPS (Global Positioning System). O primeiro satélite *Navstar* foi lançado em 1978. O objetivo principal era permitir que os estimados 40.000 usuários militares americanos se orientassem sobre a terra. Os civis começaram a usar o sistema a partir da década de 80. Hoje, estima-se em mais de 50.000.000 de dispositivos capazes de receber os sinais de GPS e apresentar uma estimativa de localização.

Além do sistema americano, existe o russo *Glonass* de uso exclusivo pelos militares e a partir da última semana de 2005, o sistema *Galileo* construído e operado pela Agência Espacial Européia. Há também o chinês, chamado *Beidou*.

A grande explosão de uso, vem quando celulares e automóveis começam a vir de fábrica com a possibilidade de informar sua localização. Associados a sistemas de mapas embarcados, podem informar ao motorista qual caminho tomar para ir virtualmente a qualquer lugar sobre a superfície terrestre (olha o WAZE). Associados a telefones celulares, podem informar a localização do telefone. Associados a sistemas de proteção contra roubo de automóveis, informam a localização em tempo real e assim por diante. A lista de aplicações possíveis parece inesgotável. Seja como for surge uma tendência muito forte que é, ao cadastrar um imóvel, cadastrar também suas coordenadas geográficas.

Todos os sistemas acima se baseiam na triangulação (ou quadrangulação) do receptor em relação a 3 ou 4 (ou mais) satélites em órbita da terra.

O sistema começa pela identificação de diversos pontos fixos sobre a superfície terrestre. Tais pontos contam com estações transmissoras e permitem a cada satélite a recepção de diversos pontos. Medindo o tempo gasto por cada sinal desde a estação terrena até o satélite, e fazendo isso com diversas estações, o satélite conhece sua real posição com precisão de centímetros.

Cada satélite possui um relógio atômico, integrado à rede GPS e que garante precisão de tempo

de 1 bilionésimo de segundo.

Todos os satélites emitem 2 mensagens: A primeira, envia a identificação do satélite, sua posição no espaço e sua hora interna. Uma segunda mensagem é uma sequência de impulsos digitais em um padrão inconfundível.

Tudo funciona como se o receptor também gerasse no mesmo instante de tempo a mesma sequência de impulsos digitais. Da análise dos dois sinais (o gerado pelo satélite e o gerado internamente no receptor) é possível medir o retardo do sinal do satélite. Conhecendo a velocidade da luz (fixa e igual a 299.792.458 m/seg) e conhecendo o tempo gasto pelo sinal do satélite ao receptor, é possível estabelecer a distância entre satélite e receptor.

Daqui, pode-se inferir que o receptor está sobre a esfera de centro x, y, z (as coordenadas no espaço do satélite) e de raio R , que é a distância entre o receptor e o satélite.

Ao fazer o mesmo cálculo com um segundo satélite, obtem-se uma segunda esfera, e o receptor está sobre o círculo de interseção das duas esferas.

Finalmente ao fazer o cálculo com um terceiro satélite, obtém-se uma terceira esfera. O receptor ainda pertence às 3 esferas, e a interseção das 3 apresenta apenas 2 pontos no espaço. Um deles apenas sobre a superfície terrestre, e logo esta é a posição do receptor, com precisão de metros.

Problemas Se os receptores pudessem ter um relógio atômico 3 satélites bastariam. Como um relógio atômico é muito caro e pesado, os receptores usam relógios de quartzo comum (sujeitos a erros de 1 seg/dia).

Então usa-se um quarto satélite para refazer os cálculos e permitir acertar o atraso/adiantamento do relógio local ao relógio da rede.

Se o usuário estiver em movimento, os sinais ainda vêm afetados pelo efeito Doppler, que precisa ser analiticamente corrigido antes de estabelecer a posição do receptor. Idem para os efeitos relativísticos associados.

O sistema americano usa duas radiofrequências: L1 e L2. A L1 é chamada Sinal Civil, embora os militares também a usem. A L2 era de uso exclusivo militar, já que seus códigos (a segunda parte das mensagens) não eram públicos – protegidos por criptografia. Mas hoje ambos estão liberados. Ao usar apenas o sinal L1, receptores conseguem determinar sua posição com erros de 5 a 10 m, causados pela ação da ionosfera. Ao usar ambos, a precisão melhora: são 2 cálculos que devem coincidir. A camada de ar sobre a terra aumenta a densidade da atmosfera, causando refração nos sinais. É como um lápis colocado dentro de um copo de água.

A potência de um satélite é de 500W, o equivalente a 5 lâmpadas de 100W. Apenas para efeito de comparação, o sinal de TV de um satélite é 1 bilhão de vezes mais forte. Este fato permite que interferências (às vezes propositalmente) obscureçam ou até invalidem o sinal dos satélites. Este fato é usado em teatros de guerra, quando as forças americanas introduzem geradores locais de ruído centrados na banda usada pelo sistema. Essa é a razão pela qual embora o sistema seja público, russos, europeus e chineses tratam de ter cada um o seu.

Dois amigos

Vamos simular o cálculo de uma posição desconhecida usando triangulação, mas vamos trabalhar apenas em 2D, que ninguém aqui está a fim de enlouquecer por causa da geometria analítica. ;-)

Suponhamos uma região plana, na qual dois pontos tem coordenadas conhecidas. Você vai colocar 2 amigos, um em cada ponto, todos munidos de relógios precisos.

Você combinou que às 14h00, um dos seus amigos dará um tiro de canhão. Nessa mesma hora, você ligará um cronômetro. Alguns segundos ou minutos depois, você ouvirá o tiro e desligará o cronômetro, obtendo um tempo gasto pelo barulho para percorrer a distância entre o seu primeiro amigo e você. Vamos chamar esta distância de D_1 .

Lembrando que o som se propaga a 0.340 Km/seg, você conseguirá uma medida confiável entre você e seu amigo 1.

O procedimento será repetido às 14h15 com seu amigo número 2. Pelo mesmo mecanismo você obterá a distância entre você e seu amigo 2. Será a distância D_2 .

Sabendo que seu amigo 1, se encontra no centro de uma circunferência de raio D_1 e coordenadas do centro iguais a x_1, y_1 , e seu amigo 2 se encontra

em x_2, y_2 , também no centro de uma circunferência de raio D_2 , e que você se encontra sobre as duas circunferências, pede-se que você calcule a sua posição.

Obs: se você não tiver muita intimidade com a geometria analítica, um bom método é o gráfico. Use um papel milimetrado, disponha os dois pontos, trace duas circunferências de raios D_1 e D_2 e em um dos encontros das duas circunferências você estará.

Obs:

- ★ Este procedimento também pode ser usado para verificar o acerto/erro das contas e a verificação final do ponto.
- ★ Como só usaremos 2 amigos (ao invés de 3), haverá 2 pontos candidatos, ambos na superfície da terra. Responda os 2.
- ★ Um terceiro amigo eliminaria a ambigüidade.

Exemplo Seja um caso em que $x_1 = 77Km$, $y_1 = 40Km$. $x_2 = 120Km$, $y_2 = 38Km$. Os tempos são: $t_1 = 135.29seg$ e $t_2 = 100.00seg$.

Obtem-se $d_1 = 46Km$ e $d_2 = 34Km$. A fórmula da circunferência 1 é $x^2 + y^2 - 154x - 80y + 5413 = 0$

A fórmula da circunferência 2 é $x^2 + y^2 - 240x - 76y + 14688 = 0$

Subtraindo uma equação da outra, fica-se com $86x - 4y - 9275 = 0$ e $y = 21.5x - 2318.75$.

Substituindo o y obtido em qualquer uma das equações, chega-se ao valor correto do observador que é $x_o = 111Km$, $y_o = 71Km$, ou o outro valor possível que é $x_o = 108Km$, $y_o = 6Km$.

Para você fazer

Dados:

- ★ Seu amigo 1 está em $x_1 = 59 Km$ e $y_1 = 58 Km$.
- ★ Seu amigo 2 está em $x_2 = 102 Km$ e $y_2 = 59 Km$.
- ★ O tiro do amigo 1 demorou $t_1 = 114.71$ segundos. O tiro do amigo 2 demorou $t_2 = 85.29$ segundos.

Como fazer

Para entender o funcionamento do sistema, é necessário relembrar um pouco de física e de geometria analítica.

Da física: distância = velocidade × tempo. Sabendo que a velocidade do som é 0.340 Km/seg, obtém-se as duas distâncias.

Da geometria analítica, uma circunferência no plano é perfeitamente determinada se soubermos as coordenadas α, β do seu centro, bem como o raio dele, usualmente representada por R . Da geometria analítica, e do Teorema de Pitágoras, a fórmula é

(x - α)² + (y - β)² = R²

Do problema, são conhecidos $\alpha_1, \alpha_2, \beta_1, \beta_2, D_1$ e D_2 . Obtem-se agora duas fórmulas em x^2, x, y^2 e y . Operando com elas obtem-se dois pares de x, y que são os pontos candidatos. Os dois pontos candidatos a serem sua localização são:

x,y	x,y

Mais Um

- ★ Seu amigo 1 está em $x_1 = 57 Km$ e $y_1 = 59 Km$.
- ★ Seu amigo 2 está em $x_2 = 95 Km$ e $y_2 = 64 Km$.
- ★ O tiro do amigo 1 demorou $t_1 = 105.88$ segundos. O tiro do amigo 2 demorou $t_2 = 61.76$ segundos.

x,y	x,y



Quais os nomes válidos? Na lista de nomes a seguir, informe se cada nome é válido ou não.

nome	válido	inválido
R33		
33R		
SALDOBANCARIO		
JKGHHGGJHG		
OBA AQUI		
A+B		
ZERO/DOIS		

Tipos de variáveis Para representar as variáveis a seguir, qual o(s) tipo(s) mais adequado ? (Se houver mais de um tipo, indique com “1” o melhor, com “2” o seguinte e assim por diante).

nome do campo	int	real	lógico	carac	cadeia
nome					
salário					
número de filhos					
o sujeito tem filhos ?					
idade					
ano de nascimento					
categoria de motorista					
nome de uma vitamina					

Resolva os exercícios a seguir Cada um dos exercícios a seguir apresenta uma resposta numérica única. Descubra-a e coloque o seu valor no quadrado duplo que segue a definição.

1. Em um grande restaurante industrial existem ajudantes de cozinha, que tem as seguintes habilidades:

habilidade	quantos sabem fazer
descascar batatas	7
cortar rabanetes	8
picar cebolas	11
c. rabanete e p. cebolas	5
c. rabanete e d. batatas	6
d. batatas e p. cebolas	12
as 3 coisas	9
nada	3

Pergunta-se, qual a porcentagem da equipe que é capaz de picar cebolas ?

2. Três torneiras podem encher um certo tanque de volume V. Se são totalmente abertas as torneiras 1 e 2, a demora para encher é de 12 horas. Se são abertas a 1 e a 3, a demora é 15 horas. Se as abertas são a 2 e a 3, a demora é 20 horas. Pergunta-se: se as três forem abertas, quantas horas demora o enchimento ?

3. A soma de dois números primos a e b é 34 e a soma dos primos a e c é 33. Quanto vale $a + b + c$? (OBM 2004)

4. Um time de futebol ganhou 8 jogos a mais do que perdeu e empatou 3 jogos menos do que ganhou em 31 partidas jogadas. Quantas partidas o time ganhou ? (OBM 2006)

5. Uma processadora de alimentos desenvolveu uma marca de salada de fruta enlatada de muito sucesso. A fórmula desta salada, envolve as seguintes frutas e ingredientes nas seguintes proporções: 1 Kg de banana; 0,5 Kg de abacaxi; 2 Kg de laranja e 250 gramas de açúcar.

Suponha que em um determinado dia, existem disponíveis as seguintes quantidades no depósito: 100 Kg de açúcar, 300 Kg de abacaxi, 800 Kg de bananas e 1 tonelada de laranja.

Pergunta-se quantos quilos de salada de fruta poderão ser produzidos preservando as proporções da fórmula original ?

6. Na oficina mecânica do Zé Tião, nesta semana foram reparados 40 veículos, entre motos e carros. Como curiosidade, o auxiliar do Zé Tião contou que o número total de rodas dos veículos reparados foi 100 rodas. Pergunta-se quantas motos foram reparadas ?

7. Três anos atrás, a população de Pirajussaraí era igual à população que Tucupira tem hoje. De lá para cá, a população de Pirajussaraí não mudou, mas a população de Tucupira cresceu 50%. Atualmente as duas cidades somam 9000 habitantes. Há 3 anos, qual era a soma das duas populações ? (OBM 2005)

8. As letras O , B e M representam números inteiros. Se $O \times B \times M = 240$, $(O \times B) + M = 46$ e $O + (B \times M) = 64$, quanto vale $O + B + M$? (OBM 2005)

9. O tanque do carro de Esmeralda, com capacidade de 60 litros, contém uma mistura de 20% de álcool e 80% de gasolina, ocupando a metade de sua capacidade. Esmeralda pediu para colocar álcool no tanque até que a mistura ficasse com quantidades iguais de álcool e gasolina. Quantos litros de álcool devem ser colocados ? (OBM 2005)

10. O correio de um determinado país só aceita transportar pacotes com peso inferior a 3 Kg e comprimento master menor que 250 centímetros. O comprimento master é o maior barbante que dá a volta completa ao pacote. Supondo um pacote de 2,85 Kg e cujas dimensões em metros são: 0,3 x 0,7 x 1,1 m, pergunta-se Este pacote pode (responda 1) ou não pode (responda 0) ser mandado pelo correio ?



- 1 - /

Alternativas da folha 008

- 1 Uma processadora de alimentos desenvolveu uma marca de salada de fruta enlatada de muito sucesso. A fórmula desta salada, envolve as seguintes frutas e ingredientes nas seguintes proporções: 1 Kg de banana; 0,5 Kg de abacaxi; 2 Kg de laranja e 250 gramas de açúcar.
- Suponha que em um determinado dia, existem disponíveis as seguintes quantidades no depósito: 100 Kg de açúcar, 300 Kg de abacaxi, 800 Kg de bananas e 1 tonelada de laranja.
- Pergunta-se quantos quilos de salada de fruta poderão ser produzidos preservando as proporções da fórmula original ?
- 1500
- 2 O correio de um determinado país só aceita transportar pacotes com peso inferior a 3 Kg e comprimento master menor que 250 centímetros. O comprimento master é o maior barbante que dá a volta completa ao pacote. Supondo um pacote de 2,85 Kg e cujas dimensões em metros são: 0,3 x 0,7 x 1,1 m, pergunta-se Este pacote pode (responda 1) ou não pode (responda 0) ser mandado pelo correio ?
- 0
- 3 56 biscoitos polivalentes devem servir de comida para dez animais, sem que sobre nada. Cada animal é um cachorro ou um gato. Cada cachorro recebe seis biscoitos e cada gato recebe 5 biscoitos. Pergunta-se quantos gatos há ?
- 4
- 4 Na oficina mecânica do Zé Tião, nesta semana foram reparados 40 veículos, entre motos e carros. Como curiosidade, o auxiliar do Zé Tião contou que o número total de rodas dos veículos reparados foi 100 rodas. Pergunta-se quantas motos foram reparadas ?
- 30
- 5 Antônio e Benedita amassaram 400 batatas. Antônio amassava 3 batatas a cada minuto e Benedita conseguia amassar apenas 2 batatas por minuto. Benedita trabalhou 25 minutos a mais que Antônio. Pergunta-se, quantos minutos Antônio trabalhou ?
- 70
- 6 Um certo espetáculo musical no Japão, tem um custo para entrar de 3500 ienes para adultos e 1200 ienes para estudantes. No sábado passado havia 608 pessoas, e a renda da noite foi de 1.707.100 ienes. Quantos estudantes havia ?
- 183
- 7 Três torneiras podem encher um certo tanque de volume V. Se são totalmente abertas as torneiras 1 e 2, a demora para encher é de 12 horas. Se são abertas a 1 e a 3, a demora é 15 horas. Se as abertas são a 2 e a 3, a demora é 20 horas. Pergunta-se: se as três forem abertas, quantas horas demora o enchimento ?
- 10
- 8 Em um grande restaurante industrial existem ajudantes de cozinha, que tem as seguintes habilidades:
- | habilidade | quantos sabem fazer |
|--------------------------|---------------------|
| descascar batatas | 7 |
| cortar rabanetes | 8 |
| picar cebolas | 11 |
| c. rabanete e p. cebolas | 5 |
| c. rabanete e d. batatas | 6 |
| d. batatas e p. cebolas | 12 |
| as 3 coisas | 9 |
| nada | 3 |
- Pergunta-se, qual a percentagem da equipe que é capaz de picar cebolas ?
- 60.6
- 9 O perímetro de determinado retângulo é 16 vezes maior do que o menor lado do retângulo. O comprimento do maior lado é 12 centímetros maior do que o menor lado. Qual o comprimento do maior lado, em centímetros?
- 14
- 10 Na floresta, o lobo dorme quando a coruja está acordada e está acordado quando a coruja dorme. O lobo dorme tanto numa semana quanto a coruja dorme num dia. Quantas horas dorme o lobo por dia ? (XXIV OPM, 2005)
- 3
- 11 Em um tanque há 4000 bolinhas de pingue-pongue. Um menino começou a retirar as bolinhas, uma por uma, com velocidade constante quando eram 10h. Após 6 horas, há no tanque 3520 bolinhas. Se o menino continuasse no mesmo ritmo, a que horas do dia seguinte o tanque ficaria com 2000 bolinhas ? (OBM 2006)
- 11
- 12 Um time de futebol ganhou 8 jogos a mais do que perdeu e empatou 3 jogos menos do que ganhou em 31 partidas jogadas. Quantas partidas o time ganhou ? (OBM 2006)

- 14
- 13 O tanque do carro de Esmeralda, com capacidade de 60 litros, contém uma mistura de 20% de álcool e 80% de gasolina, ocupando a metade de sua capacidade. Esmeralda pediu para colocar álcool no tanque até que a mistura ficasse com quantidades iguais de álcool e gasolina. Quantos litros de álcool devem ser colocados ? (OBM 2005)
- 18
- 14 Três anos atrás, a população de Pirajussaraí era igual à população que Tucupira tem hoje. De lá para cá, a população de Pirajussaraí não mudou, mas a população de Tucupira cresceu 50%. Atualmente as duas cidades somam 9000 habitantes. Há 3 anos, qual era a soma das duas populações ? (OBM 2005)
- 7500
- 15 As letras O , B e M representam números inteiros. Se $O \times B \times M = 240$, $(O \times B) + M = 46$ e $O + (B \times M) = 64$, quanto vale $O + B + M$? (OBM 2005)
- 20
- 16 A soma de dois números primos a e b é 34 e a soma dos primos a e c é 33. Quanto vale $a + b + c$? (OBM 2004)
- 36
- 17 Na multiplicação a seguir, a , b e c são algarismos:

$$\begin{array}{r} 1\ a\ b \\ \ b\ 3\ x \\ \hline \ * \ * \ * \\ \ * \ * \ * \\ \hline 1\ c\ c\ 0\ 1 \end{array}$$

Qual o valor de $a + b + c$

10

Qual o número que falta? Neste folha de exercícios, serão apre-
sentadas diversas seqüências de 11 números. Os 3 números iniciais são alea-
tórios. Os próximos, a partir do quarto e até o final seguem uma determinada
lei de formação. Um dos números do fim da seqüência é representado por

Para se certificar, aplique a lei de formação a vários números e veja se
não há nenhuma discrepância. Feito isso, a segunda coisa a fazer é descobrir
qual o número que falta. Finalmente, em terceiro, deve-se somar os números
achados de três em três, respondendo a soma no local apropriado.

Exemplo Veja-se alguns casos

N.	1	2	3	4	5	6	7	8	9	10	11	resposta	some 3 em 3
1	8	12	21	37	62	98	147	211	292	?	513		=
2	4	7	3	14	24	41	79	?	264	487	895	+	
3	8	2	15	22	42	69	116	?	311	506	822	+	
4	5	7	12	19	31	50	81	131	212	?	555		=
5	4	8	17	33	58	94	143	207	?	388	509	+	
6	9	13	22	38	63	99	148	212	293	?	514	+	
7	1	4	5	9	14	23	37	60	?	157	254		=
8	3	7	16	32	57	93	142	206	?	387	508	+	
9	6	9	24	57	138	333	804	1941	?	11313	27312	+	
10	9	5	14	19	33	52	85	137	222	?	581		=
11	8	1	7	16	24	47	87	158	292	?	987	+	
12	5	9	1	15	25	41	81	147	?	497	913	+	
13	1	9	10	19	29	48	77	125	?	327	529		=
14	5	4	9	13	22	35	57	?	149	241	390	+	
15	9	13	22	38	63	99	148	212	293	?	514	+	

Os resultados: 392 144 190 343 288 393 97 287 4686 359 537 269 202 92 393 que somados são: 726 1024 5070 1165 687

Para você fazer

N.	1	2	3	4	5	6	7	8	9	10	11	resposta	some 3 em 3
1	3	7	9	19	35	63	117	215	?	727	1337		=
2	3	3	4	10	17	31	58	106	195	?	660	+	
3	2	4	7	18	43	104	251	?	1463	3532	8527	+	
4	7	7	9	25	43	77	129	215	?	577	939		=
5	3	5	7	15	27	49	91	167	307	?	1039	+	
6	1	4	7	23	48	84	133	197	278	?	499	+	
7	4	9	9	24	39	69	114	189	?	504	819		=
8	2	4	7	23	48	84	133	197	?	378	499	+	
9	6	6	9	16	26	43	70	114	185	?	486	+	
10	2	6	6	22	47	83	132	?	277	377	498		=
11	6	7	9	25	50	86	135	199	?	380	501	+	
12	4	8	9	25	50	86	135	199	280	?	501	+	
13	7	8	9	17	26	43	69	112	?	293	474		=
14	6	7	8	22	38	82	158	322	638	?	2558	+	
15	1	1	9	11	29	51	109	211	429	?	1709	+	
16	1	5	6	12	23	41	76	140	?	473	870		=
17	1	6	7	20	47	114	275	664	?	3870	9343	+	
18	3	3	8	13	23	38	63	103	?	273	443	+	
19	3	4	8	12	20	32	52	84	?	220	356		=
20	1	4	4	20	45	81	130	194	?	375	496	+	
21	3	4	5	21	46	82	131	?	276	376	497	+	
22	2	5	7	23	48	84	133	?	278	378	499		=
23	4	7	8	16	25	42	68	111	?	292	473	+	
24	5	5	5	15	25	45	85	155	285	?	965	+	
25	3	6	6	18	30	66	126	?	510	1026	2046		=
26	4	4	9	13	22	35	57	?	149	241	390	+	
27	1	3	3	19	44	80	129	?	274	374	495	+	
28	4	7	7	21	35	77	147	301	595	?	2387		=
29	4	6	9	21	39	81	159	321	639	?	2559	+	
30	4	5	8	18	34	70	138	278	554	?	2218	+	

O talento de D. Mariquinhas

A Dona Mariquinhas é uma talentosa artista plástica. Já andou pela pintura, escultura, mas atualmente tem se dedicado aos mosaicos. Suas paisagens e retratos feitos com cacos de azulejo ficaram famosos. No momento, ela trabalha uma série de mosaicos feitos com pastilhas quadradas, aos quais ela deu o nome de "Poemas Matemáticos".

Ela começou achando 128 cores distintas de pastilhas (numeradas de 0 a 127). As diferenças são sutis, mas o resultado final é cheio de nuances de cor. Ela distribui as cores de uma maneira aparentemente aleatória. Mas, antes de terminar, ela escolhe 10 pastilhas, que ela identifica com as letras A, B, C, ..., J.

Eis como se parece um mosaico feito por ela, neste ponto

62	115	116	8	115	65	66	41	126	63	34
12	A	10	64	F	36	117	D	59	C	7
97	98	106	16	3	88	111	80	94	93	127
113	30	B	45	E	76	108	53	107	35	53
69	60	37	23	20	73	102	5	68	64	122

Para determinar as cores que deverão ser usadas no lugar ocupado pelas letras, a Dona Mariquinhas, inventou uma série de 7 regras. Cada regra, quando aplicada à letra, pode responder SIM ou NÃO. Antes de aplicar cada regra é necessário localizar e conhecer os 4 vizinhos diretos da letra em questão. São eles: "C" (o vizinho de cima), "B" (o de baixo), "E" (esquerda) e "D" (direita).

- Um azulejo tem 1 se $C \geq 20 \wedge B \leq 45$ e 0 senão.
- Um azulejo é 2 se e somente se $D + E > 100$ e 0 senão.
- Um azulejo é 4 se (C é par e B é ímpar) OU (se B é par e E é ímpar) e 0 senão.
- Um azulejo é 8 se $C < ((B + E + D)/3) \wedge B > 45$, e 0 senão.
- Um azulejo é 16 se B, E, C, D estão em ordem crescente e 0 senão. (Obs: 3,3,6,7 está em ordem crescente).
- Um azulejo é 32 se $(B + E) \geq (C + D)$ e 0 senão
- Um azulejo é 64 se $|E - D|$ e $|B - C|$ valem ambos mais de 25 e 0 senão.

Você deve analisar o valor da cor que deverá ter o azulejo que ocupará o lugar de cada uma das letras, verificando sua vizinhança e somando os resultados parciais apontados por cada uma das regras.

Após aplicar as 7 regras e eventualmente somar os valores de cada regra, você obterá a cor do azulejo que deverá ser colocado onde está cada letra.

Finalmente, para ter maior conhecimento e controle sobre o mosaico, a Dona Mariquinhas estabeleceu 3 tipos de cores:

cor quente é aquela cujo valor numérico é maior que 70. As cores frias são aquelas que valem de 0 a 70.

cor forte é aquela cujo número é par. Cor fraca é aquela cujo número é ímpar.

cor composta é aquela que é matizada por mais do que duas regras. A cor básica é aquela que é atingida por zero, uma ou duas regras.

O objetivo do exercício é calcular quantas cores são quentes, fortes e compostas.

Um exemplo

Seja um mosaico assim constituído:

68	25	65	85	68	65	1	17	79	35	107
47	**B	15	**D	96	126	34	1	24	63	65
110	40	110	45	93	126	86	10	116	113	6
45	**F	77	123	17	78	26	**I	9	83	40
48	127	34	96	93	27	116	81	73	68	51
83	**E	36	**H	121	30	105	121	13	26	87
67	100	49	57	127	62	92	97	11	47	13
93	**G	9	15	55	68	18	8	9	55	9
123	47	73	120	8	52	107	72	14	94	28
69	33	67	**J	68	**C	61	10	72	**A	90
4	110	78	74	49	48	72	8	89	50	121

Neste mosaico, as cores dos azulejos identificados com letras são

B =	37	3	(regras 1, 3, 6)
D =	67	3	(regras 1, 2, 7)
F =	110	5	(regras 2, 3, 4, 6, 7)
I =	44	3	(regras 3, 4, 6)
E =	102	4	(regras 2, 3, 6, 7)
H =	70	3	(regras 2, 3, 7)
G =	102	4	(regras 2, 3, 6, 7)
J =	6	2	(regras 2, 3)
C =	42	3	(regras 2, 4, 6)
A =	2	1	(regra 2)

Aplicando as regras das cores quente, forte e compostas a este resultados, as contagens são

3 8 8

Para você fazer

- Verifique as cores para o seguinte mosaico:

51	104	98	60	78	15	4	17	96	49	106
82	59	55	106	5	104	53	**G	55	97	121
84	83	1	64	5	60	3	38	58	90	57
97	95	105	**J	5	**D	38	11	91	90	37
104	45	56	103	122	5	46	78	73	74	41
92	62	110	**H	117	**I	14	**F	38	77	40
10	100	81	93	46	50	114	122	55	40	16
20	66	55	**A	93	8	48	**B	16	63	125
49	94	11	8	26	43	76	80	24	63	80
127	73	20	94	51	118	101	**C	43	**E	5
69	8	28	39	106	68	60	31	100	91	41

Para ajudar eis o gabarito para preencher:										
L	1	2	3	4	5	6	7	Σ		
Σ	1	2	4	8	16	32	64			
A										
B										
C										
D										
E										
F										
G										
H										
I										
J										

Depois de preencher o gabarito, você deve fazer a contagem sobre as 10 letras, buscando descobrir destas, quantas são

quentes	fortes	compostas

Mais um...

- Verifique as cores para o seguinte mosaico:

79	70	113	103	80	110	59	25	68	85	103
20	**J	79	**I	57	22	1	88	48	57	94
79	80	107	91	14	109	30	125	24	80	57
22	**A	6	**C	9	88	35	49	10	49	13
74	104	72	23	13	77	4	86	125	35	15
118	105	17	79	88	40	11	**E	92	**D	46
35	68	121	13	81	81	63	84	86	6	28
16	12	29	**H	50	97	111	22	69	**G	69
58	101	84	78	95	33	99	2	76	82	40
20	**F	100	15	10	6	16	35	86	**B	19
123	102	19	126	42	43	75	29	5	92	1

Para ajudar eis o gabarito para preencher:										
L	1	2	3	4	5	6	7	Σ		
Σ	1	2	4	8	16	32	64			
A										
B										
C										
D										
E										
F										
G										
H										
I										
J										

Depois de preencher o gabarito, você deve fazer a contagem sobre as 10 letras, buscando descobrir destas, quantas são

quentes	fortes	compostas

- Verifique as cores para o seguinte mosaico:

97	82	102	54	101	29	64	92	112	89	24
84	98	115	13	11	86	109	12	86	**G	88
22	67	68	41	62	61	113	8	89	81	94
115	117	98	**E	45	92	101	5	43	71	26
93	15	84	120	30	94	44	67	69	125	70
107	**H	100	115	59	8	4	**C	66	**F	47
113	79	37	67	6	6	106	3	112	41	84
60	**B	99	**J	74	**D	68	**A	12	2	114
84	44	85	25	85	61	67	15	18	80	65
94	12	19	56	87	**I	124	18	50	104	109
72	109	111	25	26	56	111	6	22	105	110

Para ajudar eis o gabarito para preencher:										
L	1	2	3	4	5	6	7	Σ		
Σ	1	2	4	8	16	32	64			
A										
B										
C										
D										
E										
F										
G										
H										
I										
J										

Depois de preencher o gabarito, você deve fazer a contagem sobre as 10 letras, buscando descobrir destas, quantas são

quentes	fortes	compostas



- 1 - /

O diâmetro da Terra foi medido pela primeira vez por Eratóstenes. Este feito foi obtido sem que ele sáisse da biblioteca em que trabalhava, localizada na cidade de Alexandria, no norte do Egito, entre 276 a.C e 196 a.C. Eratóstenes era o responsável pela biblioteca do museu, tinha muitos interesses sobre as ciências e ouviu comentários de viajantes que tinham estado na cidade de Siene, onde está localizada hoje a represa de Assuam, que exatamente ao meio dia do primeiro dia de verão (21 de junho), o Sol se colocava sobre as cabeças das pessoas, dirigindo os raios de uma forma vertical. Olhando-se um poço profundo, podia-se ver o reflexo do Sol no fundo do poço. Eratóstenes observou que neste mesmo dia e hora em Alexandria havia uma sombra provocada por raios solares que não estavam sendo projetados verticalmente, mas formando um ângulo um pouquinho maior que 7° em relação à cidade de Siene que ficava 800Km mais ao Sul.

Partindo destas informações e levando em consideração que muitas medidas da época eram imprecisas, Eratóstenes calculou a circunferência da Terra fazendo a seguinte análise:

Se uma circunferência tem 360° e um deslocamento angular de 7° corresponde aproximadamente a 1/50 de um círculo e esta medida em graus equivale a 800Km, então a volta completa deverá corresponder à circunferência da Terra, que deverá ser aproximadamente $800 \times 50\text{Km} = 40.000\text{Km}$.

Atualmente, a circunferência da Terra mede 39.830 Km e observamos que a medida obtida para a época era excelente.

Veja-se a propósito uma notícia divulgada em 5 de julho de 2008, dizendo

O diâmetro da Terra é 5 milímetros menor do que se acreditava até o momento, revela uma pesquisa da Universidade de Bonn (Alemanha) publicada nesta quinta-feira.

Um grupo de pesquisadores do Instituto Geodésico da universidade provou que o diâmetro da Terra, de 12.756,274 quilômetros, é na realidade 5 milímetros menor que o estabelecido na última medição, realizada há cinco anos, disse à AFP Axel Nothnagel, que dirige a equipe de cientistas.

Apesar da diferença parecer mínima para o grande público, ela é muito importante para o estudo das mudanças climáticas. “Parece uma diferença insignificante, mas é fundamental para o posicionamento dos satélites capazes de medir a elevação do nível dos oceanos”, explicou Nothnagel.

Os pesquisadores também descobriram que a Europa se separa 18 milímetros por ano do continente norte-americano.

Relembrando a geometria clássica, sabe-se que as medidas esféricas são:

circunferência é um comprimento (um barbante) cuja extensão é de $2 \times \pi \times R$, onde R é o raio da circunferência.

círculo é uma área cujo valor é de $\pi \times R^2$, onde R é o raio do círculo.

esfera é um volume, cujo valor é de $\frac{4}{3} \times \pi \times R^3$, onde R é o raio da esfera.

Além disso as coordenadas geográficas de um ponto sobre o planeta Terra, são dadas por duas medidas:

latitude é o afastamento em relação ao equador: varia de 0° (sobre a linha do equador), até 90° N (no polo norte) ou 90° S (no polo sul). O Centro Cívico em Curitiba, para termos um exemplo, está a 25° 25'S.

longitude mede a distância lateral em relação ao Meridiano de Greenwich (um subúrbio de Londres, onde está o observatório de Greenwich). As medidas variam entre 0 e 180 W (à esquerda de que olha o mapa de frente) e 0 e 180 E (à direita). O Centro Cívico em Curitiba está a 49° 16' W.

Finalmente, na trigonometria podem ser necessários achar o seno e o cosseno de um ângulo. Dado um triângulo retângulo, valem as seguintes regras:

seno de α é a relação entre o cateto oposto (a α) e a hipotenusa do triângulo. Vale o macete: seno=cateto SEparado ÷ hipotenusa.

cosseno de α é a relação entre o cateto adjacente (a α) e a hipotenusa do triângulo. Vale o macete: cosseno=cateto COLado ÷ hipotenusa.

Pitágoras Dado um triângulo retângulo, de catetos a e b e hipotenusa c , sempre vale $a^2 + b^2 = c^2$.

raio é a metade do diâmetro.

Para este exercício, suporemos a medida do diâmetro da Terra valendo 12.756,274 Km.

Finalmente, é necessário saber converter graus, minutos e segundos em quilômetros (na longitude, medidos sobre o equador e na latitude, em qualquer região da Terra).

Raio da terra		Km
Circunferência no equador		Km
Dividindo por 360	1°=	Km
Dividindo por 60	1'=	Km
Dividindo por 60	1''=	Km

Exercício 1: distância No exercício 1, serão dados 2 pontos, através de suas coordenadas geográficas. A pergunta é sobre a distância linear que os separa.

Por exemplo, seja o ponto A com coordenadas 2° 10' 17"N e 5° 7' 17" W. O ponto B tem 3° 5' 10"S e 4° 37' 42" W.

Neste exercício deve-se desprezar a variação que o afastamento da linha do equador introduz nas longitudes. Por exemplo, o comprimento de um gomo de laranja é maior no centro do gomo do que nos seus extremos. Considerar esta variação deixaria os cálculos significativamente mais complexos, e como as latitudes são pequenas (menores que 10°), a variação pode ser desprezada. A pergunta é:

Qual a distância entre os dois pontos ?
Transformando 2,10,17,N em segundos = 7817 seg
Transformando 5,7,17,W em segundos = 18437 seg
Transformando 3,5,10,S em segundos = 11110 seg
Transformando 4,37,42,W em segundos = 16662 seg
Como N ≠ S, soma-se 7817+11110 = 18927 seg
Como W = W, subtrai-se o maior do menor = 1775 seg
agora, um cateto vale = 585.2617455 Km
e o outro, vale = 54.88664861 Km
Achando a hipotenusa via Pitágoras, obtem-se 587.829784 Km que é a resposta procurada.

Exercício 2: comprimento O iate Duncan navegava ao largo do golfo do rio Clyde, na Escócia quando os seus tripulantes se depararam com um tubarão. Após uma luta entre os homens e o enorme peixe, os homens acabam por levar a melhor. A maior surpresa é que o tubarão guardava dentro de si um enigma... uma garrafa de vidro com três papeis em três línguas diferentes (inglês, francês e alemão), porém grande parte das palavras eram imperceptíveis. As poucas palavras legíveis permitiam deduzir que se tratava do naufrágio do Britannia, um navio comandado por Harry Grant, e que se encontravam a 37 graus de latitude no Hemisfério Austral. De imediato, o lorde Edward Glenarvan, proprietário do Duncan, a sua mulher Helena, o seu primo major Mac Nabbs e o capitão John Mangles tentam decifrar todas as pistas contidas nos documentos para resgatar o capitão do Britannia e os seus dois companheiros.

Mais tarde Glenarvan e a sua mulher publicam num jornal um anúncio dizendo que possuíam informações sobre o possível paradeiro de Grant. Aparecem na mansão dos Glenarvan os corajosos filhos do capitão Grant, que se julgavam orfãos

de pai e mãe. Vendo as pobres crianças (16 e 12 anos) desamparadas, Glenarvan, decide organizar uma expedição em busca do capitão Grant. Junta-se, também à expedição o distraído e sábio professor Paganell que embarcará no Duncan “por distração” e, entre outros acontecimentos fora do normal, aprende a falar português julgando aprender espanhol. Estão feitos os preparativos para uma ousada expedição ao longo do paralelo 37 em busca do capitão Grant.

No livro “Os filhos do capitão Grant”, escrito em meados do século XIX, Julio Verne conta a história do naufrágio do capitão. Depois dos acontecimentos acima descritos, e vários anos (e cerca de 400 páginas) depois, o capitão é encontrado, Mary Grant apaixonou-se e casa com John Mangles que vem a ser o capitão do barco de Glenarvan, e todos vivem felizes para sempre. Bem resumido, no bilhete está escrita a latitude (37° S), mas a longitude está apagada pela água do mar. Então eles sabem uma das coordenadas, mas não sabem a outra. Portanto, a coisa a fazer é dar a volta completa ao planeta pela latitude 37° S. A expedição de resgate atravessa o planeta começando no Chile, atravessando a Argentina, depois o atlântico, depois o pacífico, que era onde ele estava.

O exercício simula este caso e diante do bilhete do capitão informando que a latitude é algum valor, qual será o comprimento do trajeto que a expedição terá que percorrer (na hipótese de não achar nada).

No exemplo, dado um ângulo de 42 graus, o ângulo em radianos = 0.7330382858 rd com isto, o cateto horizontal = 4267.806678 Km e a circunferência pedida = 26815.42021 Km

🔗 Para você fazer

Exercício 1:

Ponto A =

2 graus 12 min 9 seg N
5 graus 48 min 35 seg E

Ponto B =

0 graus 55 min 57 seg S
15 graus 40 min 41 seg W

Exercício 2:

0 ângulo escrito no bilhete para a latitude foi de
62 graus

Respostas Responda SEM decimais

1. Distância(A,B) em Km	2. Circunferência em Km

Observação Lendo o livro do Art of Computer Programming for SE da UVA, achou-se uma fórmula para cálculo da distância esférica que é:

$$dist_{esfer}(lat_a, lon_a, lat_b, lon_b) = \arccos(\cos(sen(lat_a) \times sen(lat_b) + \cos(lat_a) \times \cos(lat_b)) \times \cos(lon_a - lon_b) + (\cos(lat_a) \times \cos(lat_b) \times sen(lon_a - lon_b)) \times r)$$

Eu usei esta fórmula para calcular o exemplo dado na folha e obtive 587.8282021 (muito próximo ao que se calculou na folha que é de 587.829784).

Para usar a fórmula acima precisei achar as latitudes e longitudes dos pontos A e B em radianos, transformando [-180..180] em [-π.. +π]

Vamos lá: ponto A: lat 2 10' 17"N ou +0.03789788 rd; lon 5 7' 17"W ou -0.08938509 rd; B: lat 3 5' 10"S ou -0.05386279 rd; lon 4 37' 42"W ou -0.08077965 rd.



- 1 - /

Exercícios

Note que a operação $x \bmod y$, quando $y = 0$, matematicamente não está definida. Entretanto, para esta folha, combinar-se-á que $x \bmod y = x$ quando $y = 0$. Da mesma maneira, $x \div 0$ não está definido. O caso particular, válido apenas aqui é $0 \div 0$ que aqui dará 1.

Outra coisa a conhecer são as funções **chão** ($\lfloor$) e **teto** ($\lceil$). As funções e os símbolos foram propostos pelo matemático canadense Ken Iverson em 1960. As definições são:

função	definição	exemplos
chão($\lfloor$)	o próximo inteiro $\leq$ número dado	$\lfloor 3.5 \rfloor = 3$; $\lfloor 5 \rfloor = 5$; $\lfloor -3.5 \rfloor = -4$
teto($\lceil$)	o próximo inteiro $\geq$ número dado	$\lceil 3.5 \rceil = 4$; $\lceil 5 \rceil = 5$; $\lceil -3.5 \rceil = -3$

Exercício 1 Resolva as 6 linhas abaixo e some (aritmeticamente) os resultados encontrados, lançando o total no quadro abaixo

- 1 $((18 \bmod 7) \bmod 3) + (((9 - 5) - (3 + 3)) + (5 - 4))$
- 2 $((((26 \div 3) \div 1) \div (21 \div 4)) + (((7 + 4) - (8 + 5)) + (2 - 9)))$
- 3 $((((7 - 3) + (6 \times 5)) + (5 + 5)) - (8 + 9))$
- 4 $((19 \bmod 10) \bmod 4) + (((3 + 6) - (4 + 3)) + (6 + 6))$
- 5 $((25 \bmod 6) \bmod 4) - (((7 + 8) \times (6 - 3)) - (\lfloor 9 \div 9 \rfloor))$
- 6 $((2 - 9) \times (\lfloor 4 \div 4 \rfloor)) + (9 + 4) - (3 + 4)$

soma dos valores

Exercício 2 Resolva as 6 linhas abaixo e some (aritmeticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((14 \div 10) \bmod 4) > (((7 - 8) - (3 + 4)) - (8 + 2))$
- 2 $(17 \div 2) < (((3 + 5) - (4 - 8)) - (8 + 6))$
- 4 $(.F. \wedge .F.) \vee ((.V. \vee .F.) \wedge .F.)$
- 8 $(11 \div 4) < (((4 - 3) + (9 - 6)) + (7 + 7))$
- 16 $(28 \div 10) > (((8 + 9) + (4 + 4)) + (3 + 6))$
- 32 $(.F. \vee .V.) \vee ((\sim(.F. \vee .V.)) \wedge .F.)$

soma das linhas V

Exercício 3 Resolva as 6 linhas abaixo e some (aritmeticamente) os resultados encontrados, lançando o total no quadro abaixo

- 1 $((((24 \bmod 5) \div 4) \div (28 \bmod 9)) - (((6 + 5) - (3 + 9)) - (2 + 4)))$
- 2 $((14 \div 3) \bmod 5) - (\lfloor (((5 + 2) + (6 + 2)) \div (8 + 6)))$
- 3 $(7 \times 9) + (((2 + 9) \times (2 - 3)) + (8 - 4))$
- 4 $((29 \div 1) \div 4) + (((8 + 3) - (6 - 5)) \times (8 + 9))$
- 5 $((28 \div 5) \div 5) \div (11 \div 10) + (((3 + 5) + (9 + 5)) - (8 + 2))$
- 6 $(8 - 3) + (((5 + 7) + (3 - 2)) \times (4 - 2))$

soma dos valores

Exercício 4 Resolva as 6 linhas abaixo e some (aritmeticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((24 \bmod 10) \div 3) < (((5 + 3) \times (5 \times 2)) + (8 + 3))$
- 2 $(24 \div 6) < (((3 + 4) \times (7 - 8)) + (3 + 6))$
- 4 $(.V. \vee .F.) \wedge ((.V. \vee .F.) \vee .F.)$
- 8 $(18 \div 5) < (((9 + 4) + (3 - 8)) - (6 + 5))$
- 16 $(28 \bmod 6) > (((5 \times 6) - (5 \times 7)) + (2 - 9))$
- 32 $(.F. \vee .V.) \wedge ((.V. \wedge .V.) \wedge (\sim .F.))$

soma das linhas V

Exercício 5 Resolva as 6 linhas abaixo e some (aritmeticamente) os resultados encontrados, lançando o total no quadro abaixo

- 1 $((15 \bmod 2) \bmod 2) + (((2 + 5) + (7 \times 8)) - (7 - 3))$
- 2 $((23 \div 4) \div 4) - (((5 + 4) + (2 + 7)) + (7 + 5))$
- 3 $((\lfloor ((7 + 2) \div (3 \times 3)) \rfloor) \times (7 + 7)) - (8 - 5)$
- 4 $((18 \bmod 7) \bmod 3) + (((2 + 5) + (7 - 2)) + (7 - 6))$
- 5 $((24 \bmod 5) \div 1) \bmod (12 \bmod 6) + (((4 \times 7) + (4 - 6)) - (8 - 9))$
- 6 $(\lfloor 9 \div 8 \rfloor) + (((5 + 8) - (3 - 4)) + (8 + 7))$

soma dos valores

Exercício 6 Resolva as 6 linhas abaixo e some (aritmeticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((11 \div 9) \bmod 4) < (((9 + 9) \times (3 + 4)) + (8 \times 8))$
- 2 $(27 \bmod 4) < (((5 \times 9) \times (8 - 3)) + (8 - 7))$
- 4 $(.V. \wedge .F.) \wedge ((\sim(.F. \wedge .V.)) \wedge .V.)$
- 8 $(12 \div 7) > (((4 + 8) \times (2 + 5)) \times (8 + 2))$
- 16 $(13 \div 4) > (((9 - 2) - (2 + 2)) + (8 - 3))$
- 32 $(.V. \vee .V.) \vee ((.F. \vee .V.) \vee .F.)$

soma das linhas V

Exercício 7 Resolva as 6 linhas abaixo e some (aritmeticamente) os resultados encontrados, lançando o total no quadro abaixo

- 1 $((12 \div 3) \bmod 3) + (((5 + 8) + (9 + 3)) - (\lfloor 5 \div 2 \rfloor))$
- 2 $((11 \bmod 3) \bmod 6) - (((8 + 5) + (6 + 3)) - (5 + 2))$
- 3 $(7 - 6) + (((7 \times 5) \times (4 + 2)) - (5 + 7))$
- 4 $((13 \div 1) \bmod 3) + (((2 - 8) + (2 - 6)) + (8 - 4))$
- 5 $((20 \div 5) \div 1) - (((6 + 6) - (5 - 9)) \times (6 + 7))$
- 6 $(3 + 7) + (((4 + 2) - (5 + 7)) - (3 - 7))$

soma dos valores

Exercício 8 Resolva as 6 linhas abaixo e some (aritmeticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((24 \bmod 7) \div 2) < (((\lfloor 5 \div 2 \rfloor) + (6 + 6)) - (3 - 9))$
- 2 $(25 \bmod 10) > (((2 - 9) - (4 - 9)) + (6 + 7))$
- 4 $(.V. \wedge .F.) \wedge ((.V. \vee .V.) \wedge .F.)$
- 8 $(11 \bmod 5) < (\lfloor (((8 \times 2) - (4 \times 6)) \div (3 - 4)))$
- 16 $(29 \bmod 3) > (((4 + 3) + (6 - 9)) + (5 + 5))$
- 32 $(.V. \wedge .V.) \wedge ((.F. \vee .V.) \wedge .V.)$

soma das linhas V

Exercício 9 Resolva as 6 linhas abaixo e some (aritmeticamente) os resultados encontrados, lançando o total no quadro abaixo

- 1 $((23 \div 8) \bmod 4) + (((4 + 8) + (5 \times 7)) - (5 - 8))$
- 2 $((13 \div 2) \div 2) - (((8 + 5) + (5 + 6)) + (6 + 4))$
- 3 $((3 + 2) \times (4 + 6)) - (8 + 2) - (6 \times 3)$
- 4 $((12 \div 5) \div 2) + (\lfloor ((6 \times 5) \div (2 - 3)) \rfloor) - (2 - 3)$
- 5 $((12 \div 6) \bmod 3) - (((8 + 2) \times (6 + 2)) + (9 - 6))$
- 6 $(9 - 7) + (\lfloor ((8 + 2) \div (7 - 4)) \rfloor) - (6 + 6)$

soma dos valores

Exercício 10 Resolva as 6 linhas abaixo e some (aritmeticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((21 \bmod 4) \div 1) > (((2 + 3) + (3 + 3)) + (8 - 7))$
- 2 $(12 \bmod 9) > (((5 - 6) + (7 - 5)) + (4 \times 7))$
- 4 $(\sim(.V. \vee .V.)) \wedge ((.F. \wedge .V.) \wedge .V.)$
- 8 $(14 \bmod 10) < (\lfloor ((5 - 4) \div (7 - 8)) \rfloor) + (8 + 4))$
- 16 $(18 \bmod 4) < (((3 - 8) + (\lfloor 2 \div 2 \rfloor)) + (6 - 9))$
- 32 $(.F. \vee .V.) \wedge ((\sim(.F. \vee .F.)) \wedge .V.)$

soma das linhas V



1
- 1

_____ / _____ / _____

Exercícios Note que a operação $x \% y$, quando $y = 0$, matematicamente não está definida. Entretanto, para esta folha, combinar-se-á que $x \% y = x$ quando $y = 0$. Da mesma maneira, $x/0$ não está definido. O caso particular, válido apenas aqui é $0/0$ que aqui dará 1.

Todas as variáveis neste exercício são **inteiras**. Isto significa que $10/6 = 1$ e não 1.666.

Exercício 1 Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- $((13 / 9) \% 5) \% (15 / 3)) - ((9 + 9) + (8 + 8)) - (9 + 8)$
- $((17 / 3) / 1) / (15 / 10)) + (((5 + 3) + (9 - 6)) + (3 + 9))$
- $((7 * 2) - (3 - 2)) + (4 * 3)) - (7 + 9)$
- $((17 / 7) \% 3) + (((9 - 8) - (3 + 2)) + (5 + 5))$
- $((14 \% 4) \% 6) - (((5 - 3) + (3 - 4)) + (9 - 5))$
- $(7 + 2) + (((6 * 2) * (5 - 2)) + (4 - 7))$

soma dos valores

Exercício 2 Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((18 / 2) / 6) < (((6 + 3) * (3 + 9)) + (9 + 7))$
2 $(15 / 1) < (((7 + 5) / (5 + 7))) - (8 + 3))$
4 $(0 \text{ || } 1) \&\& (! (1 \&\& 0)) \&\& 1$
8 $(22 / 9) < (((2 + 2) - (7 - 2)) + (4 + 8))$
16 $(18 \% 10) > (((6 + 4) + (5 + 2)) - (7 + 2))$
32 $(0 \text{ || } 1) \text{ || } ((0 \text{ || } 0) \&\& 1)$

soma das linhas V

Exercício 3 Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- $((16 \% 9) \% 4) \% (15 \% 3)) - (((6 + 5) - (4 + 3)) + (7 + 2))$
- $((22 \% 10) \% 6) \% (12 \% 9)) + (((7 - 4) + (2 + 8)) + (2 * 9))$
- $(6 - 4) + (((8 + 5) - (7 + 7)) * (5 + 6))$
- $((24 \% 10) \% 5) \% (26 \% 7)) - (((3 + 9) + (4 - 7)) + (3 + 2))$
- $((14 / 4) / 1) - (((5 * 2) + (9 + 7)) - (2 + 6))$
- $((9 - 6) + (4 + 5)) + (3 * 7)) - (5 - 2)$

soma dos valores

Exercício 4 Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((13 / 9) \% 2) < (((2 + 3) + (3 + 8)) - ((4/3)))$
2 $(25 / 10) > (((7 / 3)) + (2 * 9)) + (5 + 2))$
4 $(1 \&\& 1) \text{ || } ((1 \text{ || } 1) \text{ || } 1)$
8 $(18 / 10) < (((8 + 7) - (7 + 9)) + (7 + 3))$
16 $(29 \% 5) > (((6 - 9) - (6 + 4)) * (2 + 6))$
32 $(1 \text{ || } 1) \text{ || } ((1 \text{ || } 0) \text{ || } 0)$

soma das linhas V

Exercício 5 Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- $((12 / 3) \% 5) / (16 / 7)) - (((3 - 5) * (4 + 6)) - (9 + 7))$
- $((23 / 1) \% 6) - (((5 + 8) - (2 + 3)) * ((8 / 2)))$
- $((6 + 6) - (7 + 9)) * (3 - 5)) - (8 + 9)$
- $((24 / 6) \% 6) \% (26 / 4)) - (((2 - 8) - (5 + 4)) + (6 + 8))$
- $((23 / 4) \% 6) - (((6 - 3) + (7 + 6)) + (9 * 3))$
- $((6 - 5) * (2 * 2)) - (6 - 7)) - (5 + 8)$

soma dos valores

Exercício 6 Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((15 / 3) / 1) > (((8 + 6) + (8 + 2)) * (2 * 9))$
2 $(27 \% 8) < (((7 - 2) - (9 + 8)) - (4 + 7))$
4 $(1 \text{ || } 0) \text{ || } (! (1 \&\& 0)) \text{ || } 0$
8 $(26 / 10) > (((8 + 6) + (7 + 8)) + (8 * 3))$
16 $(13 / 1) < (((8 + 3) * (7 + 7)) / (3 + 7))$
32 $(1 \&\& 1) \text{ || } ((1 \text{ || } 0) \&\& 0)$

soma das linhas V

Exercício 7 Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- $((17 \% 10) / 3) / (13 \% 6)) - (((9 - 6) + (6 + 8)) * (2 + 3))$
- $((22 / 1) / 2) / (16 / 4)) + (((7 + 4) + (4 + 9)) + (2 + 9))$
- $((6 + 3) - (6 + 5)) + (3 - 7)) - (4 + 6)$
- $((25 \% 9) \% 5) / (20 \% 9)) - (((3 * 9) + (7 + 4)) - (5 - 6))$
- $((25 / 7) \% 6) - (((3 + 8) + (8 + 6)) + (3 + 6))$
- $(4 + 9) + (((4 + 6) - ((7 / 4))) + (9 - 5))$

soma dos valores

Exercício 8 Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((15 / 10) \% 4) > (((3 + 4) + (7 + 5)) - (7 + 8))$
2 $(15 / 7) < (((4 * 6) + (7 - 8)) - ((6 / 4)))$
4 $(0 \&\& 0) \&\& (! (1 \text{ || } 0)) \&\& 1$
8 $(27 / 9) > (((3 - 7) * (9 * 3)) + (9 * 2))$
16 $(22 \% 5) > (((6 - 9) + (8 + 8)) - ((5 / 4)))$
32 $(0 \text{ || } 0) \text{ || } ((1 \text{ || } 0) \&\& 0)$

soma das linhas V

Exercício 9 Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- $((22 / 3) / 6) + (((6 * 3) - (9 * 9)) + (6 * 3))$
- $((14 / 9) \% 5) - (((3 + 7) * (5 - 8)) + (9 * 6))$
- $((6 + 6) - (7 - 9)) + (6 - 9)) - (5 + 2)$
- $((24 \% 10) \% 6) + (((2 + 7) - (8 + 3)) + (6 * 8))$
- $((23 / 8) / 2) - (((5 * 7) + ((9 / 5))) - (8 + 2))$
- $(6 + 9) + (((8 + 7) + (8 + 7)) - (3 + 8))$

soma dos valores

Exercício 10 Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

- 1 $((12 / 5) \% 6) > (((2 * 7) + (4 - 8)) - (4 * 8))$
2 $(22 / 8) < (((9 + 7) + (6 - 5)) + (3 + 2))$
4 $(0 \&\& 0) \&\& ((0 \&\& 0) \&\& 1)$
8 $(17 / 9) > (((6 - 9) - (3 + 5)) + (2 + 3))$
16 $(29 \% 9) < (((9 - 2) + (3 - 4)) + (7 + 5))$
32 $(1 \text{ || } 1) \&\& ((1 \&\& 0) \text{ || } 0)$

soma das linhas V



- 1 - /

Números Inteiros

Geração de palíndromos

Dado um positivo, de dois dígitos ou mais, execute o seguinte algoritmo

```
1: x ← número positivo
2: y ← x invertido
3: z ← x + y
4: x ← z
```

repetidas vezes até que x seja um PALÍNDROMO. Um palíndromo é um número que pode ser lido da direita para a esquerda ou vice versa com o MESMO VALOR. Exemplos: 11, 187781, 45354 e assim por diante. No exercício a seguir, você deverá aplicar o algoritmo acima ao número dado e descobrir quantas repetições foram necessárias até chegar ao palíndromo.

Por exemplo, o número 2895 demanda 6 repetições que são 8877 16665 73326 135663 502194 993399. O 7832, três (10219 101420 125521) e o 4375 três também (10109 100210 112211).

Números homogêneos

O número *a* é dito homogêneo ao número *b* se os divisores de *a* dividem *b*. Por exemplo, no par (6,36) todos os divisores de *a* dividem *b*. Neste caso em particular *b* é homogêneo a *a*, mas isto não precisa acontecer sempre, veja-se

$$\begin{matrix} 6 & = & 2 \times 3 \\ 36 & = & 2^2 \times 3^2 \end{matrix}$$

Mais um exemplo, 255 é homogêneo a 19125, mas 19125 não é homogêneo a 125. No exercício a seguir, será mostrado um par de inteiros. Deve-se estudá-los e informar se *a* é homogêneo a *b* (respondendo 1) ou se eles não são um par desse tipo (respondendo 0).

Números amigos

Dois números inteiros *x, y* são amigos se a soma dos divisores do primeiro é igual ao segundo e se a soma dos divisores do segundo é igual ao primeiro. Em termos matemáticos, *s(x) = y* e *s(y) = x*, onde *s(m)* é a soma restrita (isto é sem o próprio *m*) dos divisores de *m*. O primeiro par de amigos conhecidos é 220,284, veja-se como

$$\begin{matrix} s(220) = \sum\{1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110\} = & 284 \\ s(284) = \sum\{1, 2, 4, 71, 142\} = & 220 \end{matrix}$$

Neste exercício, dado um número, você deve achar quem é o número amigo do número dado.

Número de Armstrong

Um número de Armstrong é um inteiro de *n* dígitos que é igual à soma das *n*-ésimas potências de seus dígitos. Este número também é chamado “número mais que perfeito”. Os dígitos 1..9 são todos números de Armstrong. Não há nenhum com *n* = 2 e o primeiro com *n* = 3 é 153, já que

$$153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$$

Neste exercício, você receberá um número e deverá dizer se ele é (respondendo 1) ou não é (respondendo 0) um número de Armstrong.

Número de Wodall

Este número tem a forma

$$W_n = (2^n . n) - 1$$

Assim, por exemplo, 7 é o segundo número de Wodall, já que

$$7 = (2^2 2) - 1$$

No exercício a seguir, você receberá um número e deverá responder se ele é (escrevendo 1) ou não é (escrevendo 0) um número de Wodall.

Números vampiros

Um número *v = xy*, com um número *n* par de dígitos, que pode ser decomposto em uma multiplicação de 2 fatores *x* e *y*, formados cada um exatamente por *n/2* dígitos. Tais dígitos são extraídos do número original em qualquer ordem.

Seja como exemplo, o número 1260, que pode ser escrito como 21 × 60. No exercício a seguir, você receberá um número e deverá responder se ele é vampiro (escrevendo 1) ou se ele não é vampiro (escrevendo 0).

Primos gêmeos

Um número é primo quando é divisível apenas por si e pela unidade. São primos os números 2,3,5,7,11,13,17,19,23,29... (crivo de Eratóstenes). A sequência de primos é infinita.

Dois números primos *p* e *q* são gêmeos, se e somente se *q* > *p* e também *q* − *p* = 2, ou seja eles diferem entre si por 2 unidades apenas. Na lista acima são primos gêmeos: (3,5), (5,7), (11,13), (17,19)...

No exercício a seguir, você receberá um número e deverá dizer se ele forma uma par gêmeo (1) ou não forma (0).

Mínimo Múltiplo Comum

Dados dois números inteiros, definem-se dois valores a eles relacionados denominados MMC (mínimo múltiplo comum) e MDC (máximo divisor comum). O máximo divisor comum é o maior inteiro que divide ambos os números originais sem deixar resto. O mínimo múltiplo comum é o menor número que ao ser dividido por ambos os números originais não deixa resto.

Por exemplo, o MDC(10,15) é 5 e o MMC(10,15) é 30. Ambas as quantidades estão relacionadas pela fórmula:

$$MMC(m, n) = \frac{m \times n}{MDC(m, n)}$$

Para o cálculo do MDC, pode-se usar o algoritmo devido a Euclides. Este algoritmo sempre é chamado com dois inteiros. Note que o primeiro número deve ser maior ou igual do que o segundo. Se esta condição não estiver satisfeita, na primeira chamada, os números são simplesmente invertidos, por característica do algoritmo. Isto pode ser feito, já que a operação MDC é comutativa. Este algoritmo é recursivo.

```
1: inteiro função MDC(inteiro a, b)
2: se b = 0 então
3:   devolva a
4: senão
5:   devolva MDC(b, (a mod b))
6: fim se
7: fim {função}
```

Suponha que far-se-á a chamada ao algoritmo acima, como segue:

MDC(1200, 1119)
0 que aconteceria ?
Chamando com 1200 1119, Chamando com 1119 81, Chamando com 81 66,
Chamando com 66 15, Chamando com 15 6, Chamando com 6 3 e
Chamando com 3 0
Resposta é 3

No exercício a seguir, você deverá calcular o MMC de dois números dados.

Para você fazer

Ciclo de palíndromo O número original é 2665 e a pergunta é quantos ciclos são necessários para que este número gere um palíndromo

Homogêneos Informe se os números 663 e 5714395 são (1) ou não são (0) Números Homogêneos

Amigo Ache o número amigo de 5020

Armstrong Informe se o número 370 é (1) ou não é (0) um número de Armstrong

Wodall Informe se o número 10240 é (1) ou não é (0) um número de Wodall

Vampiro Informe se o número 109810 é (1) ou não é (0) um número vampiro

Primo Gêmeo Informe se o número 221 é (1) ou não é (0) um elemento de um par de primos gêmeos.

MMC Ache o mínimo múltiplo comum de 132 e 144

Responda no quadro a seguir

ciclo	homog	amigo	armstrong

wodall	vampiro	pgemeo	mmc

1

Número de uns

Este número é a MENOR quantidade de 1's necessários para representar um inteiro usando apenas adições, multiplicações e parênteses. Por exemplo

1=1
2=1+1
3=1+1+1
4=(1+1)(1+1)=1+1+1+1
5=(1+1)(1+1)+1=1+1+1+1+1
6=(1+1)(1+1+1)
7=(1+1)(1+1+1)+1
8=(1+1)(1+1)(1+1)
9=(1+1+1)(1+1+1)
10=(1+1+1)(1+1+1)+1=(1+1)(1+1+1+1+1)

No exercício a seguir, você receberá um número e deverá descobrir qual é o número da uns dele.

Número de Cullen Um número de Cullen tem a forma

Cn = 2^n n + 1

Por exemplo, o segundo número de Cullen é C2 = ((2^2) × 2) + 1 = 9

No exercício a seguir, você receberá um número e deverá dizer se ele é (respondendo 1) ou não é (respondendo 0) um número de Cullen.

Número Euclideano O n-ésimo número Euclideano, está definido por

En = 1 + ∏i=1n pi,

onde pi é o i-ésimo número primo. Lembre-se que, por definição, o primeiro primo é o número 2. Tem-se então a sequência Pi = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...}. O terceiro número Euclideano é E3 = 1 + (2 × 3 × 5) = 31.

No exercício a seguir, você receberá um número Euclideano qualquer e deve dizer se ele é o primeiro, (respondendo 1), segundo (2), terceiro, etc.

Teorema de Euler 6n + 1 Diz o teorema

Qualquer primo P que tenha a forma 6n + 1 pode ser escrito na forma x^2 + 3 × y^2 com n, x e y inteiros. Ou seja é verdade

P = 6n + 1 = x^2 + 3y^2

No exercício a seguir, é dado P, e deve-se achar n, x e y.

Primo Fatorial

Um primo P é chamado primo fatorial quando ele é igual a n! ± 1.

No exercício a seguir, é dado P e deve-se achar n.

Quebra Cabeça 100 O autor Madachy escreveu um livro bem interessante, infelizmente esgotado, onde ele escreve diversas expressões cujo resultado sempre é 100, usando uma única vez cada um dos 9 dígitos 1..9 na expressão. Veja um exemplo:

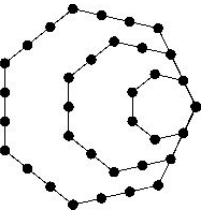
(7 - 5)^2 + 96 + 8 - 4 - 3 - 1 = 100

Neste exercício, você receberá uma expressão deste tipo onde aparecerão x, y e z, no lugar de 3 desses dígitos. Deve-se descobrir qual o valor de x,y e z.

Números de Harshad Um número de Harshad é um inteiro positivo que é divisível pela soma de seus dígitos. Por exemplo o número 111 é um Número de Harshad, já que 111 é divisível por 1+1+1=3. (111 ÷ 3 = 37).

No exercício a seguir, vem um número inteiro e deve-se dizer se ele é (respondendo 1) ou não é (0) um número de Hardshad.

Números heptagonais Certos números permitem a construção de um heptágono, como se pode ver na figura



Algebricamente falando, tais números obedecem à lei de formação n(5n - 3)/2

Por exemplo, os números 1, 7 e 18 são números heptagonais.

No exercício, dá-se um número e deve-se responder se ele é (1) heptagonal ou não.

Números de Ackermann A função de Ackermann é uma das funções conhecidas que mais cresce, (mais que a exponencial), e tem a seguinte definição:

A(x,y) = { y + 1 se x=0
A(x - 1, 1) se y=0
A(x - 1, A(x, y - 1)) senão

Você deve calcular qual o resultado pedido. Por exemplo ackermann(3,2)=29, ackermann(0,11)=12, ackermann(3,3)=61 e assim por diante.

Primo Bom Um número primo Pn é dito primo bom se e somente se Pn^2 > Pn-1 × Pn+1 para qualquer 1 ≤ i ≤ n - 1

Por exemplo, 17 é um primo bom, já que: 17^2 = 289 e n = 7 (ele é o sétimo número primo). Para ver se ele é bom mesmo, há que se fazer

17^2 = 289 > P6 × P8 13 × 19 = 247
17^2 = 289 > P5 × P9 11 × 23 = 253
17^2 = 289 > P4 × P10 7 × 29 = 203
17^2 = 289 > P3 × P11 5 × 31 = 155
17^2 = 289 > P2 × P12 3 × 37 = 111
17^2 = 289 > P1 × P13 2 × 41 = 82

No exercício, será mostrado um Primo P e deve-se informar se ele é (1) um primo bom ou não é (0).

Para você fazer

Número de uns Qual o número de uns de 22 ?

Cullen Informe se 2049 é (1) ou não é (0) um número de Cullen

Euclideano Ache o número Euclideano de 223092871

Euler 6n + 1 No Teorema de Euler conhecido como 6n + 1, ache n, x e y, sabendo que P= 31

Primo Fatorial Ache o valor de n considerando P= 5039

Quebra Cabeça 100 Informe os valores de x, y e z
xy3 + 8z - 67 - 45 = 100

Harshad Informe se o número 497 é (1) ou não é (0) um Número de Hardshad

Heptagonal Informe se o número 8444 é (1) ou não é (0) um Número Heptagonal

Ackerman Calcule Ack(m,n), onde m= 70 e n= 171498

Primo Bom Informe se o número 53 é (1) ou não é (0) um Primo Bom

Responda no quadro a seguir

Uns	Cullen	Euclid	6n + 1	P.Fat
100	Harshad	Hept	Ackermann	P.Bom



Primos de Germain

Sophie Germain nasceu em 1776, filha de um negociante, bem de vida, mas longe da nobreza. Sophie se encantou pela matemática ao ler em um livro a vida de Arquimedes. Mais do que a vida, a morte de Arquimedes. Pois, já aos 70 anos, estava Arquimedes olhando uma figura geométrica desenhada na areia da praia, quando um soldado romano, participante das tropas que recém haviam invadido Siracusa, quis saber o que aquele velho estava fazendo. O velho, dirigiu-se ao soldado e fez nenhum caso da interrupção. Deve ter dito algo assim como "Não encha o ...". O soldado não se fez de rogado e meteu a espada na barriga de Arquimedes, matando-o.

Sophie pensou que se algo era tão absorvente e inebriante deveria valer a pena estudá-lo. Ela começou e em breve estava repassando os textos de Euler e Newton, escondida, antes de dormir. Quando o pai dela descobriu, passou-lhe uma carraspana, confiscando velas e agasalhos para que ela deixasse de estudar matemática, onde já se viu. Sophie reagiu escondendo uma porção de velas e enrolando-se na roupa de cama. Passava frio mas não deixava de estudar.

Em 1794 fundou-se a Ecole Polytechnique em Paris. Templo do saber, existe até hoje, mas só tinha um problema. Era só para homens. Sophie, passou a frequentar a escola incógnita, vestida de homem. Apoderou-se da identidade de um ex-aluno, Monsieur Le Blanc. A escola não sabia que Le Blanc havia deixado Paris e continuou a imprimir resumos e exercícios para ele. Sophie escrupulosamente fazia os exercícios e devolvia-os à escola.

A coisa deu zebra, quando o professor supervisor do curso, Joseph Lagrange, outro grande matemático, quis uma entrevista com Le Blanc. Como era possível que um aluno que até então era uma anta, pudesse de uma hora para outra apresentar resultados tão maravilhosos? O que ele teria feito para aprender tão rápido e tão bem? Sophie foi obrigada a revelar seu segredo e foi um atônito mas contente Lagrange que passou a instruir e orientar "a" nova aluna. Saindo do já conhecido, Sophie começou a estudar áreas inexploradas. E sentiu necessidade de recorrer a Karl Gauss (há controvérsias, mas este aqui bem pode "o" maior matemático de todos os tempos). Com medo de ser rejeitada por Gauss, quem foi que assinou as cartas? Ele mesmo, M. Le Blanc.

A correspondência entre ambos ia de vento em popa, quando durante as guerras napoleônicas, o exército francês invadiu a Prússia. Sophie ficou com medo que seu guru tivesse o mesmo fim de Arquimedes, fosse morto por acaso. Falou sobre Gauss com seu amigo o general francês Pernety, que comandava os exércitos invasores. Este, impressionado pelo interesse de Sophie fez questão de visitar pessoalmente Gauss, dizendo-lhe que sua vida estava salva graças a interferência de Mademoiselle Germain. Gauss tomou um susto, quem seria essa mulher?

Na próxima carta, o mistério se desfez: M. Le Blanc mudou de sexo de novo. Gauss respondeu com uma carta belíssima a Sophie reconhecendo muito o trabalho dela, "ainda mais por ser mulher".

Mais para o final da vida, Gauss convenceu a Universidade de Goettingen a oferecer um título honorífico a Sophie German. Seria algo inédito, nunca antes uma mulher conseguira isso. Vencidas as barreiras, quando a universidade ia homenageá-la, um câncer a levou primeiro. Como final desta história, quando a Torre Eiffel foi erguida, colocou-se uma placa com o nome de 72 cientistas franceses cujo trabalho permitira erguer aquele monumento. Sophie, cujo colaboração provavelmente foi a mais importante dos 72, não estava lá.

Um primo p é dito "Primo de Germain" se p é primo e também $2p + 1$ é primo. Assim, 2 é um primo de Germain, já que 2 é primo e $2 \cdot 2 + 1 = 5$ também o é.

No exercício a seguir, você receberá um número e deverá responder 1 se o número é um Primo de Germain, ou 0 se não é.

Exemplo: 191 é um Primo de Germain, já que 191 é primo e $2 \times 191 + 1 = 383$ é primo também. Já 263 não é primo de Germain, pois embora primo, o valor $527 (2 \times 263 + 1 = 527)$ não é primo, já que seus divisores são 17 e 31.

Par Roth Aarão de números

Um par Roth Aarão é um par de números consecutivos n e $n + 1$ tal que a soma dos fatores primos de n e $n + 1$ são iguais. A origem do nome é a quebra do record de *home run* que era de 714 (Roth) e passou a ser de 715 em 8 de abril de 1974, quebrado que foi pelo Aarão. De fato, (714,715) é um par de Roth-Aarão. Veja-se porque:

$$\begin{aligned} 714 &= 2 \cdot 3 \cdot 7 \cdot 17 \\ 715 &= 5 \cdot 11 \cdot 13 \end{aligned}$$

Note que $2 + 3 + 7 + 17 = 5 + 11 + 13 = 29$. Se fatores múltiplos (por exemplo 2^3) são contados uma só vez (apenas um 2 neste caso), o par é dito próprio. Se fatores múltiplos são contados em todas as suas ocorrências ($2 + 2 + 2$, neste caso), o par é dito impróprio. Alguns exemplos: (24,25) é um par próprio, pois os divisores de 24 são: 2,3,4,6,8 e 12, dos quais só são primos os números 2 e 3, que somados dão 5. O 25 tem como fator o 5^2 , que somado apenas uma vez é 5. Logo o par (24,25) é próprio. O par (8,9) é um par impróprio, pois os fatores de 8 são 2 e 4, sendo que o 4 é descartado como não primo e o 2, na verdade é 2^3 , logo a soma dos fatores é $2 + 2 + 2 = 6$. Já o 9, tem como fator o 3, mas 3^2 , cuja soma é $3 + 3 = 6$, e portanto (8,9) é um par impróprio. No exercício a seguir, você receberá um par de números seguidos. Deve responder

- 0 se o par não for de Roth-Aarão
- 1 se o par for próprio
- 2 se o par for impróprio

Sequências de Cães e Gatos

A sequência de Cães e Gatos é assim definida: $C(0) = 1$ e $G(0) = 0$ e

$$\begin{aligned} C(n) &= n - G(C(n-1)) \\ G(n) &= n - C(G(n-1)) \end{aligned}$$

Os primeiros 10 termos da sequência de Cães são: 1, 1, 2, 2, 3, 3, 4, 5, 5, 6. Os dez primeiros Gatos são: 0, 0, 1, 2, 2, 3, 4, 4, 5, 6.

No exercício a seguir, você receberá n e deve calcular o $C(n)$ e o $G(n)$.

Sequencia de Jugler

Dado um número inicial, a_1 , a sequência de Jugler é um conjunto de números obtidos pela seguinte lei de formação

$$a_{k+1} = \begin{cases} \lfloor a_k^{1/2} \rfloor & \text{para } a_k \text{ par} \\ \lfloor a_k^{3/2} \rfloor & \text{para } a_k \text{ impar} \end{cases}$$

Por exemplo, começando com $a_1 = 77$ a sequência completa de Jugler é 77, 675, 17537, 2322378, 1523, 59436, 243, 3787, 233046, 482, 21, 96, 9, 27, 140, 11, 36, 6, 2, 1.

Sequência de Kantek

Para esta sequência, define-se a função $Q(n)$ = quantidade de divisores distintos de n , incluindo nesta quantidade a unidade e o próprio n . Assim, por exemplo, $Q(6) = 4$, já que os divisores de 6 são: 1, 2, 3 e 6. Define-se, por hipótese, $a_1 = 2$. Para o cálculo do termo genérico da sequência de Kantek, usa-se a seguinte fórmula:

$$k_n = \begin{cases} k_{n-1} & \text{para } k_{n-1} > Q(n) \\ Q(n) & \text{para } k_{n-1} \leq Q(n) \end{cases}$$

Uma definição alternativa é

$$k_n = \text{maior}\{Q(n) : k_{n-1}\}$$

Por exemplo, $k_{11} = 4$, $k_{12} = 6$ e $k_{13} = 6$.

Sequência de Fibonacci

Esta série de números foi estudada por Fibonacci, no século XIII, há mais de 800 anos. Os primeiros 10 termos da série são: 1 1 2 3 5 8 13 21 34 55. Neste exercício, serão dados dois termos quaisquer sempre separados por dois termos desconhecidos. O aluno deve achar o termo que se encontra três termos antes do primeiro termo dado. Por exemplo, e usando a sequência acima, se fossem dados os termos $F_7 = 13$ e $F_{10} = 55$, e pedido o F_3 , a resposta seria 2.

Números odiosos

Um número inteiro é odioso quando a sua representação binária tem um número ímpar de algarismos 1. O primeiro número odioso é o 1 ($x'1'$), o segundo é o 2 ($x'10'$) o terceiro é o 4 ($x'100'$). Note-se que o 3 não é odioso, já que sua representação em binário é $x'11'$ e o número de 1's é par. No exercício, da-se o número n e o aluno deve calcular qual o *ODIOSO*(n).

Para você fazer

Questão	Resposta
1. 277 é primo de germain ?	
2. 761 é primo de germain ?	
3. 16932 é o primeiro número em um par de Roth-Aarão ?	
4. 5959 é o primeiro número em um par de Roth-Aarão ?	
5. Quanto vale Cão(n), quando n= 15	
6. Quanto vale Gato(n), quando n= 15	
7. Em Jugler, dado $a_1 = 97$ Calcule o termo $n = 11$	
8. Qual o termo 4 da sequencia de Kantek ?	
9. Sabendo que em Fibonacci, o termo F_{n+4} vale 39088169 e o termo $F_{n+7} = 165580141$ qual o termo F_n ?	
10. Qual o ODIOSO(n) quando n vale 35	

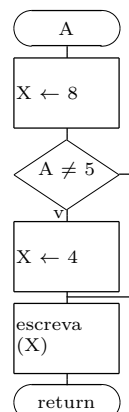


Comando SE

Nos exercícios a seguir, estão colocados pequenos trechos de código envolvendo o comando SE. Para auxiliar, junto está um fluxograma desse mesmo comando. Sugere-se que o aluno tente resolver usando o código e depois confira o resultado achado através do fluxograma.

Ex. 1

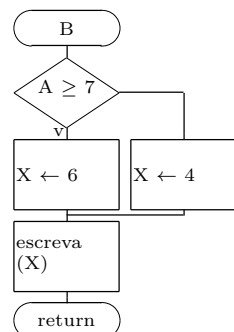
```
algoritmo A
A, X : inteiro
X ← 8
se (A ≠ 5) entao
    X ← 4
fimse
escreva(X)
finalgoritmo
```



Suponha que a variável A, assume os seguintes valores: 3, 5 e 7 e em cada um dos casos, o algoritmo deve dar como resposta:

Ex. 2

```
algoritmo B
A, X : inteiro
se (A ≥ 7) entao
    X ← 6
senao
    X ← 4
fimse
escreva(X)
finalgoritmo
```

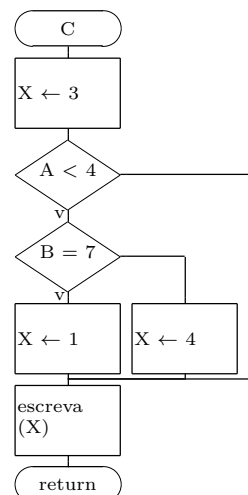


Suponha que a variável A, assume os seguintes valores: 6, 7 e 8 e em cada

um dos casos, o algoritmo deve dar como resposta:

Ex. 3

```
algoritmo C
A, B, X : inteiro
X ← 3
se (A < 4) entao
    se (B = 7) entao
        X ← 1
    senao
        X ← 4
    fimse
fimse
escreva(X)
finalgoritmo
```



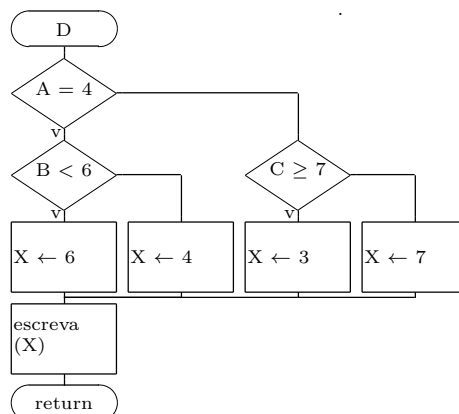
Suponha 3 casos, onde A = 2 e B = 6; A = 4 e B = 7 e finalmente A = 6 e B = 9

em cada um dos casos, o algoritmo deve dar como resposta:

Ex. 4

```
algoritmo D
A, B, C, X : inteiro
se (A = 4) entao
    se (B < 6) entao
        X ← 6
    senao
        X ← 4
    fimse
senao
    se (C ≥ 7) entao
        X ← 3
    senao
        X ← 7
    fimse
fimse
escreva(X)
finalgoritmo
```

fimse
escreva(X)
finalgoritmo

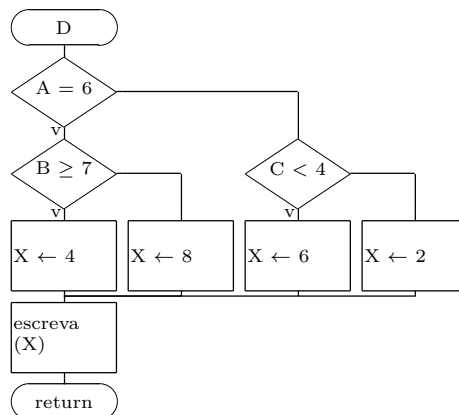


Suponha 3 casos, onde A = 3, B = 5 e C = 6; A = 4, B = 6 e C = 7 e finalmente A = 6, B = 7 e C = 8

em cada um dos casos, o algoritmo deve dar como resposta:

Ex. 5

```
algoritmo D
A, B, C, X : inteiro
se (A = 6) entao
    se (B ≥ 7) entao
        X ← 4
    senao
        X ← 8
    fimse
senao
    se (C < 4) entao
        X ← 6
    senao
        X ← 2
    fimse
fimse
escreva(X)
finalgoritmo
```

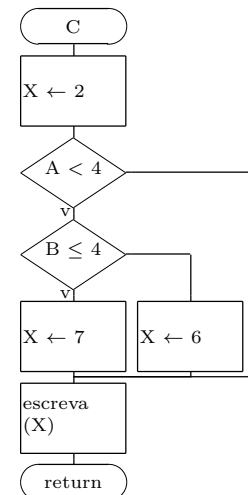


Suponha 3 casos, onde A = 5, B = 6 e C = 3; A = 6, B = 7 e C = 4 e finalmente A = 7, B = 9 e C = 5

em cada um dos casos, o algoritmo deve dar como resposta:

Ex. 6

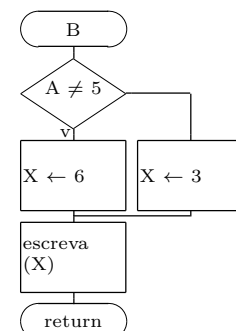
```
algoritmo C
A, B, X : inteiro
X ← 2
se (A < 4) entao
    se (B ≤ 4) entao
        X ← 7
    senao
        X ← 6
    fimse
fimse
escreva(X)
finalgoritmo
```



Suponha 3 casos, onde A = 2 e B = 3; A = 4 e B = 4 e finalmente A = 5 e B = 6

Ex. 7

```
algoritmo B
A, X : inteiro
se (A ≠ 5) entao
    X ← 6
senao
    X ← 3
fimse
escreva(X)
finalgoritmo
```



Suponha que a variável A, assume os seguintes valores: 3, 5 e 7



Comando SE

Nos exercícios a seguir, estão colocados pequenos trechos de código envolvendo o comando SE. Para auxiliar, junto está um esquema para ajudar no teste de mesa.

Ex. 1

```
1: Algoritmo A
2: A, X: inteiro
3: X ← 1
4: se (A ≠ 5)
5:   X ← 8
6: fim{se}
7: escreva (X)
8: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
3	
5	
6	

Ex. 2

```
1: Algoritmo B
2: A, X : inteiro
3: se (A > 7)
4:   X ← 2
5: senão
6:   X ← 7
7: fim{se}
8: escreva(X)
9: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
5	
7	
8	

Ex. 3

```
1: algoritmo C
2: A, B, X : inteiro
3: X ← 4
4: se (A = 5)
5:   se (B = 5)
6:     X ← 5
7:   senão
8:     X ← 7
9:   fim{se}
10: fim{se}
11: escreva(X)
12: finalgoritmo
```

Suponha 3 casos, onde A e B tem os valores abaixo:

A	B	X
3	4	
5	5	
7	6	

Ex. 4

```
1: algoritmo D
2: A, B, C, X : inteiro
3: se (A = 7)
4:   se (B = 5)
5:     X ← 2
6:   senão
7:     X ← 5
8:   fim{se}
9: senão
10:   se (C ≥ 7)
11:     X ← 1
12:   senão
13:     X ← 6
14:   fim{se}
15: fim{se}
16: escreva(X)
17: finalgoritmo
```

Suponha 3 casos, onde A e B tem os valores abaixo:

A	B	C	X
5	3	6	
7	5	7	
8	6	9	

Ex. 5

```
1: Algoritmo A
2: A, X: inteiro
3: X ← 6
4: se (A > 7)
5:   X ← 2
6: fim{se}
7: escreva (X)
8: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
5	
7	
9	

Ex. 6

```
1: Algoritmo B
2: A, X : inteiro
3: se (A ≤ 5)
4:   X ← 8
5: senão
6:   X ← 6
7: fim{se}
8: escreva(X)
9: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
4	
5	
6	

Ex. 7

```
1: algoritmo C
2: A, B, X : inteiro
3: X ← 6
4: se (A > 4)
5:   se (B ≥ 4)
6:     X ← 5
7:   senão
8:     X ← 7
9:   fim{se}
10: fim{se}
11: escreva(X)
12: finalgoritmo
```

Suponha 3 casos, onde A e B tem os valores abaixo:

A	B	X
3	2	
4	4	
6	6	

Ex. 8

```
1: algoritmo D
2: A, B, C, X : inteiro
3: se (A < 6)
4:   se (B ≠ 4)
5:     X ← 5
6:   senão
7:     X ← 3
8:   fim{se}
9: senão
10:   se (C ≠ 7)
11:     X ← 6
12:   senão
13:     X ← 8
14:   fim{se}
15: fim{se}
16: escreva(X)
17: finalgoritmo
```

Suponha 3 casos, onde A e B tem os valores abaixo:

A	B	C	X
5	2	6	
6	4	7	
7	6	8	

Ex. 9

```
1: Algoritmo A
2: A, X: inteiro
3: X ← 5
4: se (A > 4)
5:   X ← 2
6: fim{se}
7: escreva (X)
8: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
3	
4	
6	

Ex. 10

```
1: Algoritmo B
2: A, X : inteiro
3: se (A ≤ 4)
4:   X ← 1
5: senão
6:   X ← 6
7: fim{se}
8: escreva(X)
9: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
3	
4	
6	

Ex. 11

```
1: algoritmo C
2: A, B, X : inteiro
3: X ← 7
4: se (A ≥ 6)
5:   se (B ≠ 5)
6:     X ← 3
7:   senão
8:     X ← 2
9:   fim{se}
10: fim{se}
11: escreva(X)
12: finalgoritmo
```

Suponha 3 casos, onde A e B tem os valores abaixo:

A	B	X
5	4	
6	5	
7	7	

Ex. 12

```
1: algoritmo D
2: A, B, C, X : inteiro
3: se (A > 5)
4:   se (B ≤ 7)
5:     X ← 6
6:   senão
7:     X ← 1
8:   fim{se}
9: senão
10:   se (C ≠ 7)
11:     X ← 3
12:   senão
13:     X ← 4
14:   fim{se}
15: fim{se}
16: escreva(X)
17: finalgoritmo
```

Suponha 3 casos, onde A e B tem os valores abaixo:

A	B	C	X
4	6	5	
5	7	7	
7	8	8	

Ex. 13

```
1: Algoritmo A
2: A, X: inteiro
3: X ← 2
4: se (A ≥ 5)
5:   X ← 4
6: fim{se}
7: escreva (X)
8: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
4	
5	
7	

Ex. 14

```
1: Algoritmo B
2: A, X : inteiro
3: se (A ≠ 5)
4:   X ← 1
5: senão
6:   X ← 5
7: fim{se}
8: escreva(X)
9: finalgoritmo
```

Suponha que a variável A, assume os seguintes valores:

A	X
4	
5	
7	

Ex. 15

```
1: algoritmo C
2: A, B, X : inteiro
3: X ← 8
4: se (A ≥ 7)
5:   se (B ≠ 7)
6:     X ← 2
7:   senão
8:     X ← 5
9:   fim{se}
10: fim{se}
11: escreva(X)
12: finalgoritmo
```

Suponha 3 casos, onde A e B tem os valores abaixo:

A	B	X
5	6	
7	7	
9	8	



Comando Condicional SE

Nos exercícios a seguir, siga a lógica do algoritmo e escreva o valor final impresso por ele.

Ex. 1

1: A ← 1
2: se (A > 1)
3: A ← A + 1
4: senão
5: A ← A - 1
6: fim{se}
7: A ← A × 2
8: escreva A

Ex. 2

1: B ← 2
2: C ← 5
3: se (B ≥ C)
4: B ← C
5: senão
6: C ← B
7: fim{se}
8: escreva B

Ex. 3

1: E ← 6
2: F ← 4
3: se (E < 7)
4: E ← E + 6
5: fim{se}
6: se (F ≥ 5)
7: F ← F - 7
8: fim{se}
9: escreva E + F

Ex. 4

1: D ← 1
2: se ((D - 4) ≥ 0)
3: D ← D + 3
4: senão
5: D ← D - 4
6: fim{se}
7: D ← ⌈D ÷ 8
8: escreva D

Ex. 5

1: G ← 4
2: se (G ≥ 8)
3: G ← G + 3
4: se (G < 5)
5: G ← G + 6
6: fim{se}
7: fim{se}
8: escreva G

Ex. 6

1: H ← 2
2: se ((H + 10) > 14)
3: H ← H + 1
4: H ← H × 2
5: fim{se}
6: H ← H + 1
7: H ← H × 3
8: escreva H

Ex. 7

1: A ← 1
2: se (A > 4)
3: A ← A + 1
4: senão
5: A ← A - 1
6: fim{se}
7: A ← A × 2
8: escreva A

Ex. 8

1: B ← 5
2: C ← 6

3: se (B ≥ C)
4: B ← C
5: senão
6: C ← B
7: fim{se}
8: escreva B

Ex. 9

1: E ← 7
2: F ← 6
3: se (E < 7)
4: E ← E + 5
5: fim{se}
6: se (F ≥ 6)
7: F ← F - 7
8: fim{se}
9: escreva E + F

Ex. 10

1: D ← 1
2: se ((D - 4) ≥ 0)
3: D ← D + 6
4: senão
5: D ← D - 2
6: fim{se}
7: D ← ⌈D ÷ 7
8: escreva D

Ex. 11

1: G ← 6
2: se (G ≥ 7)
3: G ← G + 8
4: se (G < 6)
5: G ← G + 2
6: fim{se}
7: fim{se}
8: escreva G

Ex. 12

1: H ← 4
2: se ((H + 10) > 14)
3: H ← H + 1
4: H ← H × 2
5: fim{se}
6: H ← H + 1
7: H ← H × 3
8: escreva H

Ex. 13

1: A ← 1
2: se (A > 7)
3: A ← A + 1
4: senão
5: A ← A - 1
6: fim{se}
7: A ← A × 2
8: escreva A

Ex. 14

1: B ← 5
2: C ← 5
3: se (B ≥ C)
4: B ← C
5: senão
6: C ← B
7: fim{se}
8: escreva B

Ex. 15

1: E ← 6
2: F ← 4
3: se (E < 6)
4: E ← E + 3
5: fim{se}
6: se (F ≥ 6)
7: F ← F - 6
8: fim{se}
9: escreva E + F

Ex. 16

1: D ← 6
2: se ((D - 4) ≥ 0)
3: D ← D + 2
4: senão
5: D ← D - 6
6: fim{se}
7: D ← ⌈D ÷ 8
8: escreva D

Ex. 17

1: G ← 8
2: se (G ≥ 4)
3: G ← G + 3
4: se (G < 2)
5: G ← G + 2
6: fim{se}
7: fim{se}
8: escreva G

Ex. 18

1: H ← 5
2: se ((H + 10) > 14)
3: H ← H + 1
4: H ← H × 2
5: fim{se}
6: H ← H + 1
7: H ← H × 3
8: escreva H

Ex. 19

1: A ← 7
2: se (A > 1)
3: A ← A + 1
4: senão
5: A ← A - 1
6: fim{se}
7: A ← A × 2
8: escreva A

Ex. 20

1: B ← 5
2: C ← 6
3: se (B ≥ C)
4: B ← C
5: senão
6: C ← B
7: fim{se}
8: escreva B

Ex. 21

1: E ← 1
2: F ← 1
3: se (E < 4)
4: E ← E + 6
5: fim{se}
6: se (F ≥ 1)
7: F ← F - 4
8: fim{se}
9: escreva E + F

Ex. 22

1: D ← 3
2: se ((D - 4) ≥ 0)
3: D ← D + 3
4: senão
5: D ← D - 8
6: fim{se}
7: D ← ⌈D ÷ 3
8: escreva D

Ex. 23

1: G ← 2
2: se (G ≥ 6)
3: G ← G + 3
4: se (G < 3)
5: G ← G + 2
6: fim{se}
7: fim{se}
8: escreva G

Ex. 24

1: H ← 6
2: se ((H + 10) > 14)
3: H ← H + 1
4: H ← H × 2
5: fim{se}
6: H ← H + 1
7: H ← H × 3
8: escreva H

Ex. 25

1: A ← 5
2: se (A > 5)
3: A ← A + 1
4: senão
5: A ← A - 1
6: fim{se}
7: A ← A × 2
8: escreva A

Ex. 26

1: B ← 3
2: C ← 3
3: se (B ≥ C)
4: B ← C
5: senão
6: C ← B
7: fim{se}
8: escreva B

Ex. 27

1: E ← 2
2: F ← 2
3: se (E < 4)
4: E ← E + 6
5: fim{se}
6: se (F ≥ 2)
7: F ← F - 4
8: fim{se}
9: escreva E + F

Respostas

1.A	2.B	3. E+F	SOMA →
4.D	5.G	6.H	SOMA →
7.A	8.B	9. E+F	SOMA →
10.D	11.G	12.H	SOMA →
13.A	14.B	15. E+F	SOMA →
16.D	17.G	18.H	SOMA →
19.A	20.B	21. E+F	SOMA →
22.D	23.G	24.H	SOMA →
25.A	26.B	27. E+F	SOMA →



Comando Condicional SE

Nos exercícios a seguir, siga a lógica do algoritmo e escreva o valor final impresso por ele. No final de tudo, some os valores encontrados, de 3 em 3 conforme mostra a grade de correção.

Ex. 1

```
1: A ← 3
2: B ← 8
3: C ← 4
4: D ← 8
5: se (A > 6) ∧ (B ≤ 4)
6:   se (C < D)
7:     C ← C + 1
8:   senão
9:     D ← D + 3
10:  fim{se}
11: senão
12:   se (C > 3) ∨ (D < 1)
13:     C ← C × 2
14:   fim{se}
15: fim{se}
16: escreva (A + B + C)-D
```

Ex. 2

```
1: A ← 2
2: B ← 1
3: C ← 8
4: se ((A+C) > (B+6))
5:   A ← A + 5
6:   se (B > 4)
7:     A ← A + 2
8:   fim{se}
9:   se (C < 1)
10:    se (A > 11)
11:      C ← B + 2
12:    fim{se}
13:    C ← A + B
14:  fim{se}
15: fim{se}
16: escreva (A + B + C)
```

Ex. 3

```
1: G ← 1
2: B ← 1
3: F ← 6 + 2
4: se ((G > B) ∨ (F < 12))
5:   se (B ≥ 3)
6:     F ← F + 2
7:   fim{se}
8:   F ← F + 3
9: fim{se}
10: se (G ≤ 5)
11:   se (((B + 2) > 6) ∧ (F > 3))
12:     B ← B + 2
13:   fim{se}
14: fim{se}
15: G ← G + 1
16: escreva G + B + (2×F)
```

Ex. 4

```
1: H ← 2
2: B ← 5
3: J ← 4
4: L ← 4
5: se (H < 6)
6:   se (B < 5)
7:     se (J ≤ 5)
8:       K ← 4
9:     senão
10:      K ← 4
11:    fim{se}
12:    L ← K + 6
```

```
13:  fim{se}
14:  L ← L + 1
15: fim{se}
16: escrevaJ + L-H
```

Ex. 5

```
1: M ← 8
2: N ← 5
3: P ← 2
4: se (M < 7)
5:   se (N ≥ 2)
6:     se (P ≤ 4)
7:       M ← M + 6
8:     senão
9:       N ← N + 7
10:      P ← P + 1
11:    fim{se}
12:    M ← N + P
13:  fim{se}
14:  P ← N + M
15: fim{se}
16: escreva P + M + N
```

Ex. 6

```
1: K ← 8
2: M ← 8
3: H ← 1
4: K ← K + 6
5: M ← M + K
6: se (H < 2)
7:   K ← K + 1
8:   M ← K + 3
9: senão
10:  H ← H + 3
11:  M ← 2
12: fim{se}
13: H ← 1
14: K ← K + 8-3
15: M ← M + 8-1
16: escreva K + M + H
```

Ex. 7

```
1: A ← 4
2: B ← 2
3: C ← 8
4: D ← 5
5: se (A > 6) ∧ (B ≤ 2)
6:   se (C < D)
7:     C ← C + 1
8:   senão
9:     D ← D + 3
10:  fim{se}
11: senão
12:   se (C > 1) ∨ (D < 3)
13:     C ← C × 2
14:   fim{se}
15: fim{se}
16: escreva (A + B + C)-D
```

Ex. 8

```
1: A ← 5
2: B ← 6
3: C ← 3
4: se ((A+C) > (B+6))
5:   A ← A + 5
6:   se (B > 4)
7:     A ← A + 2
8:   fim{se}
9:   se (C < 1)
10:    se (A > 11)
11:      C ← B + 2
12:    fim{se}
13:    C ← A + B
14:  fim{se}
15: fim{se}
16: escreva (A + B + C)
```

Ex. 9

```
1: G ← 1
2: B ← 3
3: F ← 3 + 2
4: se ((G > B) ∨ (F < 12))
5:   se (B ≥ 3)
6:     F ← F + 2
7:   fim{se}
8:   F ← F + 3
9: fim{se}
10: se (G ≤ 5)
11:   se (((B + 2) > 6) ∧ (F > 3))
12:     B ← B + 2
13:   fim{se}
14: fim{se}
15: G ← G + 1
16: escreva G + B + (2×F)
```

Ex. 10

```
1: H ← 2
2: B ← 3
3: J ← 4
4: L ← 8
5: se (H < 6)
6:   se (B < 5)
7:     se (J ≤ 5)
8:       K ← 5
9:     senão
10:      K ← 6
11:    fim{se}
12:    L ← K + 4
13:  fim{se}
14:  L ← L + 1
15: fim{se}
16: escrevaJ + L-H
```

Ex. 11

```
1: M ← 6
2: N ← 4
3: P ← 3
4: se (M < 7)
5:   se (N ≥ 3)
6:     se (P ≤ 5)
7:       M ← M + 2
8:     senão
9:       N ← N + 7
10:      P ← P + 2
11:    fim{se}
12:    M ← N + P
13:  fim{se}
14:  P ← N + M
15: fim{se}
16: escreva P + M + N
```

Ex. 12

```
1: K ← 5
2: M ← 4
3: H ← 8
4: K ← K + 2
5: M ← M + K
6: se (H < 5)
7:   K ← K + 6
8:   M ← K + 4
9: senão
10:  H ← H + 2
11:  M ← 2
12: fim{se}
13: H ← 1
14: K ← K + 5-3
15: M ← M + 4-1
16: escreva K + M + H
```

Ex. 13

```
1: A ← 1
2: B ← 5
```

```
3: C ← 2
4: D ← 8
5: se (A > 4) ∧ (B ≤ 2)
6:   se (C < D)
7:     C ← C + 1
8:   senão
9:     D ← D + 3
10:  fim{se}
11: senão
12:   se (C > 6) ∨ (D < 3)
13:     C ← C × 2
14:   fim{se}
15: fim{se}
16: escreva (A + B + C)-D
```

Ex. 14

```
1: A ← 4
2: B ← 5
3: C ← 7
4: se ((A+C) > (B+6))
5:   A ← A + 5
6:   se (B > 4)
7:     A ← A + 2
8:   fim{se}
9:   se (C < 3)
10:    se (A > 11)
11:      C ← B + 2
12:    fim{se}
13:    C ← A + B
14:  fim{se}
15: fim{se}
16: escreva (A + B + C)
```

Ex. 15

```
1: G ← 6
2: B ← 7
3: F ← 3 + 6
4: se ((G > B) ∨ (F < 12))
5:   se (B ≥ 3)
6:     F ← F + 2
7:   fim{se}
8:   F ← F + 3
9: fim{se}
10: se (G ≤ 5)
11:   se (((B + 2) > 6) ∧ (F > 3))
12:     B ← B + 2
13:   fim{se}
14: fim{se}
15: G ← G + 1
16: escreva G + B + (2×F)
```

Respostas

1.	2.	3.	SOMA →
4.	5.	6.	SOMA →
7.	8.	9.	SOMA →
10.	11.	12.	SOMA →
13.	14.	15.	SOMA →

1



Comando Condicional SE

Nos exercícios a seguir, siga a lógica do algoritmo e escreva o valor final impresso por ele.

1.
1: funcao AAA
2: A←3;B←6;C←4;D←3;E←3;F←5
3: se A>D
4: A←E-1
5: senão
6: F←B-3
7: D←E-1
8: fim{se}
9: se (~ ((D=E) ∧ (E≠F)))
10: se (E>0) ∨ (F≥D)
11: D←C-1
12: se (B>7) ∧ (A>6)
13: B←E+2
14: fim{se}
15: se A≥9
16: D←B-4
17: B←A+3
18: senão
19: E←C-1
20: fim{se}
21: A←F-1
22: senão
23: C←F+2
24: D←D-2
25: fim{se}
26: A←E+3
27: senão
28: C←F-2
29: fim{se}
30: escreva (A+B+C+D+E+F)
31: fimfuncao
2.
1: funcao AAA
2: A←5;B←3;C←2;D←6;E←6;F←6
3: se (E≥D) ∨ (E=0)
4: D←C+1
5: senão
6: A←A-2
7: fim{se}
8: se (C>5) ∧ (D=B)
9: D←C+1
10: C←D+1
11: E←E+3
12: senão
13: E←A-3
14: E←D+1
15: fim{se}
16: F←B+3
17: se (~ ((F<6) ∧ (D<1)))
18: A←C-1
19: D←A-1
20: fim{se}
21: se (F<D) ∨ (D≥7)
22: se (~ (C≥A))
23: B←D+1
24: fim{se}
25: E←B+3
26: senão
27: F←C+2
28: fim{se}
29: escreva (A+B+C+D+E+F)
30: fimfuncao
3.
1: funcao AAA
2: A←4;B←4;C←1;D←1;E←1;F←4
3: se A=2
4: B←D-2

- 5: senão
6: A←E-2
7: fim{se}
8: se E<8
9: se (C≠0) ∧ (E=9)
10: B←A-1
11: B←E+3
12: fim{se}
13: D←B-3
14: senão
15: A←E+1
16: F←A-1
17: fim{se}
18: F←A+2
19: se (~ (B≤3))
20: A←A-4
21: F←E-1
22: se (B>8) ∧ (A>5)
23: E←E+2
24: fim{se}
25: C←B+2
26: senão
27: D←B+4
28: fim{se}
29: escreva (A+B+C+D+E+F)
30: fimfuncao
4.
1: funcao AAA
2: A←3;B←3;C←3;D←5;E←4;F←2
3: C←B+1
4: se (F≠5) ∧ (A≥2)
5: C←B-1
6: fim{se}
7: se E≥3
8: se (F≥B) ∧ (D≠3)
9: D←C-3
10: senão
11: D←B-2
12: E←A-2
13: E←F+2
14: A←D-3
15: A←E+3
16: E←C+4
17: fim{se}
18: A←F+1
19: fim{se}
20: se (~ ((D>3) ∧ (C≠5)))
21: A←F-1
22: fim{se}
23: escreva (A+B+C+D+E+F)
24: fimfuncao
5.
1: funcao AAA
2: A←5;B←3;C←6;D←1;E←1;F←6
3: se D≠2
4: A←C+4
5: fim{se}
6: C←B+3
7: C←F-1
8: A←A+1
9: F←B+1
10: E←B+3
11: se D<8
12: F←B+2
13: fim{se}
14: se (D<5) ∧ (D≠B)
15: se (~ (B<F))
16: E←C+1
17: fim{se}
18: B←D-2
19: senão
20: F←C+1
21: se B>3
22: E←A-3

- 23: senão
24: C←E-3
25: fim{se}
26: B←C+1
27: fim{se}
28: escreva (A+B+C+D+E+F)
29: fimfuncao
6.
1: funcao AAA
2: A←6;B←2;C←4;D←3;E←4;F←5
3: F←A+3
4: B←D+4
5: A←C+1
6: D←A-2
7: se (~ (B≤5))
8: E←F+4
9: C←A-1
10: senão
11: se (B≠5) ∧ (A<5)
12: E←F+3
13: fim{se}
14: A←D+1
15: B←F-2
16: B←D-2
17: se A≤C
18: se (E<9) ∨ (A≥8)
19: E←C+2
20: senão
21: F←C+2
22: fim{se}
23: F←F-3
24: senão
25: C←B-2
26: fim{se}
27: C←A-1
28: fim{se}
29: escreva (A+B+C+D+E+F)
30: fimfuncao
7.
1: funcao AAA
2: A←6;B←6;C←5;D←5;E←1;F←4
3: A←F+3
4: A←E+2
5: F←D-1
6: C←A+1
7: D←C-1
8: D←D-3
9: F←D+2
10: se B<F
11: C←A+3
12: A←D-4
13: A←B-1
14: senão
15: C←A+1
16: fim{se}
17: se (D≥1) ∧ (F=6)
18: D←D-2
19: fim{se}
20: A←F-1
21: C←A-2
22: escreva (A+B+C+D+E+F)
23: fimfuncao
8.
1: funcao AAA
2: A←1;B←6;C←2;D←5;E←1;F←3
3: se (D≤0) ∧ (C≤0)
4: D←B-2
5: senão
6: A←E+2
7: fim{se}
8: C←B-3
9: B←E-2
10: se B=1

- 11: C←A-4
12: A←F+3
13: B←B+4
14: D←F-1
15: fim{se}
16: se B<5
17: se (~ ((D≥6) ∨ (C>1)))
18: D←A+1
19: B←F-4
20: se B<5
21: D←C+2
22: A←D+4
23: senão
24: E←F+3
25: fim{se}
26: C←C+1
27: senão
28: E←F-3
29: fim{se}
30: D←A-2
31: B←B+1
32: senão
33: C←C-1
34: fim{se}
35: escreva (A+B+C+D+E+F)
36: fimfuncao
9.
1: funcao AAA
2: A←1;B←5;C←2;D←2;E←5;F←3
3: se F>6
4: F←D-2
5: senão
6: C←C-1
7: fim{se}
8: B←D-4
9: se (~ ((A=4) ∨ (E<3)))
10: D←D+2
11: A←F-1
12: senão
13: se (E≥B) ∨ (A>F)
14: se (D≠0) ∧ (F=B)
15: A←B+1
16: senão
17: F←C+2
18: fim{se}
19: se B≤B
20: A←D+3
21: senão
22: E←D+1
23: fim{se}
24: E←A+1
25: E←B-1
26: senão
27: A←A-2
28: D←D+1
29: fim{se}
30: D←A-1
31: fim{se}
32: escreva (A+B+C+D+E+F)
33: fimfuncao

Respostas

1.	2.	3.
4.	5.	6.
7.	8.	9.

SEs compostos e encadeados

Comando Condicional Permite executar uma ou mais instruções caso uma condição seja verdadeira. Opcionalmente, pode ter a alternativa inversa, ou seja, executar outros comandos, se a condição for falsa. Comandos internos ao comando "se" devem estar identados de 3 espaços, para clareza.

```
se <condição>
    <comando-1>
    <comando-2>
    ...
[senão <comando-11>
    <comando-12>
    ...]

fimse
```

A alternativa simples só possui a primeira parte do comando (comandos são executados se a condição for verdadeira). A alternativa Composta possui as duas partes. A primeira, se a condição for verdadeira e a segunda se for falsa.

Lembre que o símbolo $\wedge$ indica E; o símbolo $\vee$ indica OU e finalmente o símbolo $\sim$ indica NÃO.

👉 Para você fazer

A seguir diversos algoritmos envolvendo apenas o comando de alternativa. Siga cada um dos algoritmos e ao final responda qual o valor impresso por cada um deles.

Exercício 1 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 6, 4, 2, 3, 4, 6

```
1: algoritmo EX1
2: se (C > 2)
3:   se ( $\sim$  (D ≤ 3))
4:     A ← E + 5
5:   senão
6:     B ← D - C
7:   fim{se}
8:   D ← C × E
9: senão
10:  A ← C + 4
11:  se ( $\sim$  (E ≤ 4))
12:    A ← E - 3
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 2 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 1, 2, 5, 2, 2, 3

```
1: algoritmo EX2
2: se (C > 6)
3:   se (B ≠ 2)
4:     F ← C - 4
5:   senão
6:     F ← F - A
7:   fim{se}
8:   E ← A - 5
9: senão
10:  D ← E × 5
11:  se ( $\sim$  (B ≥ 3))  $\wedge$  ( $\sim$  (A < 1))
12:    E ← D + F
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 3 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 5, 1, 5, 2, 6, 1

```
1: algoritmo EX3
2: se ((A ≤ 3)  $\wedge$  (B ≠ 3))
3:   se ((E ≠ 3)  $\vee$  ( $\sim$  (E ≠ 6)))
4:     D ← C × 3
5:   senão
6:     E ← B + F
7:   fim{se}
8:   C ← F + 3
9: senão
10:  D ← E × B
11:  se (D > 6)
12:    C ← B × F
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 4 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 5, 2, 4, 2, 1, 1

```
1: algoritmo EX4
2: se ( $\sim$  (A > 6))  $\vee$  (D = 5))
3:   se ((D < 3)  $\wedge$  (E = 6))
4:     E ← E - 4
5:   senão
6:     B ← D + 5
7:   fim{se}
8:   D ← B + 5
9: senão
10:  F ← E - D
11:  se ( $\sim$  (D ≤ 4))
12:    A ← D + 5
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 5 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 4, 2, 5, 4, 4, 4

```
1: algoritmo EX5
2: se ((D ≥ 3)  $\vee$  (A > 2))
3:   se (((D > 5)  $\vee$  (E < 1))  $\wedge$  ( $\sim$  (F ≠ 3)))
4:     A ← D - 4
5:   senão
6:     B ← B - A
7:   fim{se}
8:   D ← C - E
9: senão
10:  E ← C + 2
11:  se ((( $\sim$  (E < 1))  $\wedge$  ( $\sim$  (E ≤ 2)))  $\wedge$  (F < 1))
12:    D ← A × 2
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 6 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 5, 6, 4, 6, 1, 2

```
1: algoritmo EX6
2: se ( $\sim$  (E ≠ 1))  $\vee$  ( $\sim$  (B ≥ 3)))
3:   se ((F ≠ 6)  $\vee$  (A > 6))
4:     C ← F + A
5:   senão
6:     C ← C - B
7:   fim{se}
8:   E ← E × 3
9: senão
10:  D ← E + D
11:  se ( $\sim$  (B > 6))  $\wedge$  ( $\sim$  (B = 5)))
12:    B ← E + 5
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
```

16: fim algoritmo

Exercício 7 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 4, 5, 6, 1, 3, 6

```
1: algoritmo EX7
2: se (B ≤ 5)
3:   se (C > 5)
4:     D ← D - 2
5:   senão
6:     A ← A × 4
7:   fim{se}
8:   E ← C - D
9: senão
10:  C ← D - C
11:  se ((( $\sim$  (A > 5))  $\vee$  (A ≥ 2))  $\wedge$  (D ≠ 4))
12:    C ← B + F
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 8 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 4, 5, 2, 4, 2, 6

```
1: algoritmo EX8
2: se ( $\sim$  (E = 1))  $\vee$  ( $\sim$  (E = 1)))
3:   se ( $\sim$  (C > 4))  $\vee$  (D ≠ 3))
4:     D ← A × 3
5:   senão
6:     B ← A × 3
7:   fim{se}
8:   F ← C - E
9: senão
10:  B ← C × 5
11:  se (((C ≥ 1)  $\wedge$  ( $\sim$  (C < 2)))  $\vee$  (A > 1))
12:    C ← E + A
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 9 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 2, 5, 4, 2, 1, 4

```
1: algoritmo EX9
2: se ((C = 5)  $\vee$  ( $\sim$  (E > 6)))
3:   se ((( $\sim$  (F = 5))  $\wedge$  (D = 2))  $\wedge$  (C = 4))
4:     B ← E × C
5:   senão
6:     F ← F + 4
7:   fim{se}
8:   A ← D × 5
9: senão
10:  B ← D - B
11:  se ( $\sim$  (A > 2))  $\wedge$  (F = 5))
12:    A ← B - C
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Respostas

1	2	3	4	5
6	7	8	9	não responda aqui

Working Path

O working path (caminho de trabalho) é a sequência de comandos executados pelo algoritmo enquanto ele estiver sendo processado. Se não houver comandos condicionais nem comandos de repetição, então não haverá saltos (desvios) e com isso o working path é a sequência dos números naturais: começando no primeiro comando e terminando no último.

Neste exercício haverá comandos condicionais e portanto haverá saltos. Em cada algoritmo você deverá construir o Working Path. Trata-se de um vetor contendo o número dos comandos efetivamente executados ao processar um algoritmo.

Por exemplo, no algoritmo

```
1: algoritmo QUEBRA
2: A ← 55
3: escreva (A)
4: fimalgoritmo
O working path é o vetor 1, 2, 3 e 4.
Já se o algoritmo é
```

```
1: algoritmo QUEBRA
2: se 3 > 5
3:   A ← A + 1
4: senão
5:   B ← 1
6:   C ← 2
7: fim{se}
8: A ← B + C
9: fimalgoritmo
O working path é o vetor 1, 2, 5, 6, 7, 8 e 9.
Se o algoritmo é
```

```
1: algoritmo QUEBRA
2: se 6 > 5
3:   A ← A + 1
4: senão
5:   B ← 1
6:   C ← 2
7: fim{se}
8: A ← B + C
9: fimalgoritmo
O working path é o vetor 1, 2, 3, 4, 7, 8 e 9.
```

Regras do Working Path

- 1. O comando algoritmo (ou função) sempre é o primeiro a ser executado.
- 2. O comando fimalgoritmo (ou fimfunção) é sempre o último a ser executado.
- 3. Não havendo salto, a um comando segue-se o seguinte.
- 4. Ao executar um comando SE, a condição é avaliada e daí: Se verdadeira, segue-se a execução dos comandos seguintes, até o SENÃO, que é executado também. Daí salta-se tudo até o FIMSE que é executado também. Não havendo SENÃO, executa-se o FIMSE apenas.

Se falsa, o próximo comando a ser executado é aquele posterior ao SENÃO (se houver SENÃO) até o FIMSE que é também executado. Se não houver SENÃO o próximo comando é o FIMSE

Exemplos Seja o algoritmo abaixo testado com A=4, B=2, C=2, D=6, E=1 e F=1.

```
1: algoritmo EX1
2: se (((~ (E = 5)) ∨ (C ≠ 5)) ∨ (D ≠ 6))
3:   se (((A > 4) ∧ (F ≥ 1)) ∧ (B ≠ 6))
4:     F ← F × C
```

```
5:   senão
6:     F ← C + 4
7:   fim{se}
8:   E ← F - B
9: senão
10:  A ← C - D
11:  se ((~ (D < 1)) ∨ (~ (B ≥ 4)))
12:    B ← E + B
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
Neste caso o WP é 1 2 3 6 7 8 9 14 15 16.
```

Mais um com A=1, B=3, C=4, D=3, E=4 e F=4.

```
1: algoritmo EX2
2: se (F > 4)
3:   se (~ (B ≠ 3))
4:     D ← A + 4
5:   senão
6:     C ← B + C
7:   fim{se}
8:   F ← E × 3
9: senão
10:  C ← F - B
11:  se (~ (C ≤ 2))
12:    E ← F + 2
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
O WP é 1 2 10 11 13 14 15 16.
```

Para você fazer

A seguir diversos algoritmos envolvendo apenas o comando de alternativa. Siga cada um dos algoritmos ache o WP, some os elementos do vetor e subtraia 48 deste total (para excluir os comandos 1, 2, 14, 15 e 16 que serão sempre executados).

Exercício 1 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 4, 9, 1, 4, 9, 9

```
1: algoritmo EX1
2: se (E ≤ 1)
3:   se (F > 6)
4:     A ← E + A
5:   senão
6:     D ← B - 4
7:   fim{se}
8:   B ← C × F
9: senão
10:  D ← F + B
11:  se (((~ (B ≠ 1)) ∨ (E > 6)) ∨ (E = 3))
12:    C ← D × A
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 2 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 3, 5, 1, 8, 9, 7

```
1: algoritmo EX2
2: se (B < 4)
3:   se (~ (D = 1))
4:     B ← A × E
5:   senão
6:     E ← F + 3
7:   fim{se}
8:   F ← D - 4
9: senão
10:  A ← B × C
11:  se ((E ≥ 3) ∧ (E = 1))
12:    A ← E - B
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 3 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F)

receberão respectivamente os valores: 4, 2, 1, 3, 4, 5

```
1: algoritmo EX3
2: se (C ≠ 4)
3:   se (((~ (B < 5)) ∧ (B < 4))
4:     F ← F + 5
5:   senão
6:     A ← B - 2
7:   fim{se}
8:   F ← E - 2
9: senão
10:  C ← C + A
11:  se (((C > 3) ∨ (~ (E > 4)))
12:    C ← A - 3
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 4 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 1, 9, 9, 7, 4, 9

```
1: algoritmo EX4
2: se (((~ (A ≥ 1)) ∨ (~ (D < 1)))
3:   se (A ≥ 2)
4:     A ← F × 5
5:   senão
6:     A ← A + B
7:   fim{se}
8:   A ← A + 4
9: senão
10:  D ← E + 2
11:  se ((A = 3) ∧ (~ (F = 2)))
12:    C ← E × A
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 5 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 6, 6, 9, 7, 5, 3

```
1: algoritmo EX5
2: se (((B ≥ 3) ∨ (A > 6)) ∧ (B > 2))
3:   se (((E ≠ 6) ∧ (~ (C > 4))) ∨ (~ (A = 6)))
4:     B ← C + A
5:   senão
6:     C ← C × E
7:   fim{se}
8:   F ← A - 4
9: senão
10:  A ← C × 5
11:  se (((~ (E ≠ 1)) ∧ (A > 4))
12:    F ← C + 5
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

Exercício 6 Suponha a execução do algoritmo abaixo, onde as 6 variáveis (A, B, C, D, E e F) receberão respectivamente os valores: 5, 8, 4, 1, 1, 1

```
1: algoritmo EX6
2: se (((C > 3) ∧ (~ (A > 3)))
3:   se (((~ (E ≤ 6)) ∧ (A ≥ 2))
4:     C ← D × 2
5:   senão
6:     B ← B + 5
7:   fim{se}
8:   A ← E - C
9: senão
10:  C ← B × 2
11:  se (D ≠ 5)
12:    F ← B + C
13:  fim{se}
14: fim{se}
15: escreva (A + B + C + D + E)
16: fim algoritmo
```

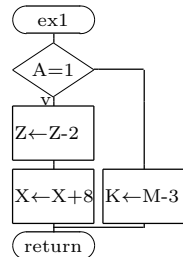
Respostas

1	2	3	4	5	6



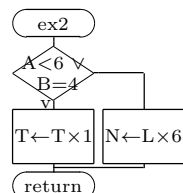
Fluxos e Pseudocódigo

A seguir, informe quais exercícios em que o trecho de pseudocódigo é 100% equivalente ao fluxograma apresentado



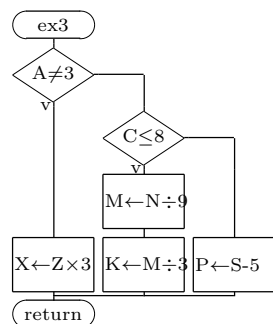
```

se (A≠1) entao
  K ← M-3
senao
  Z ← Z-2
  X ← X+8
fimse
  
```



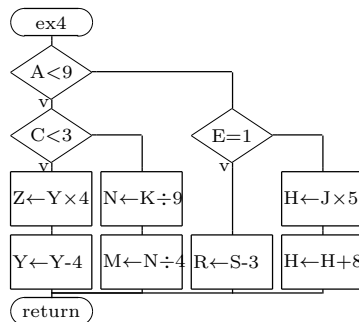
```

se (A<6) ∨ (B=4) entao
  N ← L×6
senao
  T ← T×1
fimse
  
```



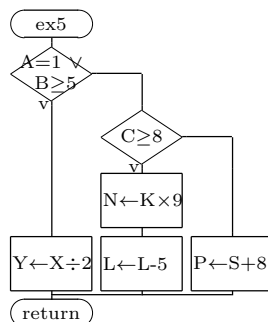
```

se (A=3) entao
  se (C>8) entao
    P ← S-5
  senao
    M ← N÷9
    K ← M÷3
  fimse
senao
  X ← Z×3
fimse
  
```



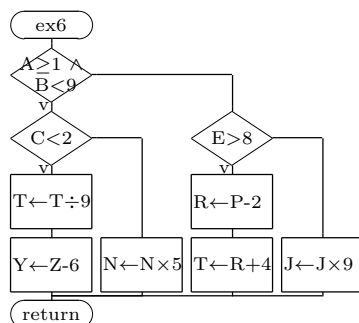
```

se (A<9) entao
  se (C<3) entao
    R ← S-3
  senao
    H ← J×5
    H ← H+8
  fimse
senao
  se (E=1) entao
    N ← K÷9
    M ← N÷4
  senao
    Z ← Y×4
    Y ← Y-4
  fimse
fimse
  
```



```

se (A≠1) ∧ (B<5) entao
  se (C<8) entao
    P ← S+8
  senao
    N ← K×9
    L ← L-5
  fimse
senao
  Y ← X÷2
fimse
  
```

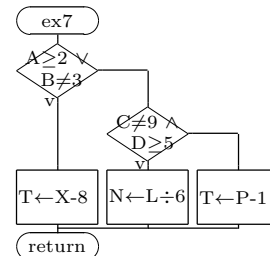


```

se (A<1) ∨ (B≥9) entao
  se (C<2) entao
  
```

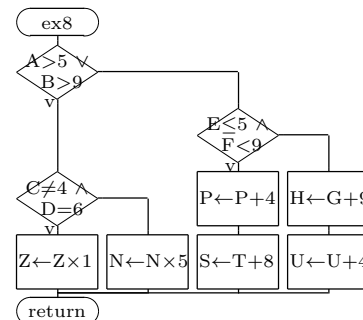
```

T ← T÷9
Y ← Z-6
senao
  N ← N×5
fimse
senao
  se (E≤8) entao
    R ← P-2
    T ← R+4
  senao
    J ← J×9
  fimse
fimse
  
```



```

se (A<2) ∧ (B=3) entao
  T ← X-8
senao
  se (C=9) ∨ (D<5) entao
    N ← L÷6
  senao
    T ← P-1
  fimse
fimse
  
```



```

se (A≤5) ∧ (B≤9) entao
  se (E>5) ∨ (F≥9) entao
    H ← G+9
    U ← U+4
  senao
    P ← P+4
    S ← T+8
  fimse
senao
  se (C=4) ∨ (D≠6) entao
    N ← N×5
  senao
    Z ← Z×1
  fimse
fimse
  
```

Responda se os trechos acima são equivalentes: (0=não, 1=sim)

1	2	3	4
5	6	7	8

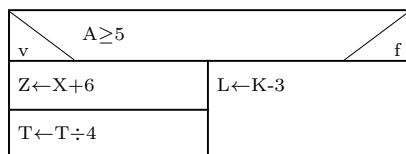


- 1

____ / ____ / ____

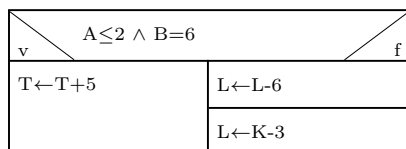
Nassi e Pseudocódigo: A seguir, informe quais exercícios em que o trecho de pseudo- código é 100% equivalente ao diagrama de Nassi apresentado

ex1 —



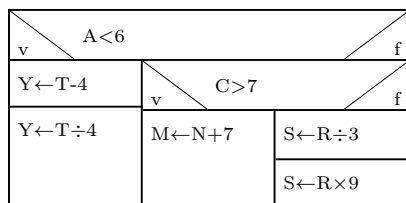
```
se (A<5) entao
  Z←X+6
  T←T÷4
senao
  L←K-3
fimse
```

ex2 —



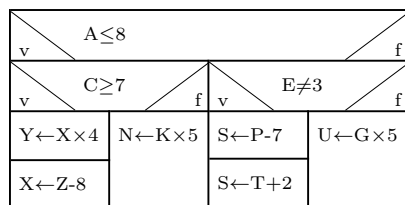
```
se (A≤2)^(B=6) entao
  L←L-6
  L←K-3
senao
  T←T+5
fimse
```

ex3 —



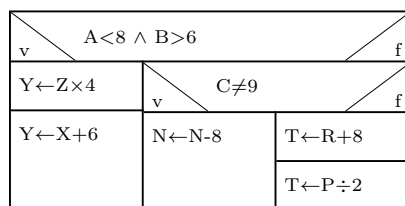
```
se (A<6) entao
  Y←T-4
  Y←T÷4
senao
  se (C≤7) entao
    M←N+7
  senao
    S←R÷3
    S←R×9
  fimse
fimse
```

ex4 —



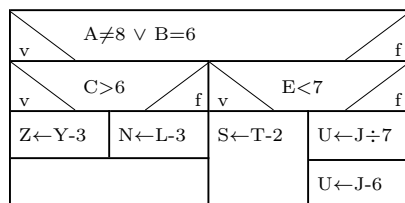
```
se (A>8) entao
  se (C≥7) entao
    Y←X×4
    X←Z-8
  senao
    N←K×5
  fimse
senao
  se (E=3) entao
    S←P-7
    S←T+2
  senao
    U←G×5
  fimse
fimse
```

ex5 —



```
se (A≥8)^(B≤6) entao
  Y←Z×4
  Y←X+6
senao
  se (C=9) entao
    N←N-8
  senao
    T←R+8
    T←P÷2
  fimse
fimse
```

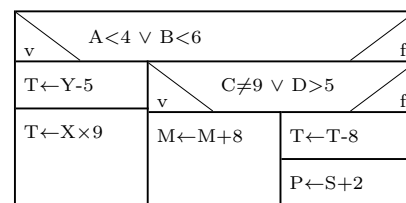
ex6 —



```
se (A≠8)^(B=6) entao
  se (C>6) entao
    S←T-2
  senao
```

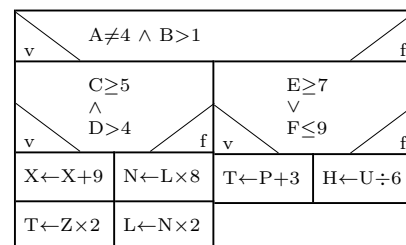
```
U←J÷7
U←J-6
fimse
senao
  se (E<7) entao
    N←L-3
  senao
    Z←Y-3
  fimse
fimse
```

ex7 —



```
se (A<4)^(B<6) entao
  T←Y-5
  T←X×9
senao
  se (C≠9)^(D>5) entao
    M←M+8
  senao
    T←T-8
    P←S+2
  fimse
fimse
```

ex8 —



```
se (A=4)^(B≤1) entao
  se (C≥5)^(D>4) entao
    X←X+9
    T←Z×2
  senao
    N←L×8
    L←N×2
  fimse
senao
  se (E<7)^(F>9) entao
    T←P+3
  senao
    H←U÷6
  fimse
fimse
```

Responda se os trechos acima são equivalentes: (0=não, 1=sim)

1	2	3	4	5	6	7	8



Nos algoritmos a seguir, você deve fazer um teste de mesa (chinês) com os dados que são fornecidos no final da página e inferir qual o resultado que o algoritmo apresentará após rodar.

1. Linha de Produção

Uma linha de produção produz peças que devem ter um certo índice de desempenho que é medido ao final da linha de produção. Dependendo deste resultado a peça recebe um selo que é apostado na peça a depender do índice, seguindo este algoritmo:

```
1: leia D {D=índ. desempenho}
2: se D > 5
3:   COR ← “vermelho”
4:   MSG ← “rejeitado”
5: senão
6:   se D > 3
7:     COR ← “amarelo”
8:     MSG ← “3. classe”
9:   senão
10:    COR ← “azul”
11:    se D > 1
12:      MSG ← “2. classe”
13:    senão
14:      MSG ← “1. classe”
15:    fim{se}
16:  fim{se}
17: fim{se}
18: escreva MSG, COR
```

2. Mérito ao funcionário

Uma empresa possui um enorme time de vendedores. Preocupada em reconhecer bons desempenhos, a empresa estabeleceu um processo de medição que no final do ano aponta merecedores de prêmios. Os dados de entrada do algoritmo são: VLV (valor total de vendas em milhares de reais), RC (reclamações de clientes em unidades) e ID (idade do vendedor em anos). A saída indica QTS (quantidade de salários a serem pagos), e a BF (quantidade de bônus adicionais de férias), seguindo o algoritmo:

```
1: leia VLV, RC, ID
2: QTS ← 13
3: BF ← 0
4: se VLV > 250
5:   se RC ≤ 2
6:     QTS ← QTS + 5
7:     BF ← BF + 3
8:   senão
9:     QTS ← QTS + 3
10:  fim{se}
11: fim{se}
12: se VLV > 100 ∧ VLV ≤ 250
13:   se RC ≤ 4
14:     QTS ← QTS + 4
15:     BF ← BF + 2
16:   senão
17:     QTS ← QTS + 2
18:  fim{se}
19: fim{se}
20: se ID > 40
21:   BF ← BF + 3
22: fim{se}
23: escreva QTS, BF
```

3. Taxa de Juros

O Banco Queromeu SA desenvolveu uma política agressiva de aquisição de clientes baseado em taxas decrescentes de juros para empréstimos. Dependendo de 3 variáveis (AC=anos em que o cliente está conosco em anos; VF=volume financeiro conosco em milhares de reais no último ano e PC=porte do cliente

que pode ser 1=pequeno, 2=médio, 3=grande e 4=muito grande) e da taxa referencial de juros (TRJ), o banco estabelece a taxa a cobrar (TC), seguindo o algoritmo:

```
1: leia AC, VF, PC, TRJ
2: se VF > 500
3:   se AC > 3
4:     se PC < 3
5:       TC ← TRJ × 1.1
6:     senão
7:       TC ← TRJ × 1.2
8:   fim{se}
9:   senão
10:    TC ← TRJ × 1.3
11:  fim{se}
12: senão
13:   se VF > 200
14:     se AC > 4
15:       TC ← TRJ × 1.4
16:     senão
17:       TC ← TRJ × 1.5
18:   fim{se}
19:   senão
20:     se VF > 50
21:       se PC > 3
22:         TC ← TRJ × 1.6
23:       senão
24:         TC ← TRJ × 1.7
25:     fim{se}
26:   senão
27:     TC ← TRJ × 1.8
28:   fim{se}
29: fim{se}
30: fim{se}
31: escreva TC
```

4. Preço de comercial

Uma certa estação de TV do interior do país desenvolveu um método inteligente de cobrar comerciais. A partir da duração do comercial em segundos (DCS), do horário da apresentação (HAP, que pode ser 1=manhã, 2=tarde, 3=nobre e 4=noite/madrugada) e da quantidade de inserções (QI, em vezes) ela estabelece o valor a cobrar em reais, usando o seguinte algoritmo:

```
1: leia DCS, HAP, QI
2: se DCS > 180
3:   DCS ← ⌊ DCS × 0.75 ⌋
4: fim{se}
5: se DCS > 60
6:   DCS ← ⌊ DCS × 0.9 ⌋
7: fim{se}
8: se ((HAP = 1) ∨ (HAP = 4))
9:   se QI > 3
10:    VAL ← DCS × QI × 76
11:   senão
12:    VAL ← DCS × QI × 86
13:   fim{se}
14: senão
15:   se HAP = 2
16:     se QI ≥ 3
17:       VAL ← DCS × QI × 96
18:     senão
19:       VAL ← DCS × QI × 106
20:   fim{se}
21:   senão
22:     VAL ← DCS × QI × 126
23:   fim{se}
24: fim{se}
25: escreva VAL
```

5. Decolagem possível

Como se sabe, muitas variáveis interferem na decolagem de um avião. São as principais: peso do avião, comprimento da pista e temperatura ambiente. Para um certo tipo de avião, a empresa de Transporte Aéreo Tô Indo usa o seguinte algoritmo: O Peso do avião (PAV) está dado em toneladas. O comprimento da pista disponível (CPI) está em m e a temperatura ambiente (TMB) está dado em graus centígrados. O algoritmo deve decidir se a decolagem é possível (1) ou não é possível (0) no caso apresentado.

```
1: leia PAV, CPI, TMB
```

```
2: A ← PAV × ⌊ (((TMB+20)÷40) × 1.0078) ⌋
3: se A < 40
4:   B ← 2000
5: senão
6:   se A < 80
7:     B ← 2000 + (⌊ ((A-40) ÷ 40) ⌋ × 200)
8:   senão
9:     B ← 2000 + (⌊ ((A-40) ÷ 60) ⌋ × 180)
10:  fim{se}
11: fim{se}
12: se B ≥ CPI
13:   escreva 0
14: senão
15:   escreva 1
16: fim{se}
```

6. Letras em ordem

Neste algoritmo, há 4 variáveis (A,B,C e D). Cada uma tem um valor entre 1 e 4. Após executar o algoritmo abaixo, as variáveis devem ser colocadas em ordem crescente (em caso de empate de duas ou mais variáveis, usar a ordem alfabética).

```
1: leia A,B,C,D
2: se A ≥ 3
3:   se C ≥ 3
4:     B ← B+1
5:     D ← D-2
6:   senão
7:     se D < 3
8:       B ← B+1
9:       A ← A-1
10:    senão
11:      B ← B-1
12:      A ← A+2
13:    fim{se}
14:  fim{se}
15: senão
16:   D ← D-2
17:   se B ≤ 2
18:     C ← C-1
19:   D ← D+2
20:   senão
21:     se D ≥ 2
22:       A ← A-2
23:       B ← B+1
24:     senão
25:       A ← A+1
26:       B ← B-2
27:     fim{se}
28:   fim{se}
29: fim{se}
30: escreva as variáveis em ordem crescente
```

7. Imposto devido

O Governo de um país longínquo, preocupado com as altas taxas de sonegação de impostos, resolveu substituir diversos impostos por um imposto único de renda que será cobrado das pessoas segundo o algoritmo a seguir. As entradas são a Renda Bruta (RB), o valor dos Abatimentos (AB) e o número de Dependentes (DP). A saída do algoritmo é o Valor do Imposto Devido (VI).

```
1: leia RB, AB, DP
2: se RB > 8000
3:   se RB > 15000
4:     se RB > 25000
5:       se AB > 10000
6:         VI ← 0
7:       senão
8:         se DP > 4
9:           VI ← 1500
10:        senão
11:          VI ← 2000
12:        fim{se}
13:      fim{se}
14:    senão
15:      se AB > 5000
16:        VI ← 0
17:      senão
18:        se DP > 3
19:          VI ← 800
20:        senão
21:          VI ← 1000
22:        fim{se}
```

```
23:   fim{se}
24:   fim{se}
25:   senão
26:     se AB > 2000
27:       VI ← 0
28:     senão
29:       se DP > 2
30:         VI ← 300
31:       senão
32:         VI ← 500
33:     fim{se}
34:   fim{se}
35: fim{se}
36: senão
37:   VI ← 0
38: fim{se}
39: escreva VI
```

Para você fazer

Linha de Produção

D	msg	cor
4		

Mérito ao Funcionário

VLV	RC	ID	QTS	BF
93	3	33		

Taxa de Juros

AC	VF	PC	TRJ	TC
5	192	2	100	

Preço do comercial

DCS	HAP	QI	VAL
210	4	4	

Decolagem Possível

PAV	CPI	TMB	n
58	1400	2	

Letras em ordem (entrada)

A	B	C	D
1	2	3	4

(saída)

1º	2º	3º	4º

Imposto Devido

RB	AB	DP	VI
17000	1000	2	



O comando SE e suas variações

Este exercício visa trabalhar o comando SE desde sua encarnação mais simples, até uma construção complexa conhecida como ninho de SEs. Na sua versão mais simples, o comando SE apresenta o seguinte formato:

```
se <condição> então
  bloco A
fimse
```

O que aparece como bloco A é um conjunto de comandos válidos que só é executado se a condição for verdadeira. Neste caso, se a condição é falsa, o bloco A é saltado e o fluxo de processamento segue após o comando "fimse". A palavra fimse é necessária uma vez que o bloco A pode ser formado por um comando, uma dezena ou até milhares de comandos.

Eis alguns exemplos, com os resultados esperados:

```
1: A ← 5
2: B ← 5
3: C ← 5
4: se ((A + B + C) é par)
5:   A ← A+C
6:   B ← B-3
7: fim{se}
8: escreva (A+B+C)
Cujos resultados são _____.
```

A lógica segue igual quando a condição que acompanha o comando fica mais sofisticada, como por exemplo quando a condição é composta.

Eis um novo exemplo:

```
1: A ← 6
2: B ← 4
3: C ← 9
4: se (A > B) ∨ ((B > A) ∧ (B é par))
5:   A ← A+C
6:   B ← B-1
7: fim{se}
8: escreva (A+B+C)
Cujos resultados são _____.
```

Agora, além do caminho positivo (quando a condição é verdadeira), tem-se também o comando completo com o caminho positivo (condição V) e o caminho negativo (quando a condição é falsa). O formato do comando agora é:

```
se <condição> então
  bloco A
senão
  bloco B
fimse
```

Note que este formato é muito parecido com o anterior, e olhando um contra o outro, pode-se dizer que é o mesmo, desde que se imagine ser a cláusula "senão" opcional. Assim, lembrando que cláusulas opcionais aparecem entre "[e]", esse formato único ficaria como

```
se <condição> entao
  bloco A
```

```
[ senão
  bloco B ]
fimse
```

```
Acompanhe este exemplo:
1: A ← 4
2: B ← 4
3: C ← 5
4: se (A ≠ B) ∨ (A+2 > C)
5:   A ← B-C
6:   B ← B-1
7: senão
8:   A ← C-4
9:   B ← A+1
10: fim{se}
11: escreva (A+B+C)
Cujos resultados são _____
```

Agora, coisas começam a ficar um pouco mais complexas, já que vai-se colocar um comando SE dentro de outro comando SE. A solução não é difícil, basta examinar e determinar rigorosamente quem é o então e quem é o senão de cada um dos comandos. Use o conceito de bloco, se ajudar. Veja-se o exemplo:

```
1: A ← 6
2: B ← 4
3: C ← 6
4: se (A = B) ∧ ((C-3) < B)
5:   se ((A mod B) ≠ C)
6:     A ← A+C
7:     B ← B-1
8:   fim{se}
9: senão
10:  se (A = B) ∧ ((C-3) < B)
11:    A ← A+C
12:    B ← B-1
13:  fim{se}
14: fim{se}
15: escreva (A+B+C)
Cujos resultados são _____
```

Finalmente há os casos em que dentro do comando condicional, há comandos que não são exatamente condicionais, ou seja, que serão ou não executados, dependendo de uma avaliação de onde estão. Veja no exemplo:

```
1: A ← 4
2: B ← 3
3: C ← 5
4: se (A > B) ∨ ((B > A) ∧ (B é par))
5:   A ← B+C
6:   se ((A + B) ≥ C)
7:     A ← A+C
8:     B ← B-3
9:   fim{se}
10:  B ← B-3
11: senão
12:  se (A = B) ∧ ((C-3) < B)
13:    A ← A+C
14:    B ← B-1
15:  fim{se}
16:  A ← C-2
17:  B ← A+1
18: fim{se}
19: escreva (A+B+C)
Cujos resultados são _____
```

Para você fazer:

```
1.
1: A ← 4
2: B ← 4
3: C ← 8
4: se (A = B)
5:   A ← A+C
6:   B ← B-3
7: fim{se}
8: escreva (A+B+C)
```

```
2.
1: A ← 5
2: B ← 5
3: C ← 9
4: se (A > B)
5:   A ← A+C
6:   B ← B-2
7: fim{se}
8: escreva (A+B+C)
```

```
3.
1: A ← 3
2: B ← 3
3: C ← 5
4: se (A = B) ∧ ((C-3) < B)
5:   A ← B-C
6:   B ← B-3
7: fim{se}
8: escreva (A+B+C)
```

```
4.
1: A ← 6
2: B ← 5
3: C ← 9
4: se (A = B) ∧ ((C-3) < B)
5:   A ← B-C
6:   B ← B-3
7: fim{se}
8: escreva (A+B+C)
```

```
5.
1: A ← 4
2: B ← 3
3: C ← 9
4: se ((A mod B) ≠ C) ∧ (A ≤ C)
5:   A ← B+C
6:   B ← B-1
7: senão
8:   A ← C-4
9:   B ← A+1
10: fim{se}
11: escreva (A+B+C)
```

```
6.
1: A ← 3
2: B ← 3
3: C ← 8
4: se (A = B) ∧ ((C-3) < B)
5:   A ← B+C
6:   B ← B-2
7: senão
8:   A ← C-3
9:   B ← A+3
10: fim{se}
11: escreva (A+B+C)
```

```
7.
1: A ← 5
2: B ← 4
3: C ← 7
4: se (A > B) ∨ ((B > A) ∧ (B é par))
5:   se (A = B)
6:     A ← A+C
7:     B ← B-3
8:   fim{se}
9: senão
10:  se ((A + B) ≠ C) ∧ (A ≠ B)
11:    A ← B-C
12:    B ← B-2
13:  fim{se}
14: fim{se}
15: escreva (A+B+C)
```

```
8.
1: A ← 6
2: B ← 5
3: C ← 5
```

```
4: se (A > B) ∨ ((B > A) ∧ (B é par))
5:   se (A > B)
6:     A ← A+C
7:     B ← B-1
8:   fim{se}
9: senão
10:  se ((A mod B) ≠ C) ∧ (A ≤ C)
11:    A ← B-C
12:    B ← B-2
13:  fim{se}
14: fim{se}
15: escreva (A+B+C)
```

```
9.
1: A ← 6
2: B ← 5
3: C ← 7
4: se (A > B) ∨ ((B > A) ∧ (B é par))
5:   A ← B+C
6:   se ((A + B) ≠ C)
7:     A ← A+C
8:     B ← B-3
9:   fim{se}
10:  B ← B-1
11: senão
12:  se ((A + B) ≠ C) ∧ (A ≠ B)
13:    A ← B-C
14:    B ← B-3
15:  fim{se}
16:  A ← C-1
17:  B ← A+1
18: fim{se}
19: escreva (A+B+C)
```

```
10.
1: A ← 3
2: B ← 5
3: C ← 5
4: se ((A + B) ≥ C) ∨ (A ≠ B)
5:   A ← B+C
6:   se ((A + B) ≠ C)
7:     A ← A+C
8:     B ← B-2
9:   fim{se}
10:  B ← B-2
11: senão
12:  se ((A + B) ≥ C) ∨ (A ≠ B)
13:    A ← B-C
14:    B ← B-1
15:  fim{se}
16:  A ← C-4
17:  B ← A+3
18: fim{se}
19: escreva (A+B+C)
```

```
11.
1: A ← 5
2: B ← 4
3: C ← 8
4: se (A = B) ∧ ((C-3) < B)
5:   A ← B+C
6:   se (A = B)
7:     A ← A+C
8:     B ← B-2
9:   fim{se}
10:  B ← B-3
11: senão
12:  se ((A + B) ≥ C) ∨ (A ≠ B)
13:    A ← B-C
14:    B ← B-2
15:  fim{se}
16:  A ← C-2
17:  B ← A+3
18: fim{se}
19: escreva (A+B+C)
```

1	2	3	4	5	6
7	8	9	10	11	
					xx
					xx
					xx

Algoritmos numéricos

Os exercícios a seguir estão indicados para desenvolver as habilidades de

- saber resolver condições lógicas
- adquirir destreza em avaliar comandos condicionais
- ganhar disciplina mental para seguir testes de mesa (chineses)

🔧 Para você fazer

Os exercícios a seguir implementam algoritmos. Em todos, as variáveis A, B e C são inicializadas no início do algoritmo. Você deve seguir o algoritmo até o final e neste ponto informar os valores de A, B e C.

Exercício 1 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 5
4: B ← 4
5: C ← 9
6: se (C = 4)
7:   se (C ≥ 1)
8:     se (A ≥ 1)
9:       A ← A - 4
10:    senão
11:      C ← C + 2
12:    fim{se}
13:  senão
14:    se (¬(B ≥ A))
15:      A ← B × C
16:    senão
17:      A ← A × B
18:    fim{se}
19:  fim{se}
20: senão
21:   se (¬(C ≥ B))
22:     A ← C × A
23:   senão
24:     C ← A × B
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Exercício 2 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 8
4: B ← 1
5: C ← 6
6: se (¬(A ≥ 3))
7:   se (¬(B < 3))
8:     se (C < 1)
9:       A ← B - 4
10:    senão
11:      B ← B + 2
12:    fim{se}
13:  senão
14:    se (B < C)
15:      A ← B × 3
16:    senão
17:      B ← B + 4
```

```
18:    fim{se}
19:  fim{se}
20: senão
21:   se (¬(C ≥ A))
22:     B ← A + C
23:   senão
24:     A ← A × A
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Exercício 3 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 5
4: B ← 8
5: C ← 4
6: se (¬(C = 6))
7:   se (B < C)
8:     se (C < 3)
9:       C ← C + C
10:    senão
11:      B ← A + 2
12:    fim{se}
13:  senão
14:    se (A ≥ C)
15:      C ← C + C
16:    senão
17:      B ← C + 3
18:    fim{se}
19:  fim{se}
20: senão
21:   se (C ≤ 1)
22:     A ← A + 4
23:   senão
24:     B ← A + 4
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Exercício 4 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 4
4: B ← 7
5: C ← 3
6: se (¬(C < B))
7:   se (¬(A ≤ C))
8:     se (¬(A < 6))
9:       C ← C - 5
10:    senão
11:      A ← B + 3
12:    fim{se}
13:  senão
14:    se (¬(A ≥ B))
15:      A ← A × B
16:    senão
17:      B ← B - 2
18:    fim{se}
19:  fim{se}
20: senão
21:   se (B < A)
22:     B ← B × B
23:   senão
24:     C ← A × 3
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Exercício 5 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 7
4: B ← 6
5: C ← 2
6: se (A > B)
7:   se (A ≤ B)
8:     se (B = 2)
9:       B ← B - B
10:    senão
11:      A ← A + B
12:    fim{se}
13:  senão
14:    se (B ≠ C)
15:      A ← A - C
16:    senão
17:      A ← C - 2
18:    fim{se}
19:  fim{se}
20: senão
21:   se (B < 2)
22:     A ← B - 4
23:   senão
24:     C ← A × 2
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Exercício 6 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 1
4: B ← 6
5: C ← 2
6: se (¬(C = 5))
7:   se (C > A)
8:     se (¬(C = 6))
9:       A ← C + 4
10:    senão
11:      C ← C × 2
12:    fim{se}
13:  senão
14:    se (¬(B ≠ C))
15:      A ← A × A
16:    senão
17:      C ← A × 5
18:    fim{se}
19:  fim{se}
20: senão
21:   se (B < A)
22:     B ← B - B
23:   senão
24:     C ← C + C
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Exercício 7 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 3
4: B ← 8
5: C ← 8
6: se (¬(B ≤ A))
7:   se (A < 2)
```

```
8:   se (B ≤ C)
9:     C ← C + 4
10:   senão
11:     A ← B + 3
12:   fim{se}
13: senão
14:   se (A < 4)
15:     C ← C - B
16:   senão
17:     A ← C × 5
18:   fim{se}
19: fim{se}
20: senão
21:   se (A ≥ C)
22:     C ← A + 3
23:   senão
24:     B ← B × A
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Exercício 8 Suponha a chamada ao algoritmo CALC, com a seguinte definição

```
1: algoritmo CALC
2: inteiro A, B, C
3: A ← 7
4: B ← 3
5: C ← 7
6: se (¬(A > 3))
7:   se (¬(C > A))
8:     se (¬(B ≤ C))
9:       B ← B + 4
10:    senão
11:      B ← B + 2
12:    fim{se}
13:  senão
14:    se (B = 5)
15:      B ← A - 5
16:    senão
17:      A ← B + 4
18:    fim{se}
19:  fim{se}
20: senão
21:   se (C > A)
22:     B ← B + 5
23:   senão
24:     B ← B + B
25:   fim{se}
26: fim{se}
27: imprima A, B, C
28: fim algoritmo
```

Respostas

Exerc 1 Var. C	Exerc 2 Var. B	Exerc 3 Var. C	Exerc 4 Var. C
Exerc 5 Var. A	Exerc 6 Var. A	Exerc 7 Var. C	Exerc 8 Var. B

Comandos SE-Exercício 1

A seguir, escreve-se um pequeno trecho de um programa no qual aparecem diversos comandos SEs aninhados. Estude o trecho

```
1: se ((C≠4)^(~(B=7)))
2:   se ((~(B=9))^(A>9))
3:     se (C=7)
4:       bola ← 'azul'
5:   senão
6:     bola ← 'creme'
7:   fim{se}
8: senão
9:   se ((C≤3)^(~(A≥6)))
10:     bola ← 'verde'
11:   senão
12:     bola ← 'tijolo'
13:   fim{se}
14: fim{se}
15: senão
16:   se (A≤9)
17:     bola ← 'preta'
18:   senão
19:     bola ← 'ocre'
20:   fim{se}
21: fim{se}
```

Para este caso, são pedidas duas coisas:
Qual a cor casos as variáveis fornecidas tenham o valor

A	B	C	D	F	G	
3	6	9	4	7	8	creme
→	→	→	→	→	→	
A	B	C	D	F	G	
5	2	7	8	6	3	azul
→	→	→	→	→	→	
A	B	C	D	F	G	
9	7	4	6	2	5	preta
→	→	→	→	→	→	

Qual deveria ser a cor obtida caso se escrevesse

```
se ((C ≠ 4) ^ (~ (B = 7))) ^ ((~ (B = 9)) ∨ (A > 9)) ^ (C = 7)
  bola ← azul
fim{se}
se ((C ≠ 4) ^ (~ (B = 7))) ^ ((~ (B ≠ 9)) ∨ (A > 9)) ^ (C ≠ 7)
  bola ← X (nada)
fim{se}
se ((C ≠ 4) ^ (~ (B = 7))) ^ ((B = 9) ^ (A ≤ 9)) ^ ((C ≤ 3) ∨ (~ (A ≥ 6)))
  bola ← verde
fim{se}
se ((C ≠ 4) ^ (~ (B = 7))) ^ ((B = 9) ^ (A ≤ 9)) ^ ((C > 3) ^ (A ≥ 6))
  bola ← tijolo
fim{se}
se ((C = 4) ∨ (B = 7)) ^ (A ≤ 9)
  bola ← preta
fim{se}
se ((C = 4) ∨ (B = 7)) ^ (A > 9)
  bola ← ocre
fim{se}
```

Exercício 2

No exercício 2 é dado um trecho simples de código envolvendo comandos SE...ENTÃO...FIMSE e que manuseia 3 variáveis de nomes A, B e C. Cada uma delas recebe um valor especificado (dado no problema) antes dos comandos condicionais serem executados. Ao final, as variáveis A, B e C tem (ou não) seus valores alterados. Daí, você deve somar os 3 valores finais e responder ao exercício. Note que cada exercício tem 6 instâncias diferentes a resolver. Os valores iniciais do exemplo devem ser buscados na tabela:

A	B	C
4	5	2
5	3	2
4	3	6
4	3	5
6	4	5
6	3	2

```
1: se ((~(A<6))^(B>7))
2:   se ((~(C≤3))^(B≥8))
3:     se (C≥7)
```

```
4:   A←C-2
5:   senão
6:     B←B-2
7:   fim{se}
8:   senão
9:     se ((~(C≤7))^(A≠6))
10:       B←C+1
11:       A←B+1
12:       A←A+2
13:     fim{se}
14:   fim{se}
15: senão
16:   se (A<5)
17:     C←C+1
18:     C←C+1
19:     A←C-2
20:   fim{se}
21: fim{se}
```

Se os dados de entrada fossem

Na entrada			Na saída			Σ
A	B	C	A	B	C	
4	5	2	2	5	4	11
5	3	2	5	3	2	10
4	3	6	6	3	8	17
4	3	5	5	3	7	15
6	4	5	6	2	5	13
6	3	2	6	3	2	11

Depois de obtidos os 3 valores no final do código os mesmos devem ser somados e é este valor que vai ser corrigido. Ele aparece em negrito no exemplo acima.

🔗 Para você fazer

Exercício 1

```
1: se ((A>3)^(~(F≤4)))
2:   se ((B≤5)^(D≥2))
3:     se (A≥8)
4:       bola ← 'roxa'
5:   senão
6:     bola ← 'tijolo'
7:   fim{se}
8:   senão
9:     se ((~(A>7))^(D=4))
10:       bola ← 'preta'
11:   senão
12:     bola ← 'zebrada'
13:   fim{se}
14: fim{se}
15: senão
16:   se (A≤5)
17:     bola ← 'laranja'
18:   senão
19:     bola ← 'incolor'
20:   fim{se}
21: fim{se}
```

Para o código acima, qual deveria ser a cor, caso as variáveis tivessem os seguintes valores

cor	A	B	C	D	F	G
	5	1	4	2	7	9
	3	5	2	1	9	6
	9	2	1	7	4	3

Ainda imaginando o código acima, qual cor deveria ser alocada à variável bola, se o teste fosse escrito em um único SE. Se nenhuma cor for a correta, responda "X".

```
se ((A ≥ 3) ∨ (~ (F ≤ 4))) ^ ((B > 5) ∨ (D < 2)) ^ ((~ (A > 7)) ^ (D = 4))
  bola ← _____
fim{se}
se ((A ≥ 3) ∨ (~ (F ≤ 4))) ^ ((B ≤ 5) ^ (D ≥ 2)) ^ (A ≥ 8)
  bola ← _____
fim{se}
se ((A ≥ 3) ∨ (~ (F ≤ 4))) ^ ((B ≤ 5) ^ (D ≥ 2)) ^ (A < 8)
  bola ← _____
fim{se}
se ((A < 3) ^ (F ≤ 4)) ^ (A > 5)
  bola ← _____
fim{se}
```

Exercício 2

```
1: se ((C>3)^(A<7))
2:   se ((B≠4)^(A>3))
3:     se (C≥7)
4:       A←A+1
5:       A←C+2
6:       C←C-2
7:     fim{se}
8:   senão
9:     se ((B≥2)^(~(C=4)))
10:       A←C+2
11:       C←B-1
12:       C←A+1
13:     fim{se}
14:   fim{se}
15: senão
16:   se (C≥4)
17:     A←A-2
18:     B←B-1
19:     B←A+2
20:   fim{se}
21: fim{se}
```

Com os seguintes dados de entrada

instância	A	B	C
1	5	4	2
2	3	4	2
3	5	4	6
4	3	5	2
5	2	5	6
6	6	3	4

Para achar a resposta, considere as variáveis fornecidas no início do algoritmo e execute todas as operações que tais variáveis sofrem. Ao final, SOME O VALOR DAS 3 VARIÁVEIS e coloque a soma achada na resposta equivalente ao exercício.

1	2	3	4	5	6

Mais um do ex. 2

```
1: se ((A≥5)^(B=6))
2:   se ((B=3)^(C≠6))
3:     se (A<3)
4:       A←A+1
5:   senão
6:     C←C+2
7:   fim{se}
8:   senão
9:     se ((C≤5)^(A<5))
10:       B←C+1
11:       B←B+2
12:       B←B-1
13:     fim{se}
14:   fim{se}
15: senão
16:   se (~(A≤5))
17:     A←B-1
18:   senão
19:     A←C+1
20:   fim{se}
21: fim{se}
```

Com os seguintes dados de entrada

instância	A	B	C
1	3	6	5
2	6	4	5
3	3	4	5
4	2	4	6
5	2	3	6
6	2	4	3

Para achar a resposta, considere as variáveis fornecidas no início do algoritmo e execute todas as operações que tais variáveis sofrem. Ao final, SOME O VALOR DAS 3 VARIÁVEIS e coloque a soma achada na resposta equivalente ao exercício.

1	2	3	4	5	6

Comandos de Repetição

Enquanto Serve para iterar (repetir) partes de código de programação dependendo de alguma condição. Seu formato é

```
1: enquanto condição
2:   comando 1
3:   comando 2
4:   ...
5: fim{enquanto}
```

Quando o comando enquanto é encontrado a condição é avaliada. Se for falsa, ocorre um desvio para o próximo comando após o fimenquanto. Se for verdadeira, os comandos seguintes são executados e quando o comando fimenquanto é achado, ocorre um desvio para o enquanto (e a condição associada é avaliada de novo. Por exemplo:

```
1: J ← 5
2: enquanto J < 16
3:   escreva(J)
4:   J ← J + 3
5: fim{enquanto}
```

Este trecho dará origem às seguintes impressões: 5, 8, 11 e 14.

Repita Serve para iterar (repetir) partes de código de programação dependendo de alguma condição. Seu formato é

```
1: repita
2:   comando 1
3:   comando 2
4:   ...
5: até condição
```

Quando o comando repita é encontrado ele simplesmente é ignorado e os comandos seguintes são executados normalmente. Quando o comando até é achado, a condição que o acompanha é avaliada. Se for verdadeira, ocorre um desvio para o comando seguinte ao do até. Se for falsa, ocorre um desvio para o comando repita no alto. Por exemplo:

```
1: J ← 5
2: repita
3:   escreva(J)
4:   J ← J + 3
5: até J > 16
```

Este trecho dará origem às seguintes impressões: 5, 8, 11 e 14. Note que para obter os mesmos resultados foi necessário inverter o sinal da condição (de < para >). Este comando tal como foi descrito é aquele da linguagem PASCAL e da maioria dos processadores de algoritmos. Para C, o comando é do { ... } while(condição), com o detalhe é que a saída é quando a condição é FALSA.

Para Este comando serve para repetir (iterar) com bastante controle sobre a quantidade de repetições. Só pode ser usado quando se conhece a quantidade de repetições necessária. Seu formato:

```
1: para variável FROM valor-1 TO
   valor-2 [PASSO valor-3]
2:   comando-1
3:   comando-2
4:   ...
5: fim{para}
```

A variável de controle é inicializada com o valor-1. A seguir, ela é testada contra o valor-2. Se a variá-

vel é ≤ ao valor, o conjunto de comandos é executado. Senão desvia-se para o comando seguinte ao fimpára. Quando o comando fimpára é encontrado, a variável de controle é incrementada com o valor-3, e ocorre um desvio para o início do comando.

Se o valor-3 é negativo, a variável de controle é decrementada a cada laço e o teste de parar/continuar é: ≥.

Acompanhe no exemplo a seguir:

```
1: para J FROM 4 TO 12 PASSO 3
2:   escreva J
3: fim{para}
```

Serão impressos: 4, 7, 10 e 12. O mesmo programa, agora escrito com enquanto, fica:

```
1: J ← 4
2: enquanto J ≤ 12
3:   escreva J
4:   J ← J + 3
5: fim{enquanto}
```

Alguns exemplos Similares aos exercícios que terão que ser feitos à frente.

```
1: funcao Exercicio1
2: m ← 7
3: enquanto m < 28
4:   escreva (m)
5:   m ← m+4
6: fim{enquanto}
7: fimfuncao
```

Cuja resposta é:

```
1: funcao Exercicio2
2: para m de 8 ate 26 passo 5
3:   escreva (m)
4: fim{para}
5: fimfuncao
```

Cuja resposta é:

```
1: funcao Exercicio3
2: m ← 5
3: enquanto m < 17
4:   para k de 5 ate 9 passo 3
5:     escreva (m+k)
6:   fim{para}
7:   m ← m+5
8: fim{enquanto}
9: fimfuncao
```

Cuja resposta é:

```
1: funcao Exercicio4
2: m ← 2
3: enquanto m < 17 faca
4:   se m e divisivel por 3
5:     escreva (m)
6:   senão
7:     escreva (m ×3)
8:   fim{se}
9:   m ← m + 2
10: fim{enquanto}
11: fimfuncao
```

Cuja resposta é:

```
1: funcao Exercicio5
2: j ← 2
3: enquanto j < 15
4:   k ← 7
5:   enquanto k < 8
6:     se k mod 3 = 1
7:       escreva k-j
8:     senão
9:       escreva k+j
10:   fim{se}
11:   k ← k + 3
12:   fim{enquanto}
13:   j ← j + 2
14: fim{enquanto}
15: fimfuncao
```

Cuja resposta é:

🔧 Para você fazer

Para os exercícios a seguir, descubra a sequência de números impressos e registre no gabarito o LTIMO deles. Note que todas as séries tem menos de 11 números. Se este número for alcançado, significa que há algum erro de interpretação.

Exercício 1

```
1: funcao Exercicio1
2: m ← 7
3: enquanto m < 27
4:   escreva (m)
5:   m ← m+ 5
6: fim{enquanto}
7: fimfuncao
```

Exercício 2

```
1: funcao Exercicio2
2: para m de 8 ate 26 passo 6
3:   escreva (m)
4: fim{para}
5: fimfuncao
```

Exercício 3

```
1: funcao Exercicio3
2: m ← 7
3: enquanto m < 19
4:   para k de 3 ate 8 passo 2
5:     escreva (m+k)
6:   fim{para}
7:   m ← m+ 5
8: fim{enquanto}
9: fimfuncao
```

Exercício 4

```
1: funcao Exercicio4
2: m ← 3
3: enquanto m < 17 faca
4:   se m e divisivel por 2
5:     escreva (m)
6:   senão
7:     escreva (m ×2)
8:   fim{se}
9:   m ← m + 4
10: fim{enquanto}
11: fimfuncao
```

Exercício 5

```
1: funcao Exercicio5
2: j ← 2
3: enquanto j < 12
4:   k ← 6
5:   enquanto k < 9
6:     se k mod 2 = 0
7:       escreva k-j
8:     senão
9:       escreva k+j
10:   fim{se}
11:   k ← k + 2
12:   fim{enquanto}
13:   j ← j + 4
14: fim{enquanto}
15: fimfuncao
```

Exercício 6

```
1: funcao Exercicio6
2: para i de 4 a 15 passo 4
3:   para k de 5 a 09 passo 2
4:     imprima i+k
5:   fim{para}
6: fim{para}
7: fimfuncao
```

Exercício 7

```
1: funcao Exercicio7
2: j ← 2
3: enquanto j < 15
4:   k ← 13
5:   enquanto k > 6
6:     escreva j + k
7:     k ← k - 4
8:   fim{enquanto}
9:   j ← j + 4
10: fim{enquanto}
11: fimfuncao
```

Exercício 8

```
1: funcao Exercicio8
2: m ← 7
```

```
3: enquanto m < 30
4:   escreva (m)
5:   m ← m+ 5
6: fim{enquanto}
7: fimfuncao
```

Exercício 9

```
1: funcao Exercicio9
2: para m de 6 ate 30 passo 5
3:   escreva (m)
4: fim{para}
5: fimfuncao
```

Exercício 10

```
1: funcao Exercicio10
2: m ← 6
3: enquanto m < 17
4:   para k de 4 ate 7 passo 2
5:     escreva (m+k)
6:   fim{para}
7:   m ← m+ 4
8: fim{enquanto}
9: fimfuncao
```

Exercício 11

```
1: funcao Exercicio11
2: m ← 2
3: enquanto m < 16 faca
4:   se m e divisivel por 2
5:     escreva (m)
6:   senão
7:     escreva (m ×2)
8:   fim{se}
9:   m ← m+ 3
10: fim{enquanto}
11: fimfuncao
```

Exercício 12

```
1: funcao Exercicio12
2: j ← 4
3: enquanto j < 12
4:   k ← 6
5:   enquanto k < 9
6:     se k mod 2 = 0
7:       escreva k-j
8:     senão
9:       escreva k+j
10:   fim{se}
11:   k ← k + 4
12:   fim{enquanto}
13:   j ← j + 4
14: fim{enquanto}
15: fimfuncao
```

Responda a seguir, os últimos números de cada sequência:

1	2	3	4	5	6
7	8	9	10	11	12



Algoritmos de Dígitos Verificadores

Erros possíveis em digitação	era	foi digitado
obliteração	1234	123
repetição	1234	12234
transposição	1234	1324
substituição	1234	7234
fraude	1234	5678

O conceito de Dígito Verificador (DV) é usado para diminuir os erros de entrada de dados em sistemas computacionais. Pela utilização do DV, os códigos utilizados passam a ter uma lógica interna de funcionamento, o que permite determinar incorreções (inversões, obliterações, duplicações, invenção de códigos, etc) simplesmente “olhando” para o código sem necessidade de pesquisar em nenhuma entidade externa.

Pelo uso de um único DV, apenas um entre 10 códigos “inventados” estará correto e pelo uso de 2 DVs, apenas 1 em 100 estará correto.

Como se viu, pela simples análise do DV é possível detectar grande parte dos erros acima descritos.

A função mais usada é MOD, já que é uma função de mão única. (não tem inversa). Por exemplo DOBRO de 2 é 4. Logo A METADE de 4 é 2. Já 8 MOD 7 é 1. Mas não é possível determinar x em X mod 7 = 1. x pode ser 8, 22, 29,...

Código de produtos - EAN13 Começou como UPC (código universal de produto, iniciativa americana de 12 dígitos que atribuía 6 dígitos para o fabricante, 5 para o produto mais o DV). Mais tarde o sistema evoluiu para uma iniciativa mundial, usando-se a codificação EAN-13, originalmente chamada European Article Number, atual International Article Number, mas que manteve a sigla. Nesta padronização, usam-se:

país os 3 primeiros dígitos indicam o país (789=Brasil, 784=Paraguai, 560=Portugal, etc)

fabricante os próximos 4, 5 ou 6 dígitos

produto os dígitos 5, 4 ou 3 seguintes

dígito verificador calculado sempre segundo o algoritmo:

- Some os códigos que ocupam posição MPA
- Some os códigos que ocupam posição PAR e multiplique esta soma por 3
- Some as duas parcelas acima
- o dígito é quanto faltar para chegar a um múltiplo de 10

Observação: Se a soma já for múltipla de 10, o DV é zero.

Este código é gerado pela organização multinacional GS1 (www.gs1.org) com sede na Bélgica. Sua perna brasileira é a GS1 Brasil (www.gs1br.org) que nasceu como ABAC.

ISBN 10 (International Standard Book Number) O último dígito da série de 10 do ISBN é o DV. Ele é calculado de maneira a que multiplicando cada dígito do código pela sua posição (começando da direita e em 1) e somando tudo, o resto desta soma por 11 deve ser 0.

Acompanhe o exemplo: Seja o ISBN 85-7001-926-? (idioma-editor-livro-dv). Multiplica-se 8 por 10, 5 por 9, 7 por 8, 1 por 5, 9 por 4, 2 por 3, 6 por 2 e o resultado é 240.

Dividindo 240 por 11 tem-se 21,8, e portanto o próximo inteiro divisível por 11 é 11 vezes 22, que é 242.

Fazendo-se 242-240=2 que é o DV procurado. O ISBN 13 (em uso a partir de Janeiro de 2007) gera seu dígito da mesma maneira que o EAN13.

Códigos de cartão de crédito Usa-se o algoritmo de Luhn.

O algoritmo de Luhn (Hans Peter Luhn, funcionário da IBM, 1896-1964), também conhecido como módulo 10, foi desenvolvido nos anos 60, como um método para validar códigos. Ele é usado nos números de cartão de crédito e no código de seguro social do Canadá. O algoritmo é de domínio público. Ele protege contra o erro acidental e não contra o ataque malicioso.

Para testar o algoritmo de Luhn, calcule o DV do cartão 4931 4701 2604 479?. A resposta deve ser 2.

Este algoritmo também é usado no sistema bancário da Noruega, no código ISSN (periódicos), no número de identificação do veículo em alguns estados americanos, no cartão de identificação de israelenses e no Yugoslav Unique Master Citizen Number (JMBG).

Os principais métodos usados:

Módulo 10

- Separe os dígitos do código a processar. Ex: se o código é 13865, tem-se $d_1 = 1, d_2 = 3, d_3 = 8, d_4 = 6$ e $d_5 = 5$.
- Crie um vetor de pesos P, com o mesmo número de dígitos, e contendo 2 e 1, começando com 2 e alternando. Ex: $p_1 = 2, p_2 = 1, p_3 = 2, p_4 = 1, p_5 = 2$.
- Multiplique os dois vetores, tirando NOVES FORA em cada multiplicação. Ex: $m_1 = 1 \times 2 = 2, m_2 = 3 \times 1 = 3, m_3 = 8 \times 2 = 16$, noves fora=7, $m_4 = 1 \times 6 = 6, m_5 = 2 \times 5 = 1$
- Some os elementos do vetor multiplicação. Ex: $2 + 3 + 7 + 6 + 1 = 19$
- O que faltar para completar a próxima dezena será o DV mod 10. No exemplo, o que falta a 19 para completar 20 é 1. Logo DV = 1.
Obs: 1: noves fora: se $x > 9, x \leftarrow x - 9$;
2: se deu a dezena, a resposta é 0.

Módulo 11

- Separe os dígitos.
- Crie vetor P. O menor peso é 2, alocado ao dígito mais à direita. A sequência de pesos cresce para a esquerda. Ex: 7, 6, 5, 4, 3, 2. A lei de formação destes pesos PODE VARIAR em cada método.
- O DV = 11 - resto da soma dividido por 11.
- Se o resto é 0 ou 1, o DV é igual a 0. (No BB se R=0, DV=X)

O método da letra chave (em desuso) tem como resposta uma letra e não um dígito. Ao invés de dividir a soma por 11, divide por 26 e usa a correspondência de letras (0=A, 1=B, 2=C, ... , 25=Z). Foi usado na Receita do PR até 96.

CPF (Cadastro de Pessoas Físicas)

- a) São 2 DVs.
- b) O primeiro é calculado pelo MOD 11, com pesos = 10, 9, 8, 7, 6, 5, 4, 3, 2.
- c) O DV calculado é colocado no seu lugar e o processo refeito, agora com os pesos: 11, 10, 9, 8, 7, 6, 5, 4, 3, 2.
- d) O segundo dígito é colocado no seu lugar.
- Obs: O nono dígito de um CPF é a região onde foi criado. 6=MG, 7=ES/RJ, 8=SP, 9=PR/SC e 0=RS. Por exemplo, seja o CPF = 176.294.338, quais os DVs ? $1 \times 10 + 7 \times 9 + 6 \times 8 + 2 \times 7 + 9 \times 6 + 4 \times 5 + 3 \times 4 + 3 \times 3 + 8 \times 2 = 246$, cuja divisão por 11 tem como resto 4. Logo o primeiro DV é $11 - 4 = 7$. Refazendo $1 \times 11 + 7 \times 10 + 6 \times 9 + 2 \times 8 + 9 \times 7 + 4 \times 6 + 3 \times 5 + 3 \times 4 + 8 \times 3 + 7 \times 2 = 303$, cujo resto da divisão por 11 é 6. Assim, o DV é $11 - 6 = 5$. O CPF completo fica sendo 176.294.338 - 75.
- Curiosidade: CPF = 111 111 111, DVs=1,1. CPF = 222 222 222 - 22, 333 333 333 - 33, 444 444 444 - 44, e assim por diante até 999 999 999 - 99 e 000 000 000 - 00.

Veja se acerta:
CPF = 357 432 754 - _____ - _____. Resposta: 40.
CPF = 247 212 764 - _____ - _____. Resposta: 27.

CNPJ (antigo CGC) a) São 3 DVs.
b) O DV1 ocupa a 8.ª posição do código e é um Módulo 10 dos 7 dígitos iniciais.
c) O DV2 ocupa a 13.ª posição e é um Módulo 11 de todos os anteriores com o vetor de pesos = 543298765432. d) O DV3 ocupa a 14.ª posição e é um Módulo 11 de todos os anteriores com os pesos = 6543298765432.

Exemplo, seja o CNPJ da COPEL: 7 648 381. O DV1 = $7 \times 2 = 14$, noves fora = $5 + 6 \times 1 = 6, +4 \times 2 = 8, +8 \times 1 = 8, +3 \times 2 = 6, +8 \times 1 = 8, +1 \times 2 = 2$. A soma é 43, e o DV1 é o que falta para a próxima dezena, NESTE CASO 50, ou DV1=7. Com isso o primeiro código é: 76.483.817. Continuando:

A filial é 0001, e fica: 76.483.817/0001. O DV2 é igual a: $= 7 \times 5 + 6 \times 4 + 4 \times 3 + 8 \times 2 + 3 \times 9 + 8 \times 8 + 1 \times 7 + 7 \times 6 + 0 \times 5 + 0 \times 4 + 0 \times 3 + 1 \times 2 = 229$. O DV2 é 11 menos o resto de 229 por 11 que é 2. DV3= $7 \times 6 + 6 \times 5 + 4 \times 4 + 8 \times 3 + 3 \times 2 + 8 \times 9 + 1 \times 8 + 7 \times 7 + 0 \times 6 + 0 \times 5 + 0 \times 4 + 1 \times 3 + 2 \times 2 = 254$. O resto de 254 dividido por 11 é 1, logo o DV3=11-1=10 ou zero. Assim, o CNPJ completo da COPEL é: 76.483.817/0001-20.

Veja se acerta os CNPJs a seguir:
Carrefour: 45.543.91_____/0001 - ____ - _____. Resposta: 581.
Lacta: 57.003.88_____/0061- ____ - _____. Resposta: 152.
BC: 00.038.16_____/0001- ____ - _____. Resposta: 605.

Observação: Alguns CNPJs inexplicavelmente não seguem o aqui escrito. Aparentemente foram criados antes da regra. Por exemplo, o CGC da Light: 66 444 437/0001-46.

Para você fazer

Calcule os dígitos dos Cadastros de Pessoa Física:

1= 4 1 8 4 6 1 4 6 0 - _____.
2= 9 2 8 4 1 8 1 8 0 - _____.
Calcule os dígitos dos seguintes CNPJs:

3= 2 4 4 4 7 8 4 ____ / 0 0 6 1 - _____.
4= 3 9 1 9 2 3 0 ____ / 0 0 1 4 - _____.
 - 1 - /

Uma grande sacada foi a construção de um sistema integrado de cobrança através de um bloqueto padronizado. Por meio dele, qualquer agência bancária pode receber qualquer conta, não importa quem seja o emissor, já que o bloqueto é compensável. Antes deste sistema, consumidores precisavam fazer uma romaria pelos bancos, pagando a cada um as contas emitidas por ele. Agora, – pelo menos até o vencimento – todas as contas podem ser pagas em qualquer agência, possivelmente na agência onde o pagante tiver conta.

- Local de pagamento;
- Data de vencimento;
- Cedente (nome) - quem vai receber o dinheiro;
- Agência/Código do cedente - onde;
- Data do processamento - quando o boleto foi processado/impresso;
- Nosso número - como o cedente vai identificar o que/quem pagou;
- Valor do documento;
- Sacado (nome e endereço completo) - quem deve pagar.

mada pelas seguintes informações:

3	identificação do banco (BBB)
1	moeda (9=real) (M)
4	fator de vencimento (1000=3/7/2000) (FFFF)
10	valor (incluindo 2 decimais) (VVVVVVVVVV)
25	campo livre para identificar o pagamento (LL..LL)
1	DV calculado pelo módulo 11 (D)

O campo livre é formado por:

4	agência do cedente (sem DV)
2	código da carteira
11	nosso número
7	conta corrente do cedente
1	fixo igual a zero

Estas informações estão dispostas em 2 lugares: no alto do bloqueto em dígitos decimais prontos para serem digitados e também na parte inferior do bloqueto na forma de código de barras prontos para captura automatizada.

É importante notar que:

- As duas séries de dados não estão na mesma ordem
- O campo numérico possui 3 dígitos verificadores adicionais que são altamente recomendados para evitar erros de digitação.

A linha digitável tem a seguinte estrutura:

3	banco
1	moeda
5	posições 1-5 do campo livre
1	DV1 (módulo 10) das 9 posições anteriores
10	posições 6-15 do campo livre
1	DV2 (módulo 10) das 10 posições anteriores
10	posições 16-25 do campo livre
1	DV3 (módulo 10) das 10 posições anteriores
1	DV geral (módulo 11)
4	fator de vencimento
10	valor (com 2 decimais incluídas)

mento:

fator	vencimento
1000	03/07/2000
1002	05/07/2000
1667	01/05/2002
4789	17/11/2010
9999	21/02/2025
10313	01/01/2026

O cálculo do DV: Dispõe-se os dados do bloqueto como segue

[illegible]

e multiplica-se cada dígito por

4329876543298765432987654329876543298765432

Soma-se tudo e divide-se por 11, obtendo um resto. O DV é 11 menos o resto calculado, notando-se que o DV=1 para os restos 0, 10 ou 1. O DV é colocado na posição 5 do código de barras e no quarto campo (logo antes do fator de vencimento) na linha digitável.

O cálculo dos DVs parciais: Multiplica-se os dígitos do campo pela série 2,1,2,... começando pela direita com o valor 2. Antes de somar, deve-se aplicar a regra do “noves fora” (valores maiores do que 9 devem perder 9). Ou dito de outra maneira, somas com 2 dígitos devem ser substituídos pela soma dos 2 dígitos. Seja como for, no final, soma-se tudo. O DV é quanto falta para a próxima dezena inteira.

Exemplos

```

DV='3419489700000445001750000056413835570007000'
  é 8
DV1='0597255789983383204066891833962642336508247'
  é 3
DV='3040427768317807891654220743649745417967553'
  é 9
DV1='341917500' é 9
DV2='0005641383' é 4 e
DV3='5570007000' é 0
DV2='6361840992' é 2
DV1='405100184' é 2

```

 Para você fazer

1. Ache o fator de vencimento de um bloqueto que vai vencer em

28/03/2026

2. Ache o DV parcial de um conteúdo igual a

5674004485

3. Ache o DV completo de um conteúdo igual

5710499355239866595322987842018226097087650

fator	DV _{parc}	DV



Eis algumas datas e seus fatores de venci-

Aritmética Complexa

Na longa história da Matemática, aconteceu diversas vezes a necessidade de criar novos conjuntos de números diante de alguma demanda especial, senão vejamos:

A contagem: Parece consensual que a origem de todo o raciocínio matemático está na operação de contagem (de animais, de filhos, de esposas, de dias até a época da colheita, etc). Esta operação dá origem ao conjuntos dos números naturais ($\mathbb{N}$). Até aqui, a operação de adição está perfeitamente definida: *Tenho 3 galinhas e recebo mais 2, fico com 5.* Ou em termos matemáticos $3 + 2 = 5$.

A dívida: Se alguém pegou emprestado 5 galinhas e posteriormente devolveu apenas 3, fica um saldo de 2, ou como ele é devedor -2 , ou em termos matemáticos $3 - 5 = -2$. Então para atender à operação de subtração eventualmente serão necessários outros números, no caso os *negativos*. Agora juntando os naturais com os negativos, chega-se ao conjunto dos inteiros $\mathbb{Z}$, a origem da letra é *devida ao idioma alemão Zahl, que significa "número". O símbolo foi introduzido pelo matemático alemão Carl Friedrich Gauss (1777-1855), em seu livro Disquisitiones Arithmeticae, publicado em 1801.* Claro está que $\mathbb{N} \subset \mathbb{Z}$.

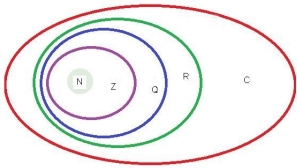
A divisão: Se tenho 2 vacas e 3 filhos, quando chega a hora de dividir o patrimônio, temos um problema: Cada filho deve receber $2/3$ de vaca, o que claramente não é um número inteiro. Para atender a esta operação, faz-se necessário expandir os números surgindo um novo conjunto, os Racionais (ou $\mathbb{Q}$ da palavra Quociente). A definição formal deste conjunto é $\mathbb{Q} = \{a/b, \forall a \in \mathbb{Z} \text{ e } b \neq 0\}$. A ressalva final é importante porque nunca é demais lembrar que estamos terminantemente proibidos de dividir qualquer número por 0. Para não perder a sequência: $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q}$.

Números Reais: Até agora, parecia tudo bem, já que manipulando convenientemente os inteiros podia achar-se a resposta a qualquer problema (Assim achavam os pitagóricos). Daí que foi uma cacetada na inteligência matemática a descoberta de algumas questões que não poderiam ser resolvidas por números racionais. Um problema aqui é a *quadratura do círculo*. Trata-se de construir um quadrado de mesma área que um círculo de raio R conhecido, usando apenas compasso e régua. Este problema sempre foi considerado difícil, mas sua impossibilidade só foi demonstrada em 1882, pelo matemático alemão Ferdinand Lindemann. A razão é a irracionalidade de π . Outro problema é a obtenção da diagonal de um quadrado de lado 1 ($\sqrt{2}$, que também é irracional). Ao juntar os racionais com os irracionais, obtem-se o conjunto real. ($\mathbb{R}$). Se você dispuser os números sobre uma reta numerada, este é o primeiro conjunto de todos os acima citados que é *denso*, no sentido que pode-se estabelecer uma relação biunívoca entre os pontos na reta e os números do conjunto dos reais. Aqui $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$.

Os Complexos: Uma equação simples como $x^3 - 15x - 4 = 0$, tem como resposta $x = 4$, $x = -2 + \sqrt{3} = -0.2679491924$ e $x = -2 - \sqrt{3} = -3.732050808$, três números reais. Só que ao usar a fórmula de Tartaglia para esta equação obtem-se $x = \sqrt[3]{2 + \sqrt{-121}} + \sqrt[3]{2 - \sqrt{-121}}$. Quando, nas equações de grau 2 a fórmula de Baskara levava a raiz quadrada de números negativos, era fácil dizer que aquilo indicava a não existência de soluções. Agora, entretanto, nota-se que há equações de grau 3 com soluções reais conhecidas, mas cuja determinação passava pela extração de raízes quadradas de números negativos. O pulo do gato foi definir um número especial chamado i e supô-lo igual a $\sqrt{-1}$ batizando-o de unidade imaginária, daí a razão do nome i e tendo feito isto, criar um novo conjunto, o Complexo, ($\mathbb{C}$) cuja definição formal é $\mathbb{C} = a + bi$, $a, b \in \mathbb{R}$ e $i = \sqrt{-1}$. O nome imaginário tem uma conotação inadequada, ao sugerir que estes números não existem na realidade. Nada mais falso, hoje a moderna eletrônica digital (só para citar um único exemplo) é 100% calculada usando números complexos. Para obter uma representação geométrica destes números, basta acres-

centar à reta numerada real, outra reta perpendicular àquela no ponto $x = 0$ e caracterizar este novo eixo como imaginário. Agora o plano é também denso e qualquer ponto nele é caracterizado pelas coordenadas a e b em $p = a + bi$. Este é o plano de Argand-Gauss. Finalmente $\mathbb{R} \subset \mathbb{C}$.

Note que para estes conjuntos $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$ ou usando diagramas de Venn



Operações

A seguir vamos estudar as operações fundamentais no campo dos complexos. Deve-se lembrar que em tudo o que estudarmos, as regras do campo dos reais permanecerão válidas (um real $\mathbb{R}$ é um complexo, cuja parte imaginária vale 0). Portanto as operações vão ser generalizadas para se adaptar aos complexos sem perder nada do que aprendemos lá nos primeiros anos do fundamental. A saída aqui é considerar cada complexo como sendo um polinômio em i e operar a partir daqui, lembrando que $i^2 = -1$ de acordo com a definição. Antes de prosseguir a importante ressalva que no campo complexo NÃO é definido nenhum critério de ordem. Por exemplo, dados dois números complexos p_1 e p_2 pode-se dizer que $p_1 = p_2$ ou que $p_1 \neq p_2$, mas não se pode dizer $p_1 > p_2$ ou $p_1 < p_2$ não importa quem sejam os pontos. No campo real a reta numerada nos ajuda, mas no campo complexo sai a reta e entra o plano, daí a dificuldade.

Adição e subtração

Dados $p_1 = a + bi$ e $p_2 = c + di$, a adição $p_1 + p_2 = (a + c) + (b + d)i$. Por exemplo $p_1 = 2 + 4i$ e $p_2 = -2i$ leva a $p_1 + p_2 = 2 + 2i$. Nada se modifica na subtração, só lembrando que a adição é comutativa e a subtração não é. Insistindo no ponto acima, veja que esta definição preserva a adição de reais, já que $3 + 4$ pode ser entendida como $(3 + 0i) + (4 + 0i) = (3 + 4) + (0 + 0)i = 3 + 4 = 7$.

Multiplicação

Dados $p_1 = a + bi$ e $p_2 = c + di$, o produto $p_1 \times p_2$ leva a $(ac - bd) + (cb + ad)i$. Para não precisar decorar esta fórmula deduz-a do produto dos polinômios. Garanto que é menos trabalhoso.

Divisão

Dados $p_1 = a + bi$ e $p_2 = c + di$, a divisão $p_1 \div p_2$ leva a $\frac{p_1}{p_2} = \frac{a + bi}{c + di}$ que pode ser resolvida mul-

tiplicando a expressão por $\frac{c - di}{c - di} = 1$ (para tirar o i do denominador) e fica $\frac{a + bi}{c + di} \times \frac{c - di}{c - di} = \frac{ac + bd}{c^2 + d^2} + \frac{cb - ad}{c^2 + d^2}i$. Note que o fator usado para simplificar $(c - di)$ e chamado *conjugado* de $c + di$ e ele sinaliza um ponto simétrico em relação ao eixo x no plano de Argand-Gauss.

Módulo

Aqui uma importante generalização. Nos reais, $|\pm x| = +x$, enquanto que nos complexos $|a + bi| = \sqrt{a^2 + b^2}$. Pensando no plano de Argand- Gauss, fica claro que esta medida é a distância do ponto p à origem $(0, 0)$ no plano.

Forma Polar

Olhando para um ponto $z = a + bi$ no plano complexo, pode-se estabelecer as medidas: $\rho = |z|$ (por Pitágoras) e $\theta = \arctg \frac{b}{a}$ ou ainda $\text{sen} \theta = \frac{b}{\rho}$ e $\text{cos} \theta = \frac{a}{\rho}$ o que nos permite dizer que $a = \rho \text{cos}(\theta)$ e $b = \rho \text{sen}(\theta)$ e portanto $a + bi = \rho(\text{cos} \theta + i \text{sen} \theta)$. Aqui surge pequena dificuldade, já que na passagem da forma algébrica $(a + bi)$ para a forma polar $\rho(\text{cos} \theta + i \text{sen} \theta)$ deve-se calcular o $\arctg \theta$. Lembrando que a tangente é a divisão do seno pelo

coseno, pode-se ter uma situação onde o seno é positivo enquanto o coseno é negativo. Dessa fração a tangente fica negativa. Em outra situação, o seno é negativo e o coseno positivo. Da divisão surge a mesma tangente negativa. Isso ocorre porque a periodicidade da tangente é π enquanto a periodicidade tanto do seno quanto do coseno é 2π . Há que se fazer esse ajuste na conversão da forma algébrica $(a + bi)$ para a forma trigonométrica ou polar.

Raízes

A extração de raízes de um número complexo se vale de sua representação polar. Ou seja dado $z = \rho(\text{cos} \theta + i \text{sen} \theta)$ a obtenção das n raízes é $\sqrt[n]{z} = z^{(\frac{1}{n})} = \sqrt[n]{\rho} \left(\text{cos} \frac{\theta + 2\pi k}{n} + i \text{sen} \frac{\theta + 2\pi k}{n} \right)$, com $k = 0, 1, 2, \dots, n - 1$.

Potências

A consequência imediata da extração de raízes é a operação inversa que atende pelo nome de potência. Assim, se $\sqrt[n]{z} = w$ então $w^n = z$.

Possíveis soluções em Python

```
import numpy as np
def compltrig(x):
    ro = ((x[0]**2)+x[1]**2)**0.5
    if x[0]!=0:
        teta=np.arctan(x[1]/x[0])
        if x[0]<0:
            teta=teta+np.pi
        elif x[1]<0:
            teta=teta+np.pi
    else:
        teta=np.pi/2
    return [ro,teta]
#print(compltrig([1,-1]))
def complalg(x):
    return [x[0]*np.cos(x[1]),x[0]*np.sin(x[1])]
#print(complalg(compltrig([-1,1])))
def complmult(a,b):
    return[(a[0]*b[0])-(a[1]*b[1]), \
            ((a[1]*b[0])+(a[0]*b[1]))]
#print(complmult([2,3],[-1,4.5]))
def complpot(n,x):
    p=complmult (x,x)
    for i in range(n-2):
        p=complmult (x,p)
    return p
#print(complpot(5,[-1,.8]))
def complraiz(n,x):
    ro=((x[0]**2)+(x[1]**2))**.5
    if x[0] != 0:
        teta=np.arctan(x[1]/x[0])
        if x[0]<0:
            teta=teta+np.pi
        elif x[1]<0:
            teta=teta+np.pi
    else:
        teta=np.pi/2
    p1=ro**(1.0/n)
    for i in range(n):
        p2=np.cos((teta+(np.pi*2)*(i-1))/n)
        p3=np.sin((teta+(np.pi*2)*(i-1))/n)
        print('raiz=',i,' ',p1*p2,' ',p1*p3)
    #complraiz(10,[3.5,-1.2])
    #print(1+0j**0.1)
```

Para você fazer

Suponha que você está em um ambiente de programação que não disponha da facilidade complexa nativa. Se você tiver que calcular (por exemplo) uma transformada de Fourier de uma imagem, precisará aritmética complexa. Mesmo que a facilidade exista, programá-la é um excelente exercício sendo que neste caso (existindo a matemática complexa) você pode verificar imediatamente se programou certo ou não.

Extraia as 4 raízes do número complexo $7+2j$ elevado a $1/4$ e informe-as aqui. Arredonde as respostas para 3 decimais.

1	2	3	4



Aritmética Inteira Básica

Esta folha começa uma viagem pela matemática dos números inteiros que tem enorme importância na ciência da computação.

Números primos e compostos Lembremos a sequência dos números naturais: $\mathbb{N} = \{1, 2, 3, 4, 5, 6, 7, \dots\}$. O primeiro número (o 1) é especial e é chamado unidade e ela divide TODOS os naturais sem deixar resto. Depois dele, todos os números pertencem a uma de duas classes: os primos (só são divisíveis pela unidade e por eles mesmos) e os compostos: admitem pelo menos um divisor a mais, além da unidade e deles mesmos. Por exemplo, 8 é composto (8 é divisível por 1, 2, 4 e 8) e 11 é primo, pois 11 só é divisível por 1 e 11.

Os números primos são infinitos: Uma demonstração deste fato aparece aqui para nos mostrar o importante mecanismo *Redução ao absurdo* ou em latim *reductio ad absurdum*: Suponhamos que os primos são finitos e são $\{2, 3, 5, \dots, p\}$. Aqui p é o maior primo. Considere agora o número $q = (2 \times 3 \times 5 \times \dots \times p) + 1$. Note que q não é divisível por nenhum primo, pois deixa resto 1. Agora q é primo ou composto. Se for primo, ele não está na série de primos e contradizemos nossa afirmação inicial. Se for composto, ele tem fatores primos que não estão na série $\{2, 3, 5, \dots, p\}$. Isto também contradiz a afirmação inicial. Logo, os primos são infinitos.

Embora à medida em que o limite (dos primos) cresce eles se tornam mais raros. Por exemplo, entre 1 e 10, há 4 primos (40%), entre 1 e 100 há 25 (25%), entre 1 e 1000 há 168 (16%), entre 1 e 10000 há 1229 (12%), e assim por diante. O estudo da distribuição dos primos é tarefa antiga na matemática e as principais contribuições vem de Gauss e o trabalho de Riemann (sobre o qual paira um prêmio de 1 milhão de dólares para quem provar a *Conjectura de Riemann*).

Crivo de Eratóstenes Um algoritmo simples para encontrar números primos até um certo valor limite. Segundo a tradição, foi criado pelo matemático grego Eratóstenes (cerca de 275-194 a.C.), o terceiro bibliotecário-chefe da Biblioteca de Alexandria desde 247 (da wikipédia). Dada a sequência de naturais, sem o 1 (ele é diferente!), comece com o 2 e risque todos os seus múltiplos começando no 4 e seguindo até o limite desejado. Faça o mesmo com o 3 e risque seus múltiplos começando no 6 (9,12,15...). O 4 pode ser pulado (já está riscado) e repita o processo com o 5, riscando a partir do 10 (15,20,25,...). O 6 já está riscado então vai-se ao 7, riscando o 14, 21, 28, 35, ... Quando você chegar o limite proposto, todos os números que permanecem sem serem riscados são os primos.

Teorema fundamental da aritmética Qualquer número natural composto pode ser expresso de uma e só uma maneira como um produto de primos (desconsiderando-se a ordem do produto: propriedade comutativa). Este teorema também recebe o nome de Teorema da Fatoração Única. O método para achar esta fatoração está em tentar dividir o número composto pela série de números primos, eventualmente repetindo cada número primo até a exaustão. Por exemplo, achar a fatoração do número composto 360. Tem-se

$$360 = 2 \times 180 = 2 \times 2 \times 90 = 2 \times 2 \times 2 \times 45 = 2^3 \times 45 = 2^3 \times 3 \times 3 \times 5 = 2^3 \times 3^2 \times 5 = 360$$

mmc e mdc Uma importante aplicação de números primos é calcular o máximo divisor comum (mdc) e o mínimo múltiplo comum (mmc) de um conjunto de inteiros. O mdc de um conjunto de inteiros é o maior número que divide integralmente (sem deixar resto) todos os números do conjunto. Por exemplo o mdc de 10, 15 e 55 é 5. O mdc de

10,12 e 16 é 2 e o de 10,12,16 e 17 é 1. Um método possível para achar o mdc de um conjunto de números:

- Escreva os números em uma linha
- Divida-os pelos seus fatores comuns, seguindo a tabela de primos (2, 3, 5, ...)
- Pare quando não houver mais primos que dividam TODOS os números do conjunto.
- O produto dos primos divisores é o mdc

Por exemplo, achar o mdc de 24,60,96

24	60	96
2	12	30
2	6	15
3	2	5
8		

mdc = 2 × 2 × 3 = 12		

Outro exemplo, o mdc de 72 e 150 é $2 \times 3 = 6$.

Algoritmo de Euclides Como estamos na seara da computação, não se pode deixar de mencionar o algoritmo de Euclides para achar o mdc. Ele é importante pois

- tem 2400 anos de idade (apareceu nos livros 7 e 10 d’*O’s elementos*).
- é recursivo
- não precisa usar a fatoração
- $mdc(a, b) = mdc(b, a \% b)$ até o resto $a \% b$ ser igual a 0.

Acompanhe a execução do algoritmo para achar o mdc(360,1000):

1000	360
360	280
280	80
80	40
40	0
40 = mdc(1000,360)	

Já o mínimo múltiplo comum (mmc) de um conjunto de inteiros é o menor número inteiro que divide integralmente todos os elementos do conjunto sem deixar resto. O método para calcular o mmc:

- Fatorize todos os elementos do conjunto
- Escolha os primos que aparecem nas fatorizações
- Escolha o maior expoente de cada primo
- O produto destes é o mmc

Por exemplo, achar o mmc de 72 e 150. Tem-se

$$72 = 2^3 \times 3^2 \quad e \quad 150 = 2 \times 3 \times 5^2$$

o mmc é $2^3 \times 3^2 \times 5^2 = 1800$.

Note que o mdc de um conjunto de números é MENOR que todos os números do conjunto enquanto o mmc é MAIOR que todos os números do conjunto. Ambos (o mdc e o mmc) estão ligados pela expressão

$$\frac{\prod(x)}{mdc(x)} = mmc(x)$$

Por exemplo, dado o conjunto {75, 123} seu mmc é 3075 e seu mdc é 3 e $75 \times 123 = 9225$ e $3 \times 3075 = 9225$.

Cr terios de divisibilidade   uma tabela que ajuda a estabelecer quando um n mero   divis vel por outro, sobretudo os n meros iniciais do conjunto N.

n	teste	exemplo
2	Seu �ltimo d�gito � par	16774
3	a soma de seus d�gitos � divis�vel por 3	123
4	seus �ltimos 2 d�gitos s�o divis�veis por 4	7728
5	o n�mero termina por 0 ou 5	7770
6	o n�mero � divis�vel por 2 e por 3	27330
8	os �ltimos 3 d�gitos s�o divis�veis por 8	1.437.816
9	a soma dos d�gitos � divis�vel por 9	428.376.105
10	o �ltimo d�gito � 0	123.456.780
12	o n�mero � divis�vel por 3 e por 4	376.984.032

Soma de fra es A soma de fra es ocorre em 2 casos:

Denominadores iguais Neste caso, a soma das fra es   obtida simplesmente somando os numeradores e repetindo o denominador. Eventualmente depois disso deve-se (se poss vel) simplificar a fra o resultado. Por exemplo:

$$\frac{3}{8} + \frac{13}{8} = \frac{13 + 3}{8} = \frac{16}{8} = 2$$

Denominadores diferentes Antes de proceder   soma, deve-se achar fra es equivalentes (= de mesmo valor) que tenham denominadores iguais para recair no caso acima. Faz-se isso achando o m nimo m ltiplo comum (mmc) entre os dois denominadores e da : Cada fra o   reescrita como $(mmc \div denominador) \times numerador$.

Agora a soma   entre duas fra es que n o s o iguais  s originais, mas s o equivalentes a elas. Recai-se no primeiro caso acima. Por exemplo:

$$\frac{11}{7} + \frac{14}{6} = ?$$

o mmc entre 7 e 6   42. Da 

$$\frac{11}{7} = \frac{(42 \div 7 = 6) \times 11 = 66}{42} + \frac{(42 \div 6 = 7) \times 14 = 98}{42} = \frac{66 + 98 = 164}{42} = \frac{164}{42} = \frac{82}{21}$$

Produto e divis o de fra es A multiplica  o   simples: multiplica-se os numeradores e os denominadores. Ent o $\frac{a}{b} \times \frac{c}{d} = \frac{a \times c}{b \times d}$. J  a divis o de fra es implica na multiplica  o da primeira fra o pelo inverso da segunda. O inverso de uma fra o   obtido invertendo-se o numerador e o denominador. Ent o $\frac{a}{b} \div \frac{c}{d} = \frac{a}{b} \times \frac{d}{c} = \frac{a \times d}{b \times c}$.

Para voc  fazer

Some as express es envolvendo fra es e ache a resposta (possivelmente simplificada) correta.

1. $\frac{98}{69} + \frac{335}{194} + \frac{11417}{13386}$
2. $\frac{164}{81} + \frac{262}{195} + \frac{3326}{5265}$
3. $\frac{65}{36} + \frac{75}{38} + \frac{151}{684}$
4. $\frac{194}{41} + \frac{23}{8} + \frac{129}{328}$

1	2	3	4
			

Sistema de numeração

Um numeral é um símbolo ou grupo de símbolos que representa um número em um determinado instante da evolução do homem. Tem-se que, numa determinada escrita ou época, os numerais diferenciaram-se dos números do mesmo modo que as palavras se diferenciaram das coisas a que se referem. Os símbolos “11”, “onze” e “XI” (onze em latim) são numerais diferentes, representativos do mesmo número, apenas escrito em idiomas e épocas diferentes. A seguir, os vários aspectos dos sistemas de numerais.

Um sistema de numeração, (ou sistema numeral) é um sistema em que um conjunto de números são representados por numerais de uma forma consistente. Pode ser visto como o contexto que permite ao numeral “11” ser interpretado como o numeral romano para dois, o numeral binário para três ou o numeral decimal para onze.

Em condições ideais, um sistema de numeração deve:

Representar uma grande quantidade de números úteis (ex.: todos os números inteiros, ou todos os números reais); Dar a cada número representado uma única descrição (ou pelo menos uma representação padrão); Refletir as estruturas algébricas e aritméticas dos números. Por exemplo, a representação comum decimal dos números inteiros fornece a cada número inteiro uma representação única como uma sequência finita de algarismos, com as operações aritméticas (adição, subtração, multiplicação e divisão) estando presentes como os algoritmos padrões da aritmética. Contudo, quando a representação decimal é usada para os números racionais ou para os números reais, a representação deixa de ser padronizada: muitos números racionais têm dois tipos de numerais, um padrão que tem fim (por exemplo 2,31), e outro que repete-se periodicamente (como 2,30999999...).

Notação posicional

O valor atribuído a um símbolo é dependente da posição em que ele se encontra no conjunto de símbolos que representa uma quantidade. O valor total do número é a soma dos valores relativos de cada algarismo (decimal). Ou seja, a ordem dos algarismos (números) que estamos acostumados a usar: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

Exemplo

No sistema de numeração decimal,

735=700+30+5

Neste exemplo temos que o algarismo 5 representa 5 unidades. O algarismo 3 representa 3 dezenas (ou 3 grupos de dez unidades). E por último o algarismo 7 representa 7 centenas (ou 7 grupos de cem unidades).

573=500+70+3

Neste outro exemplo temos que o algarismo 3 representa 3 unidades. O algarismo 7 representa 7 dezenas (ou 7 grupos de dez unidades). E, finalmente, o algarismo 5 representa 5 centenas (ou 5 grupos de cem unidades).

Para seu conhecimento, o QR CODE, usa no registro de caracteres, um base de numeração de base 45. O protetor de conteúdos conhecido como Base64, usa um sistema de numeração de base 64, como aliás é o seu nome. O ambiente Bitcoin, usa uma numeração de base 58. O músico George Harrison criou a língua Sona, cuja numeração é de base 36. A cultura Maia usou numeração de base 20 e os antigos mesopotâmicos usavam base 60, origem das horas e minutos bem como dos 360° da circunferência, usados até hoje.

Aritmética não decimal

A aritmética, tal como a conhecemos, não sofre nenhuma mudança conceitual ou operacional se passarmos a usar bases não decimais. O sistema de numeração usado, desde que posicional, funciona bem.

Para as operações a seguir, considere um conjunto de dígitos sujeito as seguintes regras:

- 1. O primeiro dígito é sempre ZERO.
- 2. Existem tantos dígitos quanto é o valor da base.
- 3. Se forem necessários mais de 10 dígitos, o 11º será a letra A, o 12º a letra B e assim por diante até o 36º dígito que será o Z.

Na soma individual, dígito a dígito, a regra é simples: se a soma for maior que a base, retira-se uma base do total e vai um...

Na subtração individual, dígito a dígito, se o número a subtrair é maior do que o número do qual vai ser feita a subtração, empresta-se uma base do vizinho esquerdo.

Exemplos:

27(10) + 45(10) = 2 7
 + 4 5

 7 2

Como 7 + 5 = 12 e 12 é maior do que a base, subtrai-se uma base (10) e o resultado é 2, e "vai um". A resposta final é 72.

94(10) - 59(10) = 9 4
 - 5 9

 3 5

Como 4 - 9 não pode ser feito (pois 9 > 4), a solução é emprestar uma base do vizinho. Com isto, tem-se 4 + 10 = 14, e agora pode-se fazer 14 - 9, cujo resultado é 5.

O um que foi emprestado do 9, deixa-o valendo apenas 8, e agora tem-se 8 - 5 = 3. O resultado final é 35.

Para facilitar, use a seguinte tabela:

0..9=0..9	E=14	J=19	O=24	T=29	Y=34
A=10	F=15	K=20	P=25	U=30	Z=35
B=11	G=16	L=21	Q=26	V=31	
C=12	H=17	M=22	R=27	W=32	
D=13	I=18	N=23	S=28	X=33	

Nos exercícios a seguir, tente fazer as conversões e as operações solicitadas:

- 1. Conversoes de Base Q para base 10
- 122(18) = _____(10)
- 07(32) = _____(10)
- 510(12) = _____(10)
- 104(24) = _____(10)
- 2120(07) = _____(10)
- 2. Conversoes de base 10 para base Q
- 868(10) = _____(22)
- 772(10) = _____(18)
- 642(10) = _____(21)
- 703(10) = _____(09)
- 232(10) = _____(25)
- 3. Somas em base diferente de 10
- 1402(06) + 1113(06) = _____(06)
- 1343(05) + 1104(05) = _____(05)
- 1A1(15) + 1B9(15) = _____(15)
- 13I(22) + 103(22) = _____(22)
- I2(19) + 138(19) = _____(19)
- 4. Subtrações em base diferente de 10
- 176(18) - 128(18) = _____(18)
- 297(12) - 181(12) = _____(12)
- LF(26) - GP(26) = _____(26)
- DD(27) - 80(27) = _____(27)

- 671(09) - 271(09) = _____(09)
- 5. Somas em bases diferentes:
- CIT(31) + 1H88(18) = _____(09)
- 105045(07) + 1I4E(19) = _____(33)
- 6. Subtrações em bases...
- KGQ(30) - 155053(06) = _____(32)
- M7B(24) - D4Q(31) = _____(25)
- 7. Conversões de Base Q para base 10
- 293 (19) = _____(10)
- 1F7 (21) = _____(10)
- 8. Conversões de base 10 para base Q
- 00640 (10) = _____(21)
- 01172 (10) = _____(17)
- 9. Somas em base diferente de 10
- F8 (29) + AJ (29) = _____(29)
- 429 (11) + 180 (11) = _____(11)
- 10. Subtrações em base diferente de 10
- 150 (20) - D4 (20) = _____(20)
- CC (21) - BF (21) = _____(21)
- 11. Somas em bases diferentes: responda na base do resultado
- QGH (27) + L04 (30) = _____(11)
- J2H (26) + 30216 (08) = _____(09)
- 12. Subtrações em bases diferentes: responda na base do resultado
- 2F1I (19) - IA1 (24) = _____(28)
- 27BF (20) - 1CJC (21) = _____(31)

Condicional aninhadas

O comando condicional aparentemente é bastante simples: Dada uma condição se ela for verdadeira, executa-se um trecho de código. Se ela for falsa, ou não se executada nada ou se executa um trecho alternativo. Isso fica mais visível ao estudar o formato

```
se <condição> então
    trecho1
[senao
    trecho2]
fimse
```

As coisas começam a ficar um pouco mais complicadas quando:

- a condição é composta, usando $\wedge$, $\vee$, ou $\sim$ ou vários ou muitos deles...
- trecho1* ou *trecho2* são compostos por outros comandos condicionais...
- tudo isso é misturado, sem limites superiores...

Veja-se por exemplo, este trecho de código

```
1: funcao AAA
2: A ← 8
3: B ← 3
4: C ← 7
5: se A > 6 então
6:     se B = 5 então
7:         se C ≤ 8 então
8:             C ← C + 4
9:             B ← B - 5
10:        fim{se}
11:    A ← A × 2
12:    fim{se}
13:    B ← A × 2
14:    A ← C - 3
15: senão
16:     B ← C × 3
17:     C ← B × 3
18:     B ← B - 5
19: fim{se}
20: escreva A+B+C
21: fimfuncao
```

Neste trecho devidamente editado com identações corretas, pode-se verificar que o resultado ao final deve ser 27.

Fica bem mais chato se você olhar o mesmo código, sem indentação:

```
funcao AAA
A = 8
B = 3
C = 7
se A > 6 então
se B == 5 então
se C <= 8 então
C = C + 4
B = B - 5
fim{se}
A = A * 2
fim{se}
B = A * 2
A = C - 3
senao
B = C * 3
C = B * 3
B = B - 5
fim{se}
imprima (A+B+C)
fim{funcao}
```

Em qualquer linguagem de programação (C, Java, Pascal, Basic, Cobol, Lisp, APL, Clipper, cpc, ... exceto

Python), ambos os trechos acima serão igualmente compilados. O objetivo deste exercício é treinar a habilidade humana de passar do trecho sem indentação para aquele com indentação e obviamente depois interpretar (corretamente) o que o programa faz. Veja a seguir, mais dois exemplos: O trecho

```
funcao AAA
A = 8
B = 3
C = 7
se A > 6 então
se B == 5 então
se C <= 8 então
C = C + 4
A = C - 3
C = B * 3
senao
B = B - 5
B = A * 2
fim{se}
B = B - 5
B = C * 3
fim{se}
fim{se}
A = A * 2
imprima (A+B+C)
fim{funcao}
```

que vale 26 e o trecho

```
funcao AAA
A = 8
B = 3
C = 7
se A > 6 então
se B == 5 então
se C <= 8 então
C = C + 4
B = C * 3
B = A * 2
senao
B = B - 5
fim{se}
fim{se}
senao
C = B * 3
C = B * 3
fim{se}
A = C - 3
A = A * 2
imprima (A+B+C)
fim{funcao}
```

que vale 18.

👉 Para você fazer

1. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 2
B = 2
C = 5
se A == 9 então
se B == 8 então
se C <= 6 então
B = A + 5
C = A - 4
fim{se}
B = C - 2
senao
A = B + 5
fim{se}
C = B * 3
C = A * 3
C = A + 3
fim{se}
imprima (A+B+C)
fim{funcao}
```

Para seu auxílio, veja o mesmo trecho escrito com indentação correta:

```
1: funcao AAA
2: A ← 2
3: B ← 2
4: C ← 5
5: se A = 9 então
6:     se B = 8 então
```

```
7:         se C ≤ 6 então
8:             B ← A + 5
9:             C ← A - 4
10:        fim{se}
11:    B ← C - 2
12:    senão
13:        A ← B + 5
14:        fim{se}
15:    C ← B × 3
16:    C ← A × 3
17:    C ← A + 3
18: fim{se}
19: escreva A+B+C
20: fimfuncao
```

O resultado foi _____

2. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 2
B = 2
C = 5
se A == 9 então
se B == 8 então
se C <= 6 então
B = A + 5
fim{se}
senao
C = A * 3
A = B + 5
fim{se}
fim{se}
senao
C = B * 3
C = A + 3
C = A - 4
fim{se}
B = C - 2
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

3. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 2
B = 2
C = 5
se A == 9 então
se B == 8 então
se C <= 6 então
B = A + 5
B = C - 2
senao
A = B + 5
fim{se}
fim{se}
senao
C = A + 3
fim{se}
C = A * 3
C = A - 4
C = B * 3
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

4. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 9
B = 4
C = 9
se A == 6 então
se B >= 9 então
se C <= 8 então
B = A * 2
fim{se}
senao
A = B - 2
fim{se}
A = C * 2
senao
B = C + 2
B = C * 4
C = C * 3
A = A - 3
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

5. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 9
B = 4
C = 9
se A == 6 então
se B >= 9 então
se C <= 8 então
B = A * 2
senao
A = B - 2
A = C * 2
fim{se}
senao
B = C + 2
B = C * 4
fim{se}
senao
A = A - 3
C = C * 3
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

6. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 9
B = 4
C = 9
se A == 6 então
se B >= 9 então
se C <= 8 então
B = A * 2
A = C * 2
B = C * 4
A = B - 2
C = C * 3
A = A - 3
fim{se}
B = C + 2
fim{se}
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

1	2	3
4	5	6



Condicionalis aninhadas e compostas

O comando condicional aparentemente é bastante simples: Dada uma condição se ela for verdadeira, executa-se um trecho de código. Se ela for falsa, ou não se executada nada ou se executa um trecho alternativo. Isso fica mais visível ao estudar o formato

```
se <condição> então
    trecho1
[senao
    trecho2]
fimse
```

As coisas começam a ficar um pouco mais complicadas quando:

- a condição é composta, usando $\wedge$, $\vee$, ou $\sim$ ou vários ou muitos deles...
- trecho1* ou *trecho2* são compostos por outros comandos condicionais...
- tudo isso é misturado, sem limites superiores...

Veja-se por exemplo, este trecho de código (A palavra *entao* foi omitida para não atrapalhar...)

```
1: funcao AAA
2: A ← 3
3: B ← 7
4: C ← 9
5: se ~A>4∨~B=4∧~C≤3
6:   se ~A≠2∨~B≤5
7:     se ~A≥2∧~B=6
8:       C ← A - 2
9:     fim{se}
10:   senão
11:     B ← C × 4
12:   fim{se}
13:   B ← C × 4
14:   C ← C - 5
15:   A ← C + 3
16:   C ← A - 4
17:   C ← B × 3
18: fim{se}
19: A ← B - 3
20: escreva A+B+C
21: fimfuncao
```

Neste trecho devidamente editado com identações corretas, pode-se verificar que o resultado ao final deve ser 177.

Fica bem mais chato se você olhar o mesmo código, sem identificação

```
funcao AAA
A = 3
B = 7
C = 9
se ((NAO(A>4))OU(NAO(B==4)))
    E(NAO(C<=3)) entao
se (NAO(A<>2))OU(NAO(B<=5)) entao
se (NAO(A>=2))E(NAO(B==6)) entao
C = A - 2
fim{se}
senao
B = C * 4
fim{se}
B = C * 4
C = C - 5
A = C + 3
C = A - 4
C = B * 3
fim{se}
A = B - 3
imprima (A+B+C)
fim{funcao}
```

Em qualquer linguagem de programação (C, Java, Pascal, Basic, Cobol, Lisp, APL, Clipper, cpc, ... exceto Python), ambos os trechos acima serão igualmente compilados. O objetivo deste exercício é treinar a habilidade humana de passar do trecho sem identificação para aquele com identificação e obviamente depois interpretar (corretamente) o que o programa faz. Veja a seguir, mais dois exemplos:

```
funcao AAA
A = 3
B = 7
C = 9
se ((NAO(A>4))OU(NAO(B==4)))
    E(NAO(C<=3)) entao
se (NAO(A<>2))OU(NAO(B<=5)) entao
se (NAO(A>=2))E(NAO(B==6)) entao
C = A - 2
A = B - 3
C = B * 3
B = C * 4
fim{se}
C = C - 5
A = C + 3
C = A - 4
fim{se}
fim{se}
imprima (A+B+C)
fim{funcao}
```

que vale 17 e o trecho

```
funcao AAA
A = 3
B = 7
C = 9
se ((NAO(A>4))OU(NAO(B==4)))
    E(NAO(C<=3)) entao
se (NAO(A<>2))OU(NAO(B<=5)) entao
se (NAO(A>=2))E(NAO(B==6)) entao
C = A - 2
fim{se}
senao
B = C * 4
fim{se}
A = C + 3
senao
C = C - 5
C = C - 5
C = A - 4
A = B - 3
fim{se}
C = B * 3
imprima (A+B+C)
fim{funcao}
```

que vale 40.

Para você fazer

1. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 7
B = 2
C = 4
se ((NAO(A<4))OU(B<>4))OU(C=6) entao
se (A==7)OU(NAO(B<>7)) entao
se ((NAO(A>4))OU(NAO(B>6)))OU(NAO(C==7))entao
C = C * 5
A = A + 2
fim{se}
senao
A = C - 4
C = B + 5
fim{se}
senao
B = B + 3
A = B - 4
A = C - 4
fim{se}
imprima (A+B+C)
fim{funcao}
```

Para seu auxílio, veja o mesmo trecho escrito com identificação correta:

```
1: funcao AAA
2: A ← 7
3: B ← 2
```

```
4: C ← 4
5: se ~A<4∨B≠4∨C≥6 entao
6:   se A=7∨~B≠7 entao
7:     se ~A>4∨~B>6∨~C=7entao
8:       C ← C × 5
9:       A ← A + 2
10:     fim{se}
11:   senão
12:     A ← C - 4
13:     C ← B + 5
14:   fim{se}
15: senão
16:   B ← B + 3
17:   A ← B - 4
18:   A ← C - 4
19: fim{se}
20: escreva A+B+C
21: fimfuncao
    O resultado foi _____
```

2. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 7
B = 2
C = 4
se ((NAO(A<4))OU(B<>4))OU(C=6) entao
se (A==7)OU(NAO(B<>7)) entao
se ((NAO(A>4))OU(NAO(B>6)))OU(NAO(C==7))entao
C = C * 5
senao
B = B + 3
A = C - 4
A = C - 4
fim{se}
senao
A = B - 4
A = A + 2
fim{se}
fim{se}
C = B + 5
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

3. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 7
B = 2
C = 4
se ((NAO(A<4))OU(B<>4))OU(C=6) entao
se (A==7)OU(NAO(B<>7)) entao
se ((NAO(A>4))OU(NAO(B>6)))OU(NAO(C==7))entao
C = C * 5
C = B + 5
senao
A = A + 2
A = C - 4
fim{se}
fim{se}
A = C - 4
senao
A = B - 4
B = B + 3
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

4. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 4
B = 6
C = 6
se (NAO(A==7))OU(B<=3) entao
se (NAO(A<>4))E(NAO(B>2)) entao
se (NAO(A>=7))OU(NAO(B<=7)) entao
C = B * 2
senao
A = A + 3
fim{se}
senao
A = C * 4
B = C - 2
A = B + 2
```

```
B = B - 3
fim{se}
fim{se}
B = B * 5
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

5. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 4
B = 6
C = 6
se (NAO(A==7))OU(B<=3) entao
se (NAO(A<>4))E(NAO(B>2)) entao
se (NAO(A>=7))OU(NAO(B<=7)) entao
C = B * 2
fim{se}
senao
A = A + 3
B = C - 2
fim{se}
B = B - 3
fim{se}
A = B + 2
A = C * 4
B = B * 5
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

6. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A = 4
B = 6
C = 6
se (NAO(A==7))OU(B<=3) entao
se (NAO(A<>4))E(NAO(B>2)) entao
se (NAO(A>=7))OU(NAO(B<=7)) entao
C = B * 2
senao
A = A + 3
A = C * 4
fim{se}
B = B - 3
B = C - 2
senao
A = B + 2
fim{se}
senao
B = B * 5
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

1	2	3
4	5	6



Condicionalis aninhadas e compostas II

O comando condicional aparentemente é bastante simples: Dada uma condição se ela for verdadeira, executa-se um trecho de código. Se ela for falsa, ou não se executada nada ou se executa um trecho alternativo. Isso fica mais visível ao estudar o formato

```
se <condição> então
    trecho1
[senao
    trecho2]
fimse
```

As coisas começam a ficar um pouco mais complicadas quando:

- a condição é composta, usando $\wedge$, $\vee$, ou $\sim$ ou vários ou muitos deles...
- trecho1* ou *trecho2* são compostos por outros comandos condicionais...
- tudo isso é misturado, sem limites superiores...

Veja-se por exemplo, este trecho de código

```
1: funcao AAA
2: A ← 7
3: B ← 5
4: C ← 4
5: se ~A=2∨~B≤7∧C>4
6:   se B ≠ 6 então
7:     se C < 8 então
8:       se C ≠ 7 então
9:         C ← C - 4
10:      senão
11:        A ← A + C × 5
12:       C ← A - 4
13:      fim{se}
14:    senão
15:      C ← C + B × 3
16:    fim{se}
17:  senão
18:    C ← C + A - 3
19:  fim{se}
20: senão
21:   C ← B + 2
22: fim{se}
23: escreva A+B+C
24: fimfuncao
```

Neste trecho devidamente editado com identações corretas, pode-se verificar que o resultado ao final deve ser 19.

Fica bem mais chato se você olhar o mesmo código, sem identação:

```
funcao AAA
A,B,C = 7,5,4
se ((NAO(A==2))OU(NAO(B<=7)))
    E(C>4) então
se B <> 6 então
se C < 8 então
se C <> 7 então
C = C - 4
senao
A = A + (C * 5)
C = A - 4
fim{se}
senao
C = C + (B * 3)
fim{se}
senao
C = C + (A - 3)
fim{se}
senao
C = B + 2
fim{se}
imprima (A+B+C)
fim{funcao}
```

Em qualquer linguagem de programação (C, Java, Pascal, Basic, Cobol, Lisp, APL, Clipper, cpc, ... exceto Python), ambos os trechos acima serão igualmente compilados. O objetivo deste exercício é treinar a habilidade humana de passar do trecho sem indentação para aquele com indentação e obviamente depois interpretar (corretamente) o que o programa faz. Veja a seguir, mais dois exemplos:

```
O trecho

funcao AAA
A,B,C = 7,5,4
se ((NAO(A==2))OU(NAO(B<=7)))
    E(C>4) então
se B <> 6 então
se C < 8 então
se C <> 7 então
C = C - 4
C = B + 2
fim{se}
senao
C = C + (A - 3)
A = A + (C * 5)
fim{se}
C = A - 4
senao
C = C + (B * 3)
fim{se}
fim{se}
imprima (A+B+C)
fim{funcao}
```

que vale 16 e o trecho

```
funcao AAA
A,B,C = 7,5,4
se ((NAO(A==2))OU(NAO(B<=7)))
    E(C>4) então
se B <> 6 então
se C < 8 então
se C <> 7 então
C = C - 4
C = A - 4
senao
A = A + (C * 5)
C = B + 2
C = C + (A - 3)
fim{se}
fim{se}
senao
C = C + (B * 3)
fim{se}
imprima (A+B+C)
fim{funcao}
```

que vale 31.

🔗 Para você fazer

1. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=5,2,9
se ((NAO(A<=3))OU(NAO(B<=3)))OU(C==3)
    então
se B <= 9 então
se C <= 5 então
se C >= 5 então
B = A - 3
senao
C = C + (A + 2)
fim{se}
senao
B = B + (C * 2)
fim{se}
senao
B = B + (A + 4)
fim{se}
A = A + 5
imprima (A+B+C)
fim{funcao}
```

Para seu auxílio, veja o mesmo trecho escrito com indentação correta:

```
1: funcao AAA
2: A ← 5
3: B ← 2
```

```
4: C ← 9
5: se ~A≤3∨~B≤3∨C=3entao
6:   se B ≤ 9 então
7:     se C ≤ 5 então
8:       se C ≥ 5 então
9:         B ← A - 3
10:      senão
11:        C ← C + A + 2
12:     fim{se}
13:   senão
14:     B ← B + C × 2
15:   fim{se}
16: senão
17:   B ← B + A + 4
18: fim{se}
19: senão
20:   B ← B × 3
21: fim{se}
22: A ← A + 5
23: escreva A+B+C
24: fimfuncao

O resultado foi _____
```

2. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=5,2,9
se ((NAO(A<=3))OU(NAO(B<=3)))OU(C==3)
    então
se B <= 9 então
se C <= 5 então
se C >= 5 então
B = A - 3
senao
C = C + (A + 2)
A = A + 5
fim{se}
fim{se}
senao
B = B + (C * 2)
fim{se}
senao
B = B * 3
B = B + (A + 4)
fim{se}
imprima (A+B+C)
fim{funcao}

O resultado foi _____
```

3. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=5,2,9
se ((NAO(A<=3))OU(NAO(B<=3)))OU(C==3)
    então
se B <= 9 então
se C <= 5 então
se C >= 5 então
B = A - 3
fim{se}
A = A + 5
B = B + (C * 2)
senao
B = B * 3
C = C + (A + 2)
fim{se}
fim{se}
B = B + (A + 4)
imprima (A+B+C)
fim{funcao}

O resultado foi _____
```

4. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=3,9,6
se (NAO(A>=7))OU(B<2)entao
se B >= 8 então
se C < 7 então
se B < 8 então
A = C * 2
C = B + 5
senao
A = A + (B + 2)
fim{se}
senao
C = C + (A * 3)
fim{se}
senao
```

```
B = B + (C * 4)
fim{se}
senao
A = A * 4
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

5. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=3,9,6
se (NAO(A>=7))OU(B<2)entao
se B >= 8 então
se C < 7 então
se B < 8 então
A = C * 2
fim{se}
senao
A = A + (B + 2)
fim{se}
fim{se}
senao
B = B + (C * 4)
C = B + 5
C = C + (A * 3)
A = A * 4
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado oi _____

6. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=3,9,6
se (NAO(A>=7))OU(B<2)entao
se B >= 8 então
se C < 7 então
se B < 8 então
A = C * 2
C = B + 5
senao
A = A + (B + 2)
B = B + (C * 4)
fim{se}
C = C + (A * 3)
fim{se}
A = A * 4
fim{se}
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

1	2	3
4	5	6



Condicionais aninhadas e compostas III

O comando condicional aparentemente é bastante simples: Dada uma condição se ela for verdadeira, executa-se um trecho de código. Se ela for falsa, ou não se executa nada ou se executa um trecho alternativo. Isso fica mais visível ao estudar o formato

```
se <condição> então
    trecho1
[senao
    trecho2]
fimse
```

As coisas começam a ficar um pouco mais complicadas quando:

- a condição é composta, usando $\wedge$, $\vee$, ou $\sim$ ou vários ou muitos deles...
- trecho1* ou *trecho2* são compostos por outros comandos condicionais...
- tudo isso é misturado, sem limites superiores...

Seja este exemplo

```
funcao AAA
A,B,C=5,7,9
se ((A>=2)OU(NAO(B>7)))E
    (NAO(C>2)) entao
se B >= 6 entao
se C < 5 entao
se A < 7 entao
se A == 7 entao
B = A * 3
senao
C = C + (A * 2)
fim{se}
senao
C = B - 4
B = B + (C * 2)
fim{se}
senao
A = A + (B - 5)
fim{se}
fim{se}
B = B - 5
fim{se}
C = A * 4
imprima (A+B+C)
fim{funcao}
```

Que vale 32. Em qualquer linguagem de programação (C, Java, Pascal, Basic, Cobol, Lisp, APL, Clipper, cpc, ... exceto Python), o trechos acima será compilado sem problema. Em Python, a linguagem mais moderna dessa lista, a indentação é necessária. O objetivo deste exercício é treinar a habilidade humana de passar do trecho sem indentação para aquele com indentação e obviamente depois interpretar (corretamente) o que o programa faz. Veja a seguir, mais um exemplo: O trecho

```
funcao AAA
A,B,C=5,7,9
se ((A>=2)OU(NAO(B>7)))E
    (NAO(C>2)) entao
se B >= 6 entao
se C < 5 entao
se A < 7 entao
se A == 7 entao
B = A * 3
fim{se}
B = B - 5
C = A * 4
```

```
fim{se}
fim{se}
senao
C = B - 4
C = C + (A * 2)
fim{se}
A = A + (B - 5)
senao
B = B + (C * 2)
fim{se}
imprima (A+B+C)
fim{funcao}
```

vale 39.

Dicas para a implementação em C++

Se o seu ambiente de trabalho é C++, considere:

- Todas as variáveis são inteiras.
- O comando de atribuição precisa ser corrigido. Ele está como está para economizar espaço.
- Ao final dos comandos é necessário incluir um ;
- as condições precisam ser colocadas entre parênteses.

🔧 Para você fazer

1. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=6,7,4
se (NAO(A>=4))OU(B<3)entao
se B < 6 entao
se C <> 7 entao
se A < 7 entao
se B > 8 entao
A = B * 3
senao
A = A + (C * 4)
fim{se}
fim{se}
senao
A = A + (B * 3)
fim{se}
senao
B = B + 5
A = B + 4
C = C + (B - 4)
fim{se}
senao
A = B - 3
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

2. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=6,7,4
se (NAO(A>=4))OU(B<3)entao
se B < 6 entao
se C <> 7 entao
se A < 7 entao
se B > 8 entao
A = B * 3
fim{se}
senao
A = A + (C * 4)
fim{se}
fim{se}
senao
A = B - 3
A = A + (B * 3)
A = B + 4
B = B + 5
fim{se}
senao
C = C + (B - 4)
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

3. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=7,5,4
se (NAO(A<=7))E(NAO(B==2))entao
se B <> 6 entao
se C < 8 entao
se C >= 8 entao
se B >= 6 entao
A = C + 2
fim{se}
senao
C = A - 4
B = B + (C + 4)
fim{se}
senao
A = B * 5
A = C * 2
B = B + (A * 5)
fim{se}
fim{se}
senao
C = C + (A * 3)
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

4. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=7,5,4
se (NAO(A<=7))E(NAO(B==2))entao
se B <> 6 entao
se C < 8 entao
se C >= 8 entao
se B >= 6 entao
A = C + 2
senao
A = C * 2
B = B + (C + 4)
fim{se}
fim{se}
senao
B = B + (A * 5)
fim{se}
C = A - 4
senao
C = C + (A * 3)
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

5. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=6,8,7
se ((NAO(A<5))OU(B<=6))E(C<2)entao
se B == 7 entao
se C > 5 entao
se C < 7 entao
se C >= 7 entao
B = C - 4
fim{se}
senao
A = A + (B - 4)
A = B - 2
C = C + 5
fim{se}
senao
C = C + (A * 2)
fim{se}
B = B - 5
senao
A = A + (B + 3)
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

6. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=6,8,7
se ((NAO(A<5))OU(B<=6))E(C<2)entao
se B == 7 entao
```

```
se C > 5 entao
se C < 7 entao
se C >= 7 entao
B = C - 4
fim{se}
senao
A = A + (B - 4)
B = B - 5
fim{se}
senao
C = C + (A * 2)
A = B - 2
fim{se}
senao
A = A + (B + 3)
fim{se}
fim{se}
C = C + 5
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

7. Calcule o valor correto que será impresso pelo trecho a seguir

```
funcao AAA
A,B,C=2,9,3
se (NAO(A<7))OU(B<=4)entao
se B < 5 entao
se C > 8 entao
se C <= 5 entao
se C == 6 entao
B = A - 3
senao
C = C + (B * 5)
fim{se}
senao
A = A + (C + 3)
fim{se}
fim{se}
senao
C = C + (B + 3)
fim{se}
A = B + 2
senao
C = A * 5
C = A * 5
fim{se}
imprima (A+B+C)
fim{funcao}
```

O resultado foi _____

1	2	3	4
5	6	7	xxx xxx



Comandos condicionais literais

Neste exercicio você vai receber alguns comandos condicionais simples e compostos e também aninhados e deve descobrir qual o resultado desses comandos. Acompanhe no exemplo:

```
1.-----
se tem neblina então
    vou ao futebol
fimse
não tem neblina. Que ação ocorre ?
2.-----
se tem nuvens então
    vou ao médico
senão
    vou à piscina
fimse
tem nuvens. Que ação ocorre ?
3.-----
se está trovejando ou faz calor então
    vou ao barigui
senão
    vou ao teatro
fimse
está trovejando e faz calor. ?
4.-----
se faz frio e é noite então
    vou ao banco
senão
    vou à piscina
fimse
não faz frio e é noite. ?
5.-----
visto camisa verde
se está trovejando então
    vou ao futebol
senão
    se é noite então
        vou ao circo
        visto camisa branca
    senão
        vou à aula
    fimse
fimse
Não está trovejando e é noite. ?
6.-----
visto camisa branca
se está chovendo então
    se faz calor então
        vou ao médico
    senão
        vou ao barigui
    fimse
    visto camisa verde
senão
    vou ao banco
fimse
Não está chovendo e faz calor. E ?
7.-----
visto camisa branca
se faz frio então
    se está chovendo então
        vou à aula
    senão
        vou ao teatro
    fimse
    visto camisa vermelha
senão
    se faz calor então
        vou ao médico
    senão
        vou ao circo
    fimse
fimse
Não faz frio e está chovendo
e não faz calor. E daí ?
```

As respostas são:
1. indeterminado (nada se pode afirmar)
2. médico
3. barigui
4. piscina
5. circo+branca
6. banco+branca
7. circo+branca

🔊 Para você fazer

```
1.-----
visto camisa verde
se tem neblina então
    se está trovejando então
        vou ao circo
    senão
        vou ao banco
    fimse
    visto camisa azul
senao
    vou à piscina
fimse
Não tem neblina e está trovejando. E ?
2.-----
se tem neblina então
    vou ao cinema
fimse
tem neblina. Que ação ocorre ?
3.-----
se faz calor e faz frio então
    vou ao teatro
senão
    vou ao médico
fimse
faz calor e faz frio. ?
4.-----
se tem nuvens então
    vou ao circo
senão
    vou ao teatro
fimse
não tem nuvens. Que ação ocorre ?
5.-----
se está chovendo ou tem neblina então
    vou ao barigui
senão
    vou ao veterinário
fimse
Nenhuma das condições é verdadeira. Que ocorre ?
6.-----
visto camisa verde
se tem neblina então
    se está trovejando então
        vou à piscina
    senão
        vou ao médico
    fimse
    visto camisa azul
senão
    se tem nuvens então
        vou ao cinema
    senão
        vou ao banco
    fimse
fimse
Não tem neblina e não está trovejando
e não tem nuvens. E daí ?
7.-----
visto camisa verde
se tem nuvens então
    vou ao barigui
senão
    se está chovendo então
        vou à piscina
        visto camisa vermelha
    senão
        vou ao futebol
    fimse
fimse
tem nuvens e está chovendo. ?
```

Responda aqui o valor de todas as variáveis significativas (as que foram mencionadas no enunciado):

1	2	3	4
5	6	7	
			xxx xxx xxx

🔊 Mais um

```
1.-----
se londrina é grande e quitéria é mala então
    jogar búrico
senão
    torcer pelo brasil
fimse
londrina é grande e não quitéria é mala. ?
2.-----
estou elegante
se irati é formosa então
    se sengés é longe então
        cantar canção
    senão
        jogar búrico
    fimse
    estou jéca
senao
    caçar sapo
fimse
Não irati é formosa e sengés é longe. E ?
3.-----
se curitiba é linda então
    torcer pelo brasil
senão
    viajar à lapa
fimse
não curitiba é linda. Que ação ocorre ?
4.-----
estou jéca
se londrina é grande então
    se sengés é longe então
        caçar sapo
    senão
        trepar em árvore
    fimse
    estou elegante
senão
    se maria é bonita então
        cantar canção
    senão
        torcer pelo brasil
    fimse
fimse
londrina é grande e não sengés é longe
e não maria é bonita. E daí ?
5.-----
se quita é simpática então
    comer salada
fimse
quita é simpática. Que ação ocorre ?
6.-----
estou indiferente
se maria é bonita então
    torcer pelo brasil
senão
    se curitiba é linda então
        jogar búrico
        estou jéca
    senão
        viajar à lapa
    fimse
fimse
maria é bonita e não curitiba é linda. ?
7.-----
se quitéria é mala ou sengés é longe então
    viajar à lapa
senão
    cantar canção
fimse
quitéria é mala e não sengés é longe. ?
```

Responda aqui o valor de todas as variáveis significativas (as que foram mencionadas no enunciado):

1	2	3	4
5	6	7	
			xxx xxx xxx



Integridade de Funções

O capítulo de funções sempre guarda inúmeras particularidades em qualquer linguagem de programação. Nesta folha, vai-se estudar alguns erros comuns na criação e manuseio de funções na linguagem C.

Antes de começar recomenda-se fortemente a leitura e estudo profundo do capítulo 6 ("Funções") do livro C Completo e Total de Herbert Schildt, ou qualquer material similar.

Definiu-se nesta folha um conjunto de 9 erros possíveis na definição e uso de funções dentro de programas C. A lista de erros não é completa, e mesmo alguns erros, não são apontados como tal por todos os compiladores, mas o que nos interessa aqui é o erro conceitual.

A tabela de erros é

0=caso particular em que NÃO há nenhum erro.
1=chama com número de parâmetros diferente do número original
2=chama com tipo de parâmetro diferente do tipo original
3=o usador não faz nada com o que a função retorna
4=define uma variável com o mesmo nome da função
5=usa fora da função uma variável definida dentro dela
6=define função com um nome e chama-a com nome parecido
7=a função não retorna nada, mas o chamador usa um possível retorno
8=a função usa variável definida fora dela
9=a função recebe um valor

A seguir, uma lista de programas C, cada um deles formado por uma função e logo depois a função main() que chama a função que foi acima definida.

Examinando cuidadosamente o bloco, descubra qual erro foi cometido. Caso não ocorra nenhum erro, responda como erro=0.

Exemplo

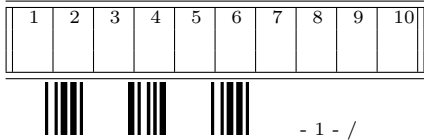
```
1) void FHJ12 (int A, float B, int K) {
    int Q, X;
    float Z, T;
    printf("%i %f\n",A,B);
}
main() {
    int F, J, K;
    float W, N, M;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    FHJ12(K,N,J);
}
2) int FBC45 (int A, float B, float C) {
    float X, Y;
    int W, Z;
    return (A + floor(B + C));
}
main() {
    int F, J, K;
    float L, M, N;
    float FBC45;
    scanf("%i %i %i %f %f %f",&F,&J,&K,&L,&M,&N);
    printf("%i\n",FBC45(F,L,M));
}
3) int FKI73 ( ) {
    int A, B;
    A := 0;
    return(12);
}
main() {
    int F, J, K;
    float W, N, M;
    printf("%i\n",FIK73());
}
4) void FHJ12 (int A, float B, int K) {
    int Q, X;
    float Z, T;
    printf("%i %f\n",A,B);
}
main() {
    int F, J, K;
    float W, N, M;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    FHJ12(K,N,J);
}
5) int FHJ12 (int A, float B, int K) {
    int Q, X;
    float Z, T;
    printf("%i %f\n",A,B);
    return(A);
}
main() {
    int F, J, K;
    float W, N, M;
```

```
scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    FHJ12(K,N,J)
}
6) int FAJ68 (int A, float B, ... ) {
    float X, Y;
    int Q, Z;
    return (A + floor(B + 2));
}
main() {
    int F, J, K;
    float W, N, M;
    float FAJ68 := 0;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    printf("%i\n",FAJ68(K,N,J,M));
}
7) void FHJ12 (int A, float B, int K) {
    int Q, X;
    float Z, T;
    T := K + 1;
    printf("%i %f\n",A,B);
}
main() {
    int F, J, K;
    float W, N, M;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    FHJ12(K,N,J);
}
8) int FKI73 ( ) {
    int A, B;
    A := 0;
    return(12);
}
main() {
    int F, J, K;
    float W, N, M;
    FKI73:=9;
    printf("%i\n",FKI73());
}
Com a seguinte resposta
0 4 6 0 3 9 8 9
```

🔗 Para você fazer

```
1) int FAA23 (int A, int B, int C) {
    int X, Y, Z;
    if (A >= B) {
        return C; }
    else {
        return (A + B); }
}
main() {
    int F, J, K;
    float L, M, N;
    scanf("%i %i %i %f %f %f",&F,&J,&K,&L,&M,&N);
    printf("%i\n",FAA23(F,J,K));
}
2) int FAA23 (int A, int B, int C) {
    int X, Y, Z;
    if (A >= B) {
        return (C+L); }
    else {
        return (A+M); }
}
main() {
    int F, J, K;
    float L, M, N;
    scanf("%i %i %i %f %f %f",&F,&J,&K,&L,&M,&N);
    printf("%i\n",FAA23(F,J,K));
}
3) int FAJ68 (int A, float B, ... ) {
    float X, Y;
    int Q, Z;
    return (A + floor(B + 2));
}
main() {
    int F, J, K;
    float W, N, M;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    printf("%i\n",FAJ68(K,N,J,M));
}
4) int FAA23 (int A, int B, int C) {
    int X, Y, Z;
    if (A >= B) {
        return C; }
    else {
        return (A + B); }
}
main() {
    int F, J, K;
    float L, M, N;
    scanf("%i %i %i %f %f %f",&F,&J,&K,&L,&M,&N);
    FAA23(F,J,K);
    printf("%f %f\n",L,M,N);
}
5) int FKI73 ( ) {
```

```
int A, B;
A := 0;
return(12);
}
main() {
    int F, J, K;
    float W, N, M;
    FKI73:=9;
    printf("%i\n",FKI73());
}
6) int FAA23 (int A, int B, int C) {
    int X, Y, Z;
    if (A >= B) {
        return C; }
    else {
        return (A + B); }
}
main() {
    int F, J, K;
    scanf("%i %i %i",&F,&J,&K);
    printf("%i\n",FAA23(F,J));
}
7) int FBC45 (int A, float B, float C) {
    float X, Y;
    int W, Z;
    return (A + floor(B + C));
}
main() {
    int F, J, K;
    float L, M, N;
    scanf("%i %i %i %f %f %f",&F,&J,&K,&L,&M,&N);
    printf("%i\n",FCB45(F,M,N));
}
8) float FVX66 (float A, int B, int D) {
    float T, Q;
    int F, G;
    if (B < D) {
        return (A); }
    else {
        return (A + 10); }
}
main() {
    int F, J, K;
    float W, N, M;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    FVX66(W,F,K);
    printf("\n");
}
9) int FAJ68 (int A, float B, ... ) {
    float X, Y;
    int Q, Z;
    return (A + floor(B + 2));
}
main() {
    int F, J, K;
    float W, N, M;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    printf("%i\n",FAJ68(K,N,J,M));
}
10) int FAA23 (int A, int B, int C) {
    int X, Y, Z;
    if (A >= B) {
        return C; }
    else {
        return (A + B); }
}
main() {
    int F, J, K;
    float W, N, M;
    scanf("%i %i %i %f %f",&F,&J,&K,&N,&M);
    printf("%i\n",FAA23(F,J,K));
}
}
```



Algoritmos de Calendário

Cálculo do dia da semana

O algoritmo seguinte é devido ao astrônomo napolitano Aloysius Lilius e ao matemático alemão e jesuíta Cristopher Clavius. Escrito no século XVI é usado pelas igrejas ocidentais para calcular o dia do domingo de Páscoa. Existiram outros algoritmos antes deste. Por exemplo o *Canon Paschalis* devido a Victorius de Aquitania escrito em cerca de 450 a.C.

Dada uma data no formato dia, mes, ano (onde ano > 1587), calcula-se o dia da semana usando a seguinte formulação:

```
def semana(dia,mes,ano):  
    a=(12-mes)//10  
    b=ano-a  
    c=mes+(12*a)  
    d=b//100  
    e=d//4  
    f=e+2-d  
    g=np.floor(365.25*b)  
    h=np.floor(30.6001*(c+1))  
    i=f+g+h+dia+5  
    r=i%7  
    return r
```

Se $R = 0$, dia, mes, ano é sábado, $R = 1$ é domingo, $R = 2$ é segunda, $R = 3$ é terça $R = 4$ é quarta, $R = 5$ é quinta e $R = 6$ é sexta-feira.

Exemplo: Calculemos o dia da semana de hoje,
dia ____ / ____ / ____

A ← _____ F ← _____
B ← _____ G ← _____
C ← _____ H ← _____
D ← _____ I ← _____
E ← _____ R ← _____

como $R =$ _____, o dia em questão é _____.

Cálculo dos feriados móveis

Antes de começar este cálculo, apenas uma curiosidade: Agora em 2024, há esforços de diversas igrejas cristãs de unificar e padronizar e sobretudo fixar a Páscoa, que deixaria de ser completamente móvel e passaria a ser semi-móvel, sendo celebrada no segundo domingo de abril. Carnaval e Corpus Christi continuariam a ser calculados como se verá a seguir.

Os 3 feriados móveis (terça de carnaval, sexta feira santa e Corpus Christi) são baseados todos no dia do domingo de Páscoa. Portanto, a primeira coisa a fazer é calcular em que dia cai a Páscoa.

Dado um ano com quatro dígitos (maior que 1587), a Páscoa é:

```
def pascoa(ano):  
    dm=[31,28,31,30,31,30,31,31,30,31,30,31]  
    dp=[0]*12;  
    a=ano%19  
    b=ano//100  
    c=ano%100  
    d=b//4  
    e=b%4  
    f=(b+8)//25  
    g=(1+b-f)//3  
    h=((19*a)+b+15-(d+g)) % 30  
    i=c//4  
    k=c % 4  
    l=(32+(2*e)+(2*i)-(h+k)) % 7  
    m=((a+(11*h)+(22*l))//451)  
    p=((h+1+14-(7*m))//31)  
    q=(h+1+14-(7*m)) % 31  
    return [q+1,p]
```

A Páscoa é o dia Q+1 do mês P.

Bissexto

A regra do bissexto pode ser assim descrita: sejam
 $R4 \leftarrow$ resto da divisão do ano por 4
 $R100 \leftarrow$ resto da divisão do ano por 100 e
 $R400 \leftarrow$ resto da divisão do ano por 400.
SE $R4=0 \wedge ((R100 \neq 0) \vee (R400 = 0))$ o ano é bissexto senão não é.
Em Python

```
def bissexto(ano):  
    r4=ano%4  
    r100=ano%100  
    r400=ano%400  
    if ((r4==0) and ((r100 !=0) or (r400==0))):  
        return True  
    else:  
        return False
```

A chave para calcular os demais feriados móveis (Terça de carnaval, Sexta Feira Santa e Corpus Christi) está em considerar que, conhecida a Páscoa, Carnaval ocorre 47 dias antes, Sexta Santa ocorre 2 dias antes e Corpus Christi, 60 dias depois.

Para aprender a calcular somas e subtrações envolvendo dias, precisamos obter a quantidade de dias por mês:

J	F	M	A	M	J	J	A	S	O	N	D
31	28/29	31	30	31	30	31	31	30	31	30	31

Vai-se desenvolver aqui apenas o processamento de anos NÃO bissextos. Fica por conta do aluno os ajustes a fazer em anos bissextos.

Somando acumuladamente e deslocando uma posição à direita, o vetor fica:

J	F	M	A	M	J	J	A	S	O	N	D
31	28	31	30	31	30	31	31	30	31	30	31
0	31	59	90	120	151	181	212	243	273	304	334

A última linha, na tabela acima nos ajuda a achar o ordinal de um dia dentro do ano. Por exemplo:

Qual o ordinal do dia 12 de maio ? Olhando a tabela acima que antes do dia 1/mai há 120 dias. Logo o dia 12/maio é o centésimo, trigésimo segundo dia ($120+12=132$).

Qual o ordinal do dia 25 de julho ? $181+25=206^\circ$ dia.

Responda, qual o ordinal de:

05/06	
10/07	
05/08	
15/11	

Já o caminho inverso, também usa a tabela acima. Por exemplo, qual o centésimo dia do ano ? Olhando a tabela percebe-se que é um dia de abril, pois $90 < 100 \leq 120$. então, $90 + d = 100$, e daí que o centésimo dia é o dia 10/abr.

Qual o duocentésimo dia ? É um dia de julho, pois $181 < 200 \leq 212$. então, $181 + d = 200$ ou 19/jul.

Responda, qual o dia correspondente ao ordinal:

178 ^o	
242 ^o	
243 ^o	
244 ^o	

Com esse conceito operacional, para calcular os feriados:

1. Ache o dia e mês da Páscoa (algoritmo do Lilius e Clavius)
2. Descubra se o ano em questão é bissexto
3. Monte o vetor de dias acumulados, mês a mês.
4. Ache o ordinal correspondente à Páscoa
5. Subtraia 47 (Carnaval), 2 (Sexta Santa) e some 60 (Corpus Christi)
6. Converta esses ordinais em datas usuais

Exemplo: Vamos calcular as 4 datas do ano de 2026 .

A ← _____	K ← _____
B ← _____	L ← _____
C ← _____	M ← _____
D ← _____	P ← _____
E ← _____	Q ← _____
F ← _____	ord(Páscoa) _____
G ← _____	ord(Carn) _____
H ← _____	ord(SexSan) _____
I ← _____	ord(CC) _____

E, com isso, o Carnaval é dia ____/____, a Sexta Santa é ____/____ e Corpus Christi é ____/____.

🔖 Para você fazer

* Calcule dia da semana do dia 29/ 4/2180 e informe: (0=sab, 1=dom,...6=sex) : _____

* Calcule o Carnaval de 2180 ____ / ____

* Corpus-Christi de 2180 ____ / ____

* Calcule dia da semana do dia 17/11/1764 e informe: (0=sab, 1=dom,...6=sex) : _____

* Calcule o Carnaval de 1764 ____ / ____

* Corpus-Christi de 1764 ____ / ____

Uma curiosidade: Notada por Martin Gardner (The Colossal Book of short puzzles & Problems, pág. 63) é que Oct 31 = Nov 27 = Dec 25, ou Halloween = Thanksgiving day = Christmas. Como é possível ?



Os calendários antigos tinham que conciliar 2 ciclos: o da lua e do sol. O da lua deu origem aos meses e o do sol aos anos. O problema é que um não é múltiplo do outro.

O ano solar médio tem a duração de aproximadamente 365 dias, 5 horas, 48 minutos e 46 segundos (365,2422 dias). O Mês lunar em média, dura 29,5 dias.

Note-se que a semana é o único ciclo que não tem a ver com a natureza e sim com a religião (é o sétimo dia, onde Deus descansou após a criação do mundo). Aliás no português essa história de Segunda, Terça, ... deriva do antigo hebreu quando só o sabath era nomeado e os demais eram numerados.

Em 45 AC, Julio Cezar criou o que se chamou: Calendário Juliano (365d + 1bissex/4 anos). O ano juliano tinha 14 min e 11 Seg a mais que o ano solar.

Em 1582, por conta das diferenças nas festas religiosas das diversas igrejas cristãs espalhadas pelo mundo, o papa Gregório XIII encomendou uma reforma no calendário e sumiram 10 dias de março. Dos anos terminados em 00 só os divisíveis por 400 seriam bissextos Demorou cerca de 250 anos a adoção deste calendário. Na Inglaterra só foi adotado em 1752, e daí passou-se de 2 a 14 de setembro.

Em 1793, após a Revolução Francesa, houve a proposta do calendário republicano. 12m x 30d e os últimos 5 dias do ano seriam feriados nacionais. Os meses tinham nomes poéticos: germinal, floral, thermidor, frutidor, brumaire...). Abolido por Napoleão em 1805.

Calendário Gregoriano

O calendário gregoriano é o calendário utilizado na maior parte do mundo, e em todos os países ocidentais; porém, existem países que não o aplicam como a China, Israel, Irã, Índia, Bangladesh, Paquistão, Argélia, etc. Foi promulgado pelo Papa Gregório XIII a 24 de Fevereiro do ano 1582 para substituir o calendário juliano. Gregório XIII reuniu um grupo de especialistas para reformar o calendário juliano e, passados cinco anos de estudos, foi elaborado o calendário gregoriano, que foi sendo implementado lentamente em várias nações. Oficialmente o primeiro dia deste calendário foi 15 de Outubro de 1582. O calendário gregoriano é o que hoje em dia se usa e distingue-se do juliano porque (a) Omitiram-se dez dias (de 5 a 14 de Outubro de 1582). (b) Corrigiu-se a medição do ano solar, estimando-se que este durava 365 dias solares, 5 horas, 49 minutos e 12 segundos, o equivalente a 365,2425 dias solares. (c) Acostumou-se a começar cada ano novo em 1 de Janeiro. Nem todos os anos seculares são bissextos. Para um ano secular ser bissexto tem de ser múltiplo de 400. Deste modo, evita-se a diferença (atraso) de três dias em cada quatrocentos anos existente no calendário juliano. A mudança para o calendário gregoriano deu-se ao longo de mais de três séculos. Primeiramente foi adotado por Itália, Portugal, Espanha e França; e de modo sucessivo, pela maioria dos países católicos europeus. Os países onde predominava o luteranismo e o anglicanismo tardariam a adotá-lo, caso da Alemanha (Baviera, Prússia e demais províncias) (1700) e Reino Unido (Inglaterra) (1751). A adoção deste calendário pela Suécia foi tão problemática que gerou até o dia 30 de fevereiro. A China aprova-o em 1912, a Bulgária em 1917, a Rússia em 1918, a Roménia em 1919, a Grécia em 1923 e a Turquia em 1927.

Manipulação de datas

No mundo comercial é frequente a necessidade de manipulação de datas, por exemplo:

- Vou fazer um empréstimo de 180 dias. Que dia ele vai vencer ?
- Este produto tem 1000 dias de garantia. Quando vence ?
- Fulano já viveu 7230 dias...
- Faltam 1011 dias para a minha aposentadoria...

Devido à irregularidade do calendário, é muito difícil (e portanto sujeito a erros) a manipulação direta das variáveis dia, mês e ano em processos de adição e subtração de datas.

Uma boa estratégia (certamente não a única) é criar um vetor de dias, onde o valor anterior corresponde à data de ontem, o atual é o dia de hoje e o próximo é amanhã. Vamos chamar este vetor de vetor cronológico.

Tendo construído este vetor (que é fixo e não precisa ser recalculado a cada momento), somar e subtrair datas se transforma no problema muito mais simples de somar e subtrair índices dentro do vetor.

Guardar uma data em um inteiro

Posto que uma data é um conjunto de 3 valores (dia, mês e ano, os 3 inteiros), para simplificar a construção do vetor cronológico, é conveniente combinar uma maneira de guardar os 3 valores em um único inteiro.

Isto pode ser feito aplicando a seguinte expressão:

data = (dia × 1000000) + (mes × 10000) + ano

Note que a expressão acima é para expressão de anos com 4 dígitos. É adequada para processamentos que envolvam virada de milênio. Se o intervalo de processamento de datas se situa dentro de um século, a expressão fica mais simples:

data = (dia × 10000) + (mes × 100) + ano

Para a construção do vetor cronológico é preciso estabelecer (combinar) com todos os que o usarão, qual o dia inicial do vetor (ou seja qual o valor de vetor[1]). Nada poderá ser feito com datas anteriores a esta.

Nesta folha, vai-se usar como data inicial 1º de janeiro de 1970, que vem a ser a data inicial de várias versões do sistema operacional Unix. Esta folha também vai realizar operações em 100 anos a partir dessa data, indo portanto até o ano de 2070.

Diante disso, o tamanho do vetor cronológico é de mais ou menos 36600 elementos (366 × 100 = 36600).

Próximo Dia

A função chave para construir o vetor cronológico é a função amanhã que recebe um valor para hoje e calcula quando é amanhã.

```
1: função AMANHÃ (hoje) : inteiro
2: dia ← ⌊ hoje ÷ 1000000
3: mes ← ⌊ (hoje-(dia × 1000000)) ÷ 10000
4: ano ← ⌊ (hoje-(dia × 1000000)+(mes× 10000))
5: se (dia = 31) ∧ (mes = 12)
6:   devolva 1010000+ano+1
7: senão
8:   UD ← 0
9:   se (dia=31) ∧ ((mes=1) ∨ (mes=3) ∨ (mes=5) ∨ (mes=7) ∨ (mes=8) ∨ (mes=10))
10:     UD ← 1
11:   fim{se}
12:   se (dia=30) ∧ ((mes=4) ∨ (mes=6) ∨ (mes=9) ∨ (mes=11))
13:     UD ← 1
14:   fim{se}
15:   se (((ano mod 4) = 0) ∧ (((ano mod 100) ≠ 0) ∨ ((ano mod 400) = 0)))
16:     BIS ← 1
17:   senão
18:     BIS ← 0
19:   fim{se}
20:   se (dia = 28) ∧ (mes = 2) ∧ (BIS = 0)
21:     UD ← 1
22:   fim{se}
23:   se (dia = 29) ∧ (mes = 2) ∧ (BIS = 1)
24:     UD ← 1
25:   fim{se}
26:   se UD = 1
27:     devolva 1000000+(10000 × (mes + 1))+ano
28:   senão
29:     devolva ((1000000 × (dia + 1))+ (10000 × mes) + ano
30:   fim{se}
31: fim{se}
```

Vetor Cronológico

Agora é hora de montar o vetor. Se ele for para um século, faça

```
1: inteiro : VC [36660]
2: VC[1] ← 01011970
3: i ← 1
4: enquanto i < 36660
5:   VC[i+1] ← AMANHÃ (VC[i])
6:   i ← i + 1
7: fim{enquanto}
```

Manipulações

- 1. Supondo que hoje é 7 de abril de 1976, qual o dia exato após 1000 dias ?

```
hoje ← 07041976
i ← 1
enquanto VC[i] ≠ hoje
  i ← i + 1
fim{enquanto}
devolva VC[i+1000]
```

- 2. Supondo que alguém hoje (dia 13 de novembro de 2012) está complementando 7000 dias de vida. Que dia ele nasceu ?

```
hoje ← 13112012
i ← 36600
enquanto VC[i] ≠ hoje
  i ← i - 1
fim{enquanto}
devolva VC[i-7000]
```

- 3. Beltrano nasceu dia 22 de julho de 1965. Ele é um sujeito meio esquisito, e decidiu que não vai celebrar seus aniversários e sim os seus quatrocentários. Ajude ele a calcular as datas em que ele deve comemorar os primeiros 10 quatrocentários (1 quatrocentário = 400 dias) após o nascimento.

```
hoje ← 22071965
i ← 1
enquanto VC[i] ≠ hoje
  i ← i + 1
fim{enquanto}
j ← 0
enquanto j < 4000
  escreva (VC[i+j])
  j ← j + 400
fim{enquanto}
```

Para você fazer

- a. Suponha que hoje é o dia 04/12/1985 e que você fez um empréstimo bancário de 1965 dias. A pergunta é: qual a data de vencimento desse empréstimo ?
- b. Suponha que hoje é o dia 10/10/2005 e o seu filho está comemorando 1485 dias de vida. A pergunta é: que dia, mês e ano ele nasceu ?
- c. Suponha que hoje é o dia 16/04/1999 e uma cápsula do tempo está sendo encerrada. Ela deve ser aberta na data 03/07/2028 . A pergunta é quantos dias a cápsula deve ficar encerrada ?

a.	b.	c.

- 1 - /

Este exercício busca desenvolver a habilidade de enfrentar um problema e buscar qual (ou quais) a estratégia que:

- resolve o problema, OU
- permite saber mais sobre o problema, OU
- permite desenvolver ferramentas para aprender mais sobre o problema, OU
- permita buscar algoritmos que resolveram problemas parecidos, OU
- ...

Perceba-se que nem sempre é possível obter uma solução 100% correta e eficaz. O mundo está cheio de problemas para os quais não se tem nenhuma solução.

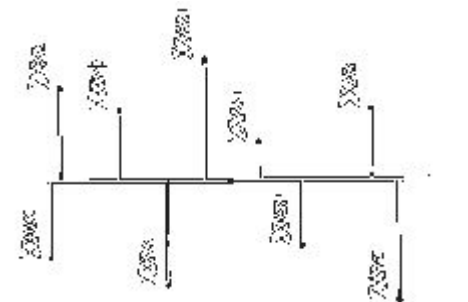
Entretanto, o engenheiro da computação e o analista de sistemas, até por sua formação, conhecimentos e sobretudo história de vida profissional, acaba meio que virando um resolvidor profissional de problemas. Não se assustem, quando durante suas vidas forem chamados a dar pitacos em situações envolvendo problemas que muitas vezes nada tem a ver com informática e computadores.

Este é o caso aqui. A ênfase não está em implementar um algoritmo, mas sim em achar (desenvolver) um.

Dito isto, vai-se ao problema:

Localização de oleoduto

Suponha que você tem um terreno de 100 km x 100 km contendo um certo número de poços de petróleo. O terreno irá ser atravessado integralmente por um duto mestre, ao qual irão se ligar (sempre de maneira transversal, isto é, fazendo 90 graus) ramais auxiliares que conectarão cada um dos poços ao duto mestre.



O objetivo do exercício é descobrir por onde deverá passar o duto mestre, de maneira a minimizar a quantidade de duto auxiliar necessário. O duto mestre sempre será instalado no sentido NORTE-SUL ou no LESTE-OESTE, conforme explicitado no exercício. Outra coisa que você deve calcular é a quantidade de duto auxiliar necessário (medido em quilômetros).

Os poços serão identificados pelas suas coordenadas (x, y), sendo que a origem dos índices, o ponto 0,0, estará localizado no canto inferior esquerdo do terreno. A primeira medida de cada poço (o x) indicará a distância no sentido leste-oeste e o segundo número (o y) indicará a distância no norte-sul.

Assim, se um poço estiver nas coordenadas (12,78), ele se encontrará 12 quilômetros à direita da origem e 78 quilômetros acima da mesma origem.

Veja-se o seguinte exemplo: Suponha-se a seguinte distribuição de poços:

x	y
11	9
10	26
17	19
5	14
16	22
13	20
1	12
26	4
15	25

Supondo que o duto principal deva ser estendido no sentido leste-oeste, em qual ordenada ele deveria estar ?

Proposta de solução

1. lembre que em (x, y), o x é a abcissa e o y é a ordenada.
2. Como o sentido é leste-oeste, podem-se desprezar as abcissas.
3. Ficando apenas com as ordenadas, têm-se: 9, 26, 19, 14, 22, 20, 12, 4, 25.
4. É razoável se supor que o duto esteja entre o ponto 4 e o ponto 26.
5. Para descobrir onde deve passar suponha-o localizado em cada um dos 9 poços e meça a quantidade de duto auxiliar necessário. Neste caso temos: 80, 83, 54, 59, 61, 55, 65, 115, 76
6. Como se pode ver o menor valor é de 54 km, correspondendo à ordenada 19, do terceiro poço.
7. A resposta esperada, portanto é 54,19, significando que são necessários 54km de duto auxiliar e que o duto principal deve estar entre os pontos 0,19 e 100,19.

Se o mesmo valor mínimo for encontrado para 2 poços, a resposta deverá ser a média entre as duas ordenadas. Por exemplo, no problema

x	y
19	13
10	12
4	17
1	5
7	23
18	20
8	16
21	11

Percebe-se que as distâncias mínimas são: 35, 37, 37, 77, 67, 49, 35, 41 que correspondem aos poços 1 e 7. Com isso, a ordenada do duto principal deve ser a média das duas ordenadas dos poços: $13 + 16 = 29 \div 2 = 14.5 \text{ km}$. A resposta deveria ser 35, 14.5. Lembrando, o valor 35 corresponde a quantidade de duto auxiliar necessária.

Observações

Lembre que se o duto principal estiver na direção norte-sul, vale o mesmo raciocínio visto acima, apenas mudando “ordenadas” por “abcissas”, e desprezando-se o segundo número das coordenadas de cada poço.

Finalmente, o algoritmo aqui esboçado, a quem poderíamos chamar de “força bruta”, funciona e apresenta o resultado correto. Entretanto você é livre para procurar outro algoritmo que seja mais eficiente, (mas que funcione também, é claro.)

Para você fazer

Exercício 1 Suponha o duto principal no eixo E-W . Os poços se encontram nas seguintes posições:

x	y
7	18
9	13
20	27
1	24
4	12
14	11
21	10
22	15
23	25

Para este caso, a metragem de duto auxiliar é de _____ e a posição do duto principal deve ser de _____.

Exercício 2 Suponha o duto principal no eixo N-S . Os poços se encontram nas seguintes posições:

x	y
29	26
12	3
23	25
20	7
9	27
15	30
11	24
10	19
16	17
6	4

Para este caso, a metragem de duto auxiliar é de _____ e a posição do duto principal deve ser de _____.

Exercício 3 Suponha o duto principal no eixo N-S . Os poços se encontram nas seguintes posições:

x	y
8	9
20	16
1	7
12	10
19	15
6	5
11	21

Para este caso, a metragem de duto auxiliar é de _____ e a posição do duto principal deve ser de _____.

Este exercício foi adaptado de CORMEN, T. et alli, Algoritmos, pág 155 (edição brasileira).



Este exercício busca desenvolver a habilidade de enfrentar um problema e buscar qual (ou quais) a estratégia que:

- resolve o problema, OU
- permite saber mais sobre o problema, OU
- permite desenvolver ferramentas para aprender mais sobre o problema, OU
- permita buscar algoritmos que resolveram problemas parecidos, OU
- ...

Perceba-se que nem sempre é possível obter uma solução 100% correta e eficaz. O mundo está cheio de problemas para os quais não se tem nenhuma solução.

Entretanto, o analista de sistemas e o engenheiro da computação, até por sua formação, conhecimentos e sobretudo história de vida profissional, acaba meio que virando um resolvidor profissional de problemas. Não se assustem, quando durante suas vidas forem chamados a dar pitacos em situações envolvendo problemas que muitas vezes nada tem a ver com informática e computadores.

Este é o caso aqui. A ênfase não está em implementar um algoritmo, mas sim em achar (desenvolver) um.

Dito isto, vai-se aos problemas:

1. Exponenciação Eficiente

Suponha que você está trabalhando em um ambiente de programação onde não existe a função de exponenciação de expoentes inteiros ($y = x^n$), onde n é sempre um número inteiro maior que 0.

Uma possibilidade simples e prática é escrever a função de exponenciação como segue (em C)

```
1: float EXPO(float X, int N) {  
2: int CT = 1;  
3: float RES = 1;  
4: WHILE (CT <= N) {  
5: RES = RES * X;  
6: CT = CT + 1;  
7: }  
8: RETURN RES;  
9: }
```

Matematicamente e imaginando um exemplo, (seja n^{15}), isto equivale a fazer:

$$n \times n \times n \times \dots \times n = n^{15}$$

ou seja equivale a fazer 14 multiplicações.

Entretanto, para calcular esse mesmo n^{15} , pode usar um método “binário” e achar o mesmo resultado com seis multiplicações:

$$\begin{aligned} n \times n &= n^2 \\ n^2 \times n^2 &= n^4 \\ n^4 \times n^4 &= n^8 \\ n^8 \times n^4 &= n^{12} \\ n^{12} \times n^2 &= n^{14} \\ n^{14} \times n &= n^{15} \end{aligned}$$

Mas, ainda é possível melhorar isso, usando apenas 5 multiplicações:

$$\begin{aligned} n \times n &= n^2 \\ n^2 \times n &= n^3 \\ n^3 \times n^3 &= n^6 \\ n^6 \times n^6 &= n^{12} \\ n^{12} \times n^3 &= n^{15} \end{aligned}$$

A seguir vai-se definir a função $m(k)$ como sendo o número mínimo de multiplicações necessárias para calcular n^k . Usando o exemplo acima, ter-se-ia $m(15) = 5$.

👉 Para você fazer

Calcule $m(k)$, para o valor de $k = 53$.

escreva aqui o valor

Este exercício foi adaptado do portal de problemas matemáticos [projecteuler.net](#). É o problema 122.

2. Diagonais

Suponha uma matriz que tem a unidade no centro e movendo-se para a direita na direção dos ponteiros do relógio forma o seguinte padrão neste caso de 5×5 :

21	22	23	24	25
20	7	8	9	10
19	6	1	2	11
18	5	4	3	12
17	16	15	14	13

Pode-se ver que neste caso a soma dos números que estão na diagonal é de 101. (Note que o 1 só é contado uma única vez).

👉 Para você fazer

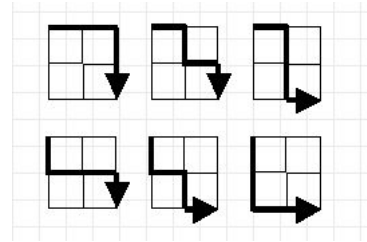
Ache a soma dos números que estão na diagonal de uma matriz espiral formada do mesmo jeito de ordem 75.

escreva aqui o valor

Este exercício foi adaptado do portal de problemas matemáticos [projecteuler.net](#). É o problema 28.

3. Caminho na treliça

Começando no canto esquerdo superior e movendo-se para a direita e para baixo. há exatamente 6 rotas desde o canto superior esquerdo até o inferior direito em uma treliça 2×2 .



👉 Para você fazer

Quantas rotas diferentes podem ser estabelecidas em uma treliça de 10 por 10?

escreva aqui o valor

Este exercício foi adaptado do portal de problemas matemáticos [projecteuler.net](#). É o problema 15. O mesmo problema apareceu no [mathschallenge.net](#) em maio de 2004.

4. Problema da média

Lembrando um pouco de estatística, dada uma distribuição de n números, a média é $\frac{\sum n}{n}$, a moda é o valor mais frequente da distribuição e a mediana é o valor que ocupa a posição central da distribuição quando os valores da mesma são ordenados.

Dado um conjunto de 5 números inteiros, sua média é 64, sua moda é 16, e a sua mediana é 80. Como informação adicional, pode-se afirmar que a diferença entre os 2 números maiores é igual à moda. Quais são os 5 números?

Este exercício foi adaptado do [mathschallenge.net](#) onde é o problema 233.



Aritmética Decimal, binária e Hexadecimal A aritmética tal como a conhecemos se desenvolveu a partir dos últimos 10.000 anos. Nesse aspecto, a matemática é privilegiada. Estivéssemos falando de ciências sociais ou mesmo de tecnologia aplicada, estaríamos tratando do último modismo, ou da última invenção. Mas, a matemática é eterna. Pitágoras e Euclides continuam tão verdadeiros e frescos (no bom sentido, por favor) como eram há 2400 anos. Aliás, se algum dia, descobrirmos uma civilização extraterrena, com biologia completamente distinta, seres com 7 braços e 3 pernas que vivem a bilhões de anos luz da Terra, pode-se garantir: o Teorema de Pitágoras lá continua valendo: $a^2 + b^2 = c^2$ ainda que eventualmente tenha outro nome e outros símbolos, talvez o Pitágoras deles seja um camarão inteligente. É batata!

A primeira grande invenção na aritmética foi o número zero (devido a matemáticos indianos). Com ele criou-se um sistema de notação posicional. A base escolhida foi a decimal, muito provavelmente devido a termos 10 dedos, já que estes são os primeiros e muito efetivos instrumentos de contagem de que dispõe o ser humano.

Eis os fatos sabidos até aqui: nosso sistema de numeração é decimal, (base=10), posicional e usa os dígitos arábicos 0,1,...,9. Uma curiosidade: embora diversos povos do planeta usem distintos alfabetos para a escrita, (cirílico, katakana, hangul, árabe, grego,...) todos os principais parecem concordar com os dígitos arábicos para escrever números. Assim o número 1003 é 1003 em qualquer rincão culto da Terra.

A mecânica da contagem é usar os 9 primeiros dígitos, começando no um e ao chegar ao 10, que é quando se completa uma base, e se acabam os dígitos unitários, escreve-se o um (o primeiro) seguido de um zero.

Aqui, a primeira novidade. Nenhuma generalidade se perde (e se ganha espaço) se a origem das contagens for o zero e não o um. Dentro da ciência da computação, em muitos momentos a origem da contagem será o zero. Fiquemos espertos.

Quando, os engenheiros ingleses construíram o primeiro computador de que se tem notícia, seguindo os planos teóricos do matemático Alan Turing (esse sim, provavelmente foi um extraterrestre...) estiveram diante de uma decisão tecnológica: como representar uma quantidade dentro de uma máquina?

A primeira hipótese foi fazer uma analogia com um circuito elétrico e variar alguma grandeza física (elétrica) proporcionalmente à quantidade em estudo. Aliás alguns computadores analógicos foram assim construídos. Imprecisão, dificuldades imensas na programação, e baixa generalidade foram fatais a este projeto. O caminho estava em outro lugar: em computadores digitais. Aliás, esta tendência que começou na década de 40, está hoje entre nós na digitalização de imagens, sons, filmes, livros, raios-x, relógios etc etc.

Ao construir uma variável digital (ao invés de uma analógica) é possível controlar sua exatidão, garantir

ausência de erros e obter a precisão tão grande quanto se queira. A pergunta agora é: que tipo de variável? A tentação é criar uma variável decimal (com 10 estados digitais). Mais fácil de falar do que fazer. Imaginem um circuito que a 0V indica 0, 1V indica 1, ... até 9V indica 9. Basta cair um raio, haver uma interferência e um 7 vira 9 fácil, fácil. A opção foi radical: variáveis binárias, contendo apenas dois valores, e completamente opostos um do outro. Digamos que 0 é +5V e 1 é -5V. Ao cair um raio, um 5V pode virar 4V ou até 3V, mas o circuito é suficiente "esperto" para não se confundir: 3V está muito mais perto de 5V do que de -5V.

Assim, chega-se à aritmética binária: a base de todos os computadores digitais (e mais genericamente de toda a eletrônica digital). Toda a circuitaria de um computador está construída assim. Deixando de lado os detalhes sórdidos (como 5V e -5V) vamos passar a usar a aritmética binária: apenas dois dígitos 0 e 1.

Continuam valendo TODAS as regras da aritmética: adição, produto, subtração, divisão, logaritmo, etc etc.

Acompanhe no exemplo: $0+0=0$, $0+1=1$, $1+0=1$ e $1+1=0$ e vai um ou seja, $1+1=10$. Resta um problema: o tamanho dos números: Por exemplo, o número $55786803_{(10)}$ é igual a $11010100110011110100110011_{(2)}$. Um número em base 2 contém em média $\log_{10} \div \log_2 = 3,32$ algarismos a mais do que em base 10. Nada que assuste, principalmente tendo em vista a aumento explosivo dos limites máximos de memória de qualquer circuitoquinho.

Por tudo isso, a unidade básica de registro (chamado, algo incongruamente de memória) em um computador é o BIT (Binary digiT), que como vimos pode conter 0 ou 1. Usa-se abreviar o BIT por um "b" minúsculo. Devido à pobreza desse elemento individual, os bits são agrupados em unidades maiores, chamadas BYTES e compostas por 8 bits. Usa-se abreviar o BYTE por um "B" maiúsculo. Deve-se este conceito ao engenheiro americano Claude Shannon. Se, em um bit só podem ser armazenados dois valores, em um byte, podem-se guardar $2^8 = 256$ estados, devidamente numerados de 0 até 255.

Sistema Hexadecimal Os dígitos binários são tudo o que o computador consegue manusear. Mas, vimos que se torna cansativo e sujeito a erros para o ser humano, manipular cadeias enormes de zeros e uns. A solução para isso, está em usar um segundo sistema, o hexadecimal. A palavra hexadecimal tem a ver com a base=16. Ressalte-se que ele é uma modificação do sistema binário (que insisto: é o único que o computador entende), criada exclusivamente para facilitar a vida dos humanos que têm que pensar em binário. Essa simplificação consiste em trocar 4 dígitos binários por 1 único dígito hexadecimal. Em um sistema de base 16, vai-se precisar de 16 dígitos. Na falta de melhor opção, vão-se usar os 10 dígitos numéricos (0..9) e depois 6 letras começando no A (A..F).

Antes de mostrar as tabelas, vamos fazer uma contagem:

em decimal 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...

em binário 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001, 10010, 10011, 10100, ...

em hexadecimal 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, ...

Agora, uma tabela de conversão:

decimal	binário	hexad.
0	0000.0000	00
1	0000.0001	01
2	0000.0010	02
9	0000.1001	09
10	0000.1010	0A
15	0000.1111	0F
16	0001.0000	10
255	1111.1111	FF

Perceba-se que dentro de um byte (8 bits) é costume pôr um ponto separando as duas metades do byte. É mais ou menos o mesmo que escrever 34.567 ao invés de 34567. Facilita a leitura.

Exercício Construa a tabela acima para os 256 primeiros valores. Comece em 0 e termine em 255.

Conversão de base 2 para base 10 Multiplique cada dígito pelas potências crescente de 2, começando à esquerda com 2^0 e some o resultado. Acompanhe no exemplo:

Quanto é $110100_{(2)} = ?_{(10)}$ Faça-se: $0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 = 0 + 0 + 4 + 0 + 16 + 32 = 52 = 52_{(10)}$.

Conversão de base 16 para base 10 A mesma coisa: Multiplique cada dígito pelas potências crescente de 16, começando à esquerda com 16^0 e some o resultado. Acompanhe no exemplo:

Quanto é $02CA_{(16)} = ?_{(10)}$ Faça-se: $A \times 16^0 + C \times 16^1 + 2 \times 16^2 + 0 \times 16^3 = 10 \times 16^0 + 12 \times 16^1 + 2 \times 16^2 + 0 \times 16^3 = 10 + 192 + 512 + 0 = 714_{(16)}$.

Conversão de base 10 para base 2 A regra é através da apropriação dos sucessivos restos da divisão inteira do número por 2. Acompanhe no exemplo:

Quanto é $52_{(10)} = ?_{(2)}$. Divida-se 52 por 2. O resultado é 26 e o resto é 0. Depois $26 \div 2 = 13$ e o resto é 0. Depois $13 \div 2 = 6$ e o resto é 1. Depois $6 \div 2 = 3$, resto 0. $3 \div 2 = 1$, resto 1. $1 \div 2 = 0$, resto 1. Tomando os restos de trás para a frente, obtem-se o número binário: 110100, que é o número buscado.

Conversão de base 10 para base 16 A regra é a mesma: a apropriação dos sucessivos restos da divisão inteira do número por 16. Acompanhe no exemplo:

Quanto é $714_{(10)} = ?_{(16)}$. Divida-se 714 por 16. Dá 44, com resto 10. Depois, $44 \div 16 = 2$, resto 12. Depois, $2 \div 16 = 0$, resto 2. Tomando os restos fica 2.12.10 ou mais propriamente 2CA, que é o número buscado. Para a realização deste tipo de raciocínio, convém decorar a seguinte tabela de potências crescentes de 2 (e de 16)



2^0	1		
2^1	2	1 b	
2^2	4		
2^3	8		
2^4	16		
2^5	32		
2^6	64		
2^7	128		
2^8	256	1 B	10^0 B
2^9	512		
2^{10}	1024	1 KB	$\approx 10^3$
2^{12}	4096		
2^{16}	65536		
2^{20}	1048576	1 MB	$\approx 10^6$
2^{30}		1 GB	$\approx 10^9$
2^{40}		1 TB	$\approx 10^{12}$
2^{50}		1 PB	$\approx 10^{15}$
2^{60}		1 EB	$\approx 10^{18}$
2^{70}		1 ZB	$\approx 10^{21}$
2^{80}		1 YB	$\approx 10^{24}$

Adição em binário trivial, assim como a soma em base decimal. Veja-se as regras (são apenas 4) que estão no começo desta folha.

Adição em hexadecimal A regra é a mesma da adição em decimal. Somam-se os valores das duas parcelas e se o resultado é igual ou maior do que a base, tira-se uma base e "vai um". Acompanhe nos exemplos:

02CAFE	166A4
+ 00765B	+ 22BB7
-----	-----
34159	3925B

 **Para você fazer**
Resolva os exercícios seguintes e informe os resultados na base pedida.

- $10110011_{(2)} = ?_{(10)}$
- $F01_{(16)} = ?_{(10)}$
- $219_{(10)} = ?_{(2)}$
- $211_{(10)} = ?_{(2)}$
- $2782_{(10)} = ?_{(16)}$
- $4349_{(10)} = ?_{(16)}$
- $11011111_{(2)} + 10101101_{(2)} = ?_{(2)}$
- $11010100_{(2)} + 110110010_{(2)} = ?_{(2)}$
- $1353_{(16)} + 1FB2_{(16)} = ?_{(16)}$
- $D9E_{(16)} + 1074_{(16)} = ?_{(16)}$

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.

Aritmética para qualquer base

Esta folha é um resumo (fechamento) de todo o assunto referente a bases não decimais. A seguir, as regras básicas de conversão e de adição e subtração em diversas bases.

Base A base pode ser qualquer número entre 2 e *k*, sendo *k* tão grande quanto se queira. O problema é que para um sistema de base *k* são necessários *k* símbolos diferentes. Como vamos usar os dígitos de 0 a 9 e o alfabeto de 26 letras, estaremos limitados à base=36.

Para tanto deve-se usar a seguinte tabela de valores de cada símbolo

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
										1	1	1	1	1	1	1	1	1	1	2
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z						
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---						
2	2	2	2	2	2	2	2	2	2	3	3	3	3	3						
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5						

Observação importante: por causa da dificuldade de distinguir zero da letra O, esta última vai ser apresentada como um o acentuado com circunflexo, assim: ô.

Para seu conhecimento, o QRCODE, usa no registro de caracteres, um base de numeração de base 45. O protetor de conteúdos conhecido como Base64, usa um sistema de numeração de base 64, como aliás é o seu nome. Já o sistema de pagamentos Bitcoin usa um sistema de numeração de base 58.

Conversão de base qualquer para 10 Para converter um número escrito em uma base qualquer, deve-se multiplicar cada dígito pelas potências crescentes da base e somar o resultado. Por exemplo:
Seja converter $CDF_{33} \rightarrow ?_{10}$
 $C = 12 \times 33^2 = 12 \times 1089 = 13068$
 $D = 13 \times 33^1 = 13 \times 33 = 429$
 $F = 15 \times 33^0 = 15 \times 1 = 15$
 $13068 + 429 + 15 = 13512$

Outro exemplo:
Seja converter $3264_7 \rightarrow ?_{10}$
 $3 \times 7^3 = 3 \times 343 = 1029$
 $2 \times 7^2 = 2 \times 49 = 98$
 $6 \times 7^1 = 6 \times 7 = 42$
 $4 \times 7^0 = 4 \times 1 = 4$
 $1029 + 98 + 42 + 4 = 1173$

Conversão de base 10 para qualquer Divida sucessivamente o número decimal pela base. Prossiga até obter um quociente igual a zero. Daí recolha os restos em ordem reversa. Zeros adicionais podem ser acrescentados à esquerda. Por exemplo:
Seja converter $13512_{10} \rightarrow ?_{33}$
 $13512 \div 33$ dá 409 e o resto é 15.
 $409 \div 33$ dá 12 e o resto é 13.
 $12 \div 33$ dá 0 e o resto é 12.
Montando o resultado (de trás para a frente tem-se 12=C, 13=D e 15=F) e finalmente o resultado é CDF_{33}

Seja converter $1173_{10} \rightarrow ?_7$
 $1173 \div 7$ dá 167 e o resto é 4.
 $167 \div 7$ dá 23 e o resto é 6.
 $23 \div 7$ dá 3 e o resto é 2.
 $3 \div 7$ dá 0 e o resto é 3.
O número procurado é 3264_7 .

Soma em base diferente de 10 Monte (desenhe) a operação como você sempre fez. Anote bem visível qual a base usada. Opere da direita para a esquerda. Se necessário consulte a tabela dos valores das letras. A regra é **SE a soma é igual ou maior do que a base ENTÃO subtraia uma base do resultado, anote o dígito**

encontrado e VAI UM. Note que é sempre VAI UM e não vai dois, três, ...
Por exemplo:
Seja a soma de $1H8D_{20} + 1DE0_{20} = ?_{20}$.
Faça $D + 0 = D$ e lance *D* na resposta. Depois, faça $8 + E = 8 + 14 = 22$. Como $22 \geq base$, subtraia uma base de 22 e lance o resultado 2. Como foi um, faça $H + D + 1 = 17 + 13 + 1 = 31$. Como $31 \geq base$, subtraia uma base ($31 - 20 = 11 = B$) e vái um. Finalmente $1 + 1 + 1 = 3$ e a conta está feita: o resultado é $3B2D$.

1 H 8 D
+ 1 D E 0

3 B 2 E b=20

Mais um exemplo, seja $26006_8 + 21054_8 = ?_8$. Fica

2 6 0 0 6
+ 2 1 0 5 4

4 7 0 6 2 b=8

Subtração em base diferente de 10 Monte (desenhe) a operação como você sempre fez. Anote bem visível qual a base usada. Opere da direita para a esquerda. Se necessário consulte a tabela dos valores das letras. A regra é **SE o dígito de cima é menor do que o de baixo, EMPRESTE uma base do dígito imediatamente à esquerda.** Prossiga normalmente.

Seja um exemplo $DH8_{29} - 6S4_{29} = ?_{29}$.
Faça $8 - 4 = 4$ sem nenhuma dificuldade. Depois, faça $H (=17) - S (=28)$. Como $17 < 28$ há necessidade de emprestar uma base e fica $17 + 29 = 46$. Agora pode-se fazer $46 - 28 = 18$. Consultando a tabela vê-se que $18=I$. Segue a conta: $D (=13)$ menos um fica $C (=12)$ menos 6, dá como resposta 6. E a conta fica:

D H 8
- 6 S 4

6 I 4 b=29

Mais um exemplo:

3 7 4 5 3
- 1 6 2 1 4

2 1 2 3 7 b=8

Para você fazer

Resolva os exercícios seguintes e informe os resultados na base pedida.

1. $10 \rightarrow Q$: faça as conversões abaixo e responda na base pedida:

converta	base	resposta
7024	33	
17753	28	
7640	23	
10917	19	
10728	25	
7950	4	
12480	30	
5712	19	
16303	5	
18387	8	

2. $Q \rightarrow 10$: faça as conversões abaixo e responda na base decimal:

converta	base	resposta
143320	6	
26513	9	
MG9	25	
G79	23	
P06	26	
25031	9	
ô0P	27	
3A6F	16	
28205	9	
KBS	31	

3. **Soma em base Q:** faça as adições abaixo e responda na mesma base em que estão os operandos.

		b.	resposta
1G57	BHI	22	
7A01	4356	11	
5FH	95N	34	
1530	1B66	17	
40351	105043	7	
JKL	19CA	24	
1BE1	4117	16	
JM6	8JT	30	
14363	31252	8	
D0X	5KI	36	

4. **Subtração em base Q:** faça as subtrações abaixo e responda na mesma base em que estão os dois operandos.

		b.	resposta
13735	560A	11	
3A45	217A	14	
G91	C1H	34	
16JM	ô5B	25	
77B0	4843	13	
35020	22661	7	
1DD3	IA2	23	
2EA8	1D14	18	
15B8	ô1C	25	
28BJ	JB7	20	

5. **Soma em bases diferentes:** para este exercício cada operando e a resposta estão em bases diferentes. O resultado deve ser calculado na última base.

	b.		b.	resposta	b.
57A6	13	13A2	22		25
KME	26	DRJ	29		30
KF4	25	5QF	35		18
22872	9	HEL	30		21
BBE	27	DDK	33		23
2AB8	18	8UQ	35		26
11603	11	B2BA	12		21
9N7	36	A7A	27		22
A9E	25	AML	28		26
1539	22	AW4	34		28

6. **Subtração em bases diferentes:** Para este exercício cada operando e a resposta estão em bases diferentes. O resultado deve ser dado na última base

	b.		b.	resposta	b.
1861	20	1A20	17		31
GRI	34	406B	12		26
26302	9	EUS	33		21
5520	12	CG6	23		14
10CC	20	6FC	31		15
B8ô	25	6RN	29		13
L92	27	99S	33		20
8562	13	5A36	11		22
2138	18	C2C	22		30
4CA6	15	4339	11		9



Base64

Base64 é um sistema de numeração posicional que usa 64 como base. Ao produzir um texto que possa ser imprimível em qualquer impressora do mundo ocidental, sem nenhuma chance de erro ou incoerência, apenas podem ser usados os caracteres A..Z, a..z e 0..9. Note que tem-se aqui 26+26+10=62 caracteres. Para completar o conjunto esperado de 64 caracteres, costuma-se usar "+“e “/“, nesta ordem. (Note-se que muitos programas que usam este princípio costumam escolher outros 2 caracteres que não sejam "+“e “/“, mas nenhum deles usa o rótulo de Base64. São exemplos deste caso: a B64 modificado para URL (usa "-“e “_“) ou B64 para expressões regulares que usa ("!“e "-“).

Há ainda um caracter importante, o “=”(em Base64) que serve para sinalizar eventuais necessidades de preechimento de *stuffing*.

A idéia desta codificação é poder mandar qualquer conteúdo binário como parte de uma mensagem de e-mail, sem correr o risco de algum conteúdo parcial acabe gerando caracteres de controle do protocolo e ponha a perder a comunicação.

A conversão se dá trocando 3 bytes (8 bits × 3 = 24 bits) em 4 caracteres Base64, cada um deles representado por 6 bits. (então 6 bits × 4 = 24 bits).

Na codificação de 8 bits/byte, usa-se um código qualquer. Atualmente a grande maioria de ambientes usa o padrão ASCII, cuja pequena parte pode ser vista aqui.

uma parte da Tabela ASCII

Binário	Dec	Hex	Car.
0010.1000	40	28	(
0010.1001	41	29	)
0010.1010	42	2A	*
0010.1011	43	2B	+
0011.1100	44	2C	,
0010.1101	45	2D	-
0010.1110	46	2E	.
0010.1111	47	2F	/
0011.0000	48	30	0
0011.0001	49	31	1
0011.0010	50	32	2
0011.0011	51	33	3
0011.0100	52	34	4
0011.0101	53	35	5
0011.0110	54	36	6
0011.0111	55	37	7
0011.1000	56	38	8
0011.1001	57	39	9

0100.0001	65	41	A
0100.0010	66	42	B
0100.0011	67	43	C
0100.0100	68	44	D
0100.0101	69	45	E
0100.0110	70	46	F
0100.0111	71	47	G
0100.1000	72	48	H
0100.1001	73	49	I
0100.1010	74	4A	J
0100.1011	75	4B	K

0100.1100	76	4C	L
0100.1101	77	4D	M
0100.1110	78	4E	N
0100.1111	79	4F	O
0101.0000	80	50	P
0101.0001	81	51	Q
0101.0010	82	52	R
0101.0011	83	53	S
0101.0100	84	54	T
0101.0101	85	55	U
0101.0110	86	56	V
0101.0111	87	57	W
0101.1000	88	58	X
0101.1001	89	59	Y
0101.1010	90	5A	Z
0110.0001	97	61	a
0110.0010	98	62	b
0110.0011	99	63	c
0110.0100	100	64	d
0110.0101	101	65	e
0110.0110	102	66	f
0110.0111	103	67	g
0110.1000	104	68	h
0110.1001	105	69	i
0110.1010	106	6A	j
0110.1011	107	6B	k
0110.1100	108	6C	l
0110.1101	109	6D	m
0110.1110	110	6E	n
0110.1111	111	6F	o
0111.0000	112	70	p
0111.0001	113	71	q
0111.0010	114	72	r
0111.0011	115	73	s
0111.0100	116	74	t
0111.0101	117	75	u
0111.0110	118	76	v
0111.0111	119	77	w
0111.1000	120	78	x
0111.1001	121	79	y
0111.1010	122	7A	z

Para efeito de conversão entre os 2 códigos (ASCII e Base64) deve-se usar o seguinte vetor de conversão:

ABCDEFGHIJKLMN
OPQRSTUVWXYZab
cdefghijklmnop
qrstuvwxyz0123
456789+/

Para numerar esta tabela, faça: A=0, B=1, C=2, D=3, E=4, F=5, G=6, H=7, I=8, J=9, K=10, L=11, M=12, N=13, O=14, P=15, Q=16, R=17, S=18, T=19, U=20, V=21, W=22, X=23, Y=24, Z=25, a=26, b=27, c=28, d=29, e=30, f=31, g=32, h=33, i=34, j=35, k=36, l=37, m=38, n=39, o=40, p=41, q=42, r=43, s=44, t=45, u=46, v=47, w=48, x=49, y=50, z=51, 0=52, 1=53, 2=54, 3=55, 4=56, 5=57, 6=58, 7=59, 8=60, 9=61, +=62 , /=63

ASCII → Base64

Deve-se pegar 3 bytes a converter (24 bits) e colocá-los lado a lado. Daí, o número binário obtido deve ser quebrado em 4 partes, cada uma com 6 bits. Como se sabe, um número binário de 6 bits varia entre 0..63. Este número obtido é que deve indexar o vetor acima estabelecido, dando como resultado 4 caracteres Base64, Acompanhe no exemplo: Seja passar o ASCII Aa) para Base64. Da tabela ASCII, sabe-se que A=0100.0001, a=0110.0001 e)=0010.1001. Juntando tudo,

fica: 0100.0001.0110.0001.0010.1001. Rearrumando, fica: 010000.010110.000100.101001, números binários que valem: 16, 22, 4 e 41, respectivamente. Obtendo os elementos 17, 23, 5 e 42 do vetor de elementos do Base64, fica-se com QWep. Então: Aa)*ascii* ≡ QWep*Base64*.

Um cuidado especial deve ser tomado quando se chega ao final do conjunto de caracteres a converter. No fim, podem sobrar nenhum, 1 ou 2 caracteres. Se sobrar nenhum, o problema está resolvido. Se sobraem 2 caracteres, estes devem ser colocados lado a lado (16 bits) e o conjunto deve ser completado com 2 zeros à direita, formando os 18 bits esperados. Para sinalizar estes 2 *stuffing bits*, após os 3 caracteres Base64 gerados, agrega-se o sinal “=”. Por exemplo, seja converter o string ASCII A1B2C. O primeiro conjunto A1B é convertido normalmente, gerando QTFC. Sobra 2C. Suas configurações em binário são 2=0011.0010 e C=0100.0011. Juntando tudo, fica 0011.0010.0100.0011 cujo comprimento é 16 bits. Agregando 2 zeros à direita e reordenando fica: 0011.0010.0100.0011.00 ou 001100.100100.001100, que correspondem a 12, 36 e 12 respectivamente. Obtendo os elementos 13, 37 e 13 do vetor formador, obtém-se M, k e M respectivamente. A maneira de informar que foram incluídos 2 zeros ao final é colocar um “=”como quarto caractere da conversão. Assim: 2C)*ascii* ≡ MkM=*Base64*.

Finalmente, se sobrar um único caractere (8 bits), o conversor precisa incluir 4 zeros à direita (*stuffing bits*). Por exemplo, para converter o ASCII “M”para Base64, faz-se: M=0100.1101, com 8 bits. Acrescentando os 4 zeros a direita, fica: 0100.1101.0000 e quebrando em dois números de 6 bits cada, fica: 010011.010000 que são os números 19 e 16. Obtendo o vigésimo e décimo sétimo elementos do vetor formador do Base64 obtem-se “TQ”. Para sinalizar que incluíram-se 4 zeros, o conversor acrescenta “==”. Com isso, M)*ascii* ≡ TQ==*Base64*.

Base64 → ASCII

A volta segue o mesmo procedimento, agora realizado em sentido inverso. Inicialmente, deve-se verificar que o comprimento da mensagem codificada em Base64 deve ter um número de caracteres múltiplos de 4. Se não for, há um erro. Tudo começa, verificando se o último caractere do grupo (de 4) em análise é igual a “=”. Se não for, este grupo não é terminal e deve sofrer o processamento padrão que é juntar os 24 bits (cada letra dá origem a um número de 6 bits, obtido pela localização de cada letra no vetor gerador do Base64) que devem ser reorganizados a fim de obter os 3 bytes em ASCII. Por exemplo, seja o conjunto Base64 “VGU4”. As posições destes caracteres (V, G, U e 4) no vetor formador são: 22, 7, 21 e 57. Subtraindo uma unidade (já que agora a origem dos índices é zero) tem-se 21, 6, 20 e 56. Convertendo tais números a binário tem-se 0 1 0 1 0 1, 0 0 0 1 1 0, 0 1 0 1 0 0 e 1 1 1 0 0 0. Juntando e reorganizando 0101.0100 0110.0101 e 0011.1000, que são “Te8”. Entao: VGU4*Base64* ≡ Te8)*ascii*.

Se o último caracter é “=”, então deve-se olhar o penúltimo. Se o penúltimo é diferente de “=”, então tem-se aqui 2 caracteres ASCII, e depois de juntar 3 x 6 bits = 18 bits, deve-se desprezar os últimos 2 bits e ficar com 16 bits que perfazem os 2 caracteres ASCII buscados.

Por exemplo, seja o Base64 igual a “QTY=”. Como o último é = e o penúltimo não é =, então este grupo codifica 2 caracteres ASCII.

Q, T e Y são os caracteres 17, 20 e 25 do vetor formador. Subtraindo 1 fica: 16, 19 e 24. Convertendo a binário fica: 0 1 0 0 0 0, 0 1 0 0 1 1 e 0 1 1 0 0 0. Juntando e desprezando os dois últimos zeros: 0100.0001, 0011.0110 e 00. Com isso os dois caracteres ASCII são: A6, que é a codificação original. Ou QTY=*Base64* ≡ A6)*ascii*.

Se o último e o penúltimo forem =, então é apenas um o caractere original. Deve-se converter os 2 caracteres iniciais (os que são diferentes de =) formando 12 bits e desprezar os últimos 4 bits.

Por exemplo, para desconverter “Rg==”, descobre-se que R e g são os elementos 18 e 33 no vetor formador. Subtraindo fica 17 e 32. Convertendo: 0 1 0 0 1 0 e 1 0 0 0 0 0. Reorganizando 0100.1010.0000. O caractere buscado é 'F'. Entao: Rg==*Base64* ≡ F)*ascii*.

Alguns exemplos

de	conversão	para
tMz	ASCII→b64	dE16
Dx	ASCII→b64	RHg=
6	ASCII→b64	Ng==
ZGxO	b64→ASCII	dIN
cU4=	b64→ASCII	qN
SQ==	b64→ASCII	I

Para você fazer

- Converta de ASCII para Base64
 - DE+
 - g6
 - i
 - jzL
- Converta de Base64 para ASCII
 - NnJq
 - ZCs=
 - Uw==
 - TGcr

a	b	c	d
e	f	g	h



Operação de contagem

Algo tão simples e que fazemos de maneira quase automática (outro dia vi num programa de televisão um cachorro que sabia contar) pode esconder alguma sutileza, principalmente quando se abandona a base decimal, nossa velha conhecida. A operação mental que é exigida aqui é separar o processo de contar do parâmetro base (que no dia a dia nosso é sempre 10).

Para podermos contar além da base 10, necessitamos de novos dígitos, já que uma regra dos sistemas de numeração posicionais estabelece que **necessitam-se tantos símbolos diferentes quanto é a base da numeração**. Assim, na base decimal, usam-se os 10 dígitos: de zero até nove.

Os novos dígitos que serão usados são os 10 dígitos decimais seguidos do alfabeto latino, formando a seguinte tabela

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3
										0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5										

Estabelecido o processo, a contagem segue sempre igual, ainda que a base tenha sido modificada. Veja-se a seguir uma tabela de contagens de 1 a 60 com diversas bases

bases =	2	3	4	5	6	7	8	9	11	12	13	14	15	16	20	25	30
1-000001	0001	001	001	001	001	001	01	01	01	01	01	01	01	01	01	01	01
2-000010	0002	002	002	002	002	002	02	02	02	02	02	02	02	02	02	02	02
3-000011	0010	003	003	003	003	003	03	03	03	03	03	03	03	03	03	03	03
4-000100	0011	010	004	004	004	004	04	04	04	04	04	04	04	04	04	04	04
5-000101	0012	011	010	005	005	005	05	05	05	05	05	05	05	05	05	05	05
6-000110	0020	012	011	010	006	006	06	06	06	06	06	06	06	06	06	06	06
7-000111	0021	013	012	011	010	007	07	07	07	07	07	07	07	07	07	07	07
8-001000	0022	020	013	012	011	10	08	08	08	08	08	08	08	08	08	08	08
9-001001	0100	021	014	013	012	11	10	09	09	09	09	09	09	09	09	09	09
10-001010	0101	022	020	014	013	12	11	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A
11-001011	0102	023	021	015	014	13	12	10	0B	0B	0B	0B	0B	0B	0B	0B	0B
12-001100	0110	030	022	020	015	14	13	11	10	0C	0C	0C	0C	0C	0C	0C	0C
13-001101	0111	031	023	021	016	15	14	12	11	10	0D	0D	0D	0D	0D	0D	0D
14-001110	0112	032	024	022	020	16	15	13	12	11	10	0E	0E	0E	0E	0E	0E
15-001111	0120	033	030	023	021	17	16	14	13	12	11	10	0F	0F	0F	0F	0F
16-010000	0121	100	031	024	022	20	17	15	14	13	12	11	10	0G	0G	0G	0G
17-010001	0122	101	032	025	023	21	18	16	15	14	13	12	11	0H	0H	0H	0H
18-010010	0200	102	033	030	024	22	20	17	16	15	14	13	12	0I	0I	0I	0I
19-010011	0201	103	034	031	025	23	21	18	17	16	15	14	13	0J	0J	0J	0J
20-010100	0202	110	040	032	026	24	22	19	18	17	16	15	14	10	0K	0K	0K
21-010101	0210	111	041	033	030	25	23	1A	19	18	17	16	15	11	0L	0L	0L
22-010110	0211	112	042	034	031	26	24	20	1A	19	18	17	16	12	0M	0M	0M
23-010111	0212	113	043	035	032	27	25	21	1B	1A	19	18	17	13	0N	0N	0N
24-011000	0220	120	044	040	033	30	26	22	20	1B	1A	19	18	14	06	06	06
25-011001	0221	121	100	041	034	31	27	23	21	1C	1B	1A	19	15	10	0P	0P
26-011010	0222	122	101	042	035	32	28	24	22	20	1C	1B	1A	16	11	0Q	0Q
27-011011	1000	123	102	043	036	33	30	25	23	21	1D	1C	1B	17	12	0R	0R
28-011100	1001	130	103	044	040	34	31	26	24	22	20	1D	1C	18	13	0S	0S
29-011101	1002	131	104	045	041	35	32	27	25	23	21	1E	1D	19	14	0T	0T
30-011110	1010	132	110	050	042	36	33	28	26	24	22	20	1E	1A	15	10	10
31-011111	1011	133	111	051	043	37	34	29	27	25	23	21	1F	1B	16	11	11
32-100000	1012	200	112	052	044	40	35	2A	28	26	24	22	20	1C	17	12	12
33-100001	1020	201	113	053	045	41	36	30	29	27	25	23	21	1D	18	13	13
34-100010	1021	202	114	054	046	42	37	31	2A	28	26	24	22	1E	19	14	14
35-100011	1022	203	120	055	050	43	38	32	2B	29	27	25	23	1F	1A	15	15
36-100100	1100	210	121	100	051	44	40	33	30	2A	28	26	24	1G	1B	16	16
37-100101	1101	211	122	101	052	45	41	34	31	2B	29	27	25	1H	1C	17	17
38-100110	1102	212	123	102	053	46	42	35	32	2C	2A	28	26	1I	1D	18	18
39-100111	1110	213	124	103	054	47	43	36	33	30	2B	29	27	1J	1E	19	19
40-101000	1111	220	130	104	055	50	44	37	34	31	2C	2A	28	20	1F	1A	1A
41-101001	1112	221	131	105	056	51	45	38	35	32	2D	2B	29	21	1G	1B	1B
42-101010	1120	222	132	110	060	52	46	39	36	33	30	2C	2A	22	1H	1C	1C
43-101011	1121	223	133	111	061	53	47	3A	37	34	31	2D	2B	23	1I	1D	1D
44-101100	1122	230	134	112	062	54	48	40	38	35	32	2E	2C	24	1J	1E	1E
45-101101	1200	231	140	113	063	55	50	41	39	36	33	30	2D	25	1K	1F	1F
46-101110	1201	232	141	114	064	56	51	42	3A	37	34	31	2E	26	1L	1G	1G
47-101111	1202	233	142	115	065	57	52	43	3B	38	35	32	2F	27	1M	1H	1H
48-110000	1210	300	143	120	066	60	53	44	40	39	36	33	30	28	1N	1I	1I
49-110001	1211	301	144	121	100	61	54	45	41	3A	37	34	31	29	16	1J	1J
50-110010	1212	302	200	122	101	62	55	46	42	3B	38	35	32	2A	20	1K	1K
51-110011	1220	303	201	123	102	63	56	47	43	3C	39	36	33	2B	21	1L	1L
52-110100	1221	310	202	124	103	64	57	48	44	40	3A	37	34	2C	22	1M	1M
53-110101	1222	311	203	125	104	65	58	49	45	41	3B	38	35	2D	23	1N	1N
54-110110	2000	312	204	130	105	66	60	4A	46	42	3C	39	36	2E	24	16	16
55-110111	2001	313	210	131	106	67	61	50	47	43	3D	3A	37	2F	25	1P	1P
56-111000	2002	320	211	132	110	70	62	51	48	44	40	3B	38	2G	26	1Q	1Q
57-111001	2010	321	212	133	111	71	63	52	49	45	41	3C	39	2H	27	1R	1R
58-111010	2011	322	213	134	112	72	64	53	4A	46	42	3D	3A	2I	28	1S	1S
59-111011	2012	323	214	135	113	73	65	54	4B	47	43	3E	3B	2J	29	1T	1T
60-111100	2020	330	220	140	114	74	66	55	50	48	44	40	3C	30	2A	20	20

Na primeira parte do exercício, você está sendo convidado a escrever a mesma sequência acima (de 1 a 60) para um sistema de numeração de base qualquer.

Na segunda parte do exercício, serão apresentadas 3 contas de adição em bases ignoradas. Pela análise do resultado, você deve inferir em qual base as contas foram feitas.

Por exemplo: suponha que a conta $EG+2GD=3A8$ lhe é apresentada, junto com a pergunta: Em que base isso foi feito ? A resposta é base 21.

Outros exemplos:

QS	+	V3	=	1LV
2C7	+	111	=	3D8
NH	+	SK	=	1L6
47D	+	199	=	605
104	+	9G	=	19K

Cujas respostas são: $QS+V3=1LV$, base=36, $2C7+111=3D8$, base=22, $NH+SK=1L6$, base=31, $47D+199=605$, base=17, $104+9G=19K$, base=35.

Observação importante: Para evitar confusão entre o zero e a letra o, esta última sempre vai ser apresentada com um acento circunflexo. (assim: ô).

Segunda observação importante: Se não ocorrer nenhuma operação de "vai um" na soma, fica impossível determinar qual a base empregada. Só é possível determinar um valor mínimo para ela (já que olhando o maior dígito que aparece na expressão, sabe-se que a base é maior do que ele). Neste caso, responder $> x$ onde $x + 1$ é o valor mínimo possível da base.

🔗 Para você fazer

- Construa a sequência de numeração de 1 até 60 para o sistema de numeração de base 31 e responda abaixo o valor das linhas
 - 42 e
 - 56
- Descubra qual base de numeração foi usada nas adições seguintes que, a propósito, estão todas corretas.
 - $GL+137=1K2$
 - $I5+10H=1IM$
 - $2FA+181=44B$
- Construa a sequência de numeração de 1 até 60 para o sistema de numeração de base 34 e responda abaixo o valor das linhas
 - 41 e
 - 46
- Descubra qual base de numeração foi usada nas adições seguintes que, a propósito, estão todas corretas.
 - $15G+25N=3BF$
 - $1H6+GK=25Q$
 - $MM+U3=1LP$

Note que se nenhuma operação de “vai um” acontece, a determinação da base pode não ser possível. Neste caso, sabe-se apenas que a base deve ser MAIOR do que um determinado valor. Se for o caso, indique este fato e escreva que a base deve ser MAIOR do que um valor, e diga que valor é este.

1.a	1.b	2.a	2.b	2.c
3.a	3.b	4.a	4.b	4.c



- 1 - /

Comandos de repetição

Nos exercícios a seguir, estão colocados pequenos trechos de código envolvendo comandos de repetição e eventualmente o comando SE. Pede-se a simulação do algoritmo (teste de mesa) para descobrir os números que são escritos ao final.

Ex. 1

```
1: Algoritmo A
2: X, S: inteiro
3: X ← 01
4: S ← 0
5: enquanto (X ≤ 15)
6:   X ← X + 3
7:   S ← S + X
8: fim{enquanto}
9: escreva (X,S)
10: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	S

Ex. 2

```
1: Algoritmo B
2: S, X : inteiro
3: X ← 24
4: S ← 0
5: repita
6:   X ← X - 6
7:   S ← S + X
8: até X < 09
9: escreva (X, S)
10: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	S

Ex. 3

```
1: algoritmo C
2: X, Y : inteiro
3: Y ← 7
4: para X ← 23 ATE 02 PASSO -4 faca
5:   Y ← Y + X
6:   se Y é par
7:     Y ← Y + 1
8:   senão
9:     Y ← Y - 2
10:  fim{se}
11: fim{para}
12: escreva (X, Y)
13: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	Y

Ex. 4

```
1: algoritmo D
2: X, Y, Z : inteiro ; X ← 2; Y ← 2; Z ← 6;
3: enquanto ((2 × (X + Y)) ≥ Z)
4:   X ← X - 3
5:   Y ← Y - 2
6: fim{enquanto}
7: se X é múltiplo de 3
8:   Y ← Y - 3
9: senão
10:  Z ← Z - 1
11: fim{se}
12: escreva(X, (Y+Z))
13: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	Y+Z

Ex. 5

```
1: Algoritmo A
2: X, S: inteiro
3: X ← 02
4: S ← 0
5: enquanto (X ≤ 17)
6:   X ← X + 5
7:   S ← S + X
8: fim{enquanto}
9: escreva (X,S)
10: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	S

Ex. 6

```
1: Algoritmo B
2: S, X : inteiro
3: X ← 03
4: S ← 0
5: repita
6:   X ← X + 4
7:   S ← S + X
8: até X ≥ 17
9: escreva (X, S)
10: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	S

Ex. 7

```
1: algoritmo C
2: X, Y : inteiro
3: Y ← 4
4: para X ← 27 ATE 07 PASSO -1 faca
5:   Y ← Y + X
6:   se Y é par
7:     Y ← Y + 1
8:   senão
9:     Y ← Y - 3
10:  fim{se}
11: fim{para}
12: escreva (X, Y)
13: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	Y

Ex. 8

```
1: algoritmo D
2: X, Y, Z : inteiro ; X ← 4; Y ← 7; Z ← 7;
3: enquanto ((2 × (X + Y)) ≥ Z)
4:   X ← X - 3
5:   Y ← Y - 1
6: fim{enquanto}
7: se X é múltiplo de 3
8:   Y ← Y - 1
9: senão
10:  Z ← Z - 2
11: fim{se}
12: escreva(X, (Y+Z))
13: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	Y+Z

Ex. 9

```
1: Algoritmo A
2: X, S: inteiro
3: X ← 13
4: S ← 0
5: enquanto (X ≥ 06)
6:   X ← X - 5
7:   S ← S + X
8: fim{enquanto}
9: escreva (X,S)
10: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	S

Ex. 10

```
1: Algoritmo B
2: S, X : inteiro
3: X ← 24
4: S ← 0
5: repita
6:   X ← X - 4
7:   S ← S + X
8: até X ≤ 08
9: escreva (X, S)
10: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	S

Ex. 11

```
1: algoritmo C
2: X, Y : inteiro
3: Y ← 5
4: para X ← 01 ATE 13 PASSO 03 faca
5:   Y ← Y + X
6:   se Y é par
7:     Y ← Y + 3
8:   senão
9:     Y ← Y - 2
10:  fim{se}
11: fim{para}
12: escreva (X, Y)
13: finalgoritmo
```

Após a execução do algoritmo acima, as duas variáveis resposta foram:

X	Y

Observação sobre repita

O *repita* neste exercício tem o padrão PASCAL, a saber: o laço se repete ATÉ a condição citada ser verdadeira. Sendo falsa, há um retorno ao verbo REPITA. Sendo verdadeira ocorre a saída do laço. Esta observação é importante porque o padrão C (e os seus aparentados: Java, C++, C#, JS, PHP, etc) faz exatamente o contrário. Sai no falso e volta no verdadeiro.

Comandos de repetição

Nos exercícios a seguir, siga a lógica do algoritmo e escreva o valor final impresso por ele.

Ex. 1

```
1: X ← 9
2: Y ← 34
3: ALFA ← 3
4: enquanto X < Y
5:   X ← X + ALFA
6:   se ((ALFA mod 2) ≠ 0)
7:     ALFA ← ALFA + 2
8:   fim{se}
9:   ALFA ← ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: Z ← 1
2: F ← 4
3: para UBA de 9 ate 17 passo 2
4:   para UCA de 1 ate 5
5:     Z ← Z + UCA
6:   fim{para}
7:   Z ← Z - 1
8: fim{para}
9: Z ← Z + F
10: F ← 5
11: escreva (Z + F)
```

Ex. 3

```
1: A ← 1
2: B ← 21
3: enquanto A ≤ B
4:   se ((B mod 2) = 0) ∨ ((A mod 3) = 0)
5:     B ← B - 1
6:   fim{se}
7:   A ← A + 3
8:   B ← B - 3
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 4

```
1: A ← 3
2: B ← 5
3: C ← 1
4: repita
5:   se B > C
6:     A ← A+B
7:   senão
8:     A ← A+C
9:   fim{se}
10: até A > (2×(B+C))
11: escreva (A + B + C)
```

Ex. 5

```
1: J ← 7
2: B ← 8
3: C ← 8
4: para J de 2 ate 14 passo 3
5:   para B de 12 ate 5 passo -1
6:     C ← J+B
7:   fim{para}
8: fim{para}
9: J ← J + 2
10: B ← B - 3
11: escreva (J + B + C)
```

Ex. 6

```
1: J ← 3
2: se J > 8
3:   repita
4:     J ← j + 1
5:   até J > 15
6: senão
```

```
7:   enquanto J >13
8:     J ← J - 2
9:   fim{enquanto}
10: fim{se}
11: escreva (J)
```

Ex. 7

```
1: X ← 10
2: Y ← 32
3: ALIN ← 4
4: enquanto X < Y
5:   X ← X + ALIN
6:   se ((ALIN mod 3) ≠ 0)
7:     ALIN ← ALIN + 2
8:   fim{se}
9:   ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 8

```
1: Z ← 2
2: F ← 3
3: para UBA de 6 ate 16 passo 1
4:   para UCA de 3 ate 5
5:     Z ← Z + UCA
6:   fim{para}
7:   Z ← Z - 1
8: fim{para}
9: Z ← Z + F
10: F ← 8
11: escreva (Z + F)
```

Ex. 9

```
1: A ← 3
2: B ← 23
3: enquanto A ≤ B
4:   se ((B mod 2) = 0) ∨ ((A mod 3) = 0)
5:     B ← B - 1
6:   fim{se}
7:   A ← A + 2
8:   B ← B - 1
9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 10

```
1: A ← 3
2: B ← 1
3: C ← 5
4: repita
5:   se B > C
6:     A ← A+B
7:   senão
8:     A ← A+C
9:   fim{se}
10: até A > (2×(B+C))
11: escreva (A + B + C)
```

Ex. 11

```
1: J ← 2
2: B ← 1
3: C ← 6
4: para J de 3 ate 15 passo 3
5:   para B de 11 ate 2 passo -4
6:     C ← J+B
7:   fim{para}
8: fim{para}
9: J ← J + 2
10: B ← B - 3
11: escreva (J + B + C)
```

Ex. 12

```
1: J ← 1
2: se J > 2
3:   repita
4:     J ← j + 1
5:   até J > 15
```

```
6: senão
7:   enquanto J >12
8:     J ← J - 7
9:   fim{enquanto}
10: fim{se}
11: escreva (J)
```

Ex. 13

```
1: X ← 9
2: Y ← 33
3: ALIN ← 3
4: enquanto X < Y
5:   X ← X + ALIN
6:   se ((ALIN mod 2) ≠ 0)
7:     ALIN ← ALIN + 2
8:   fim{se}
9:   ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 14

```
1: Z ← 3
2: F ← 1
3: para UBA de 8 ate 20 passo 4
4:   para UCA de 2 ate 5
5:     Z ← Z + UCA
6:   fim{para}
7:   Z ← Z - 2
8: fim{para}
9: Z ← Z + F
10: F ← 2
11: escreva (Z + F)
```

Ex. 15

```
1: A ← 2
2: B ← 21
3: enquanto A ≤ B
4:   se ((B mod 2) = 0) ∨ ((A mod 3) = 0)
5:     B ← B - 3
6:   fim{se}
7:   A ← A + 1
8:   B ← B - 1
9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 16

```
1: A ← 2
2: B ← 5
3: C ← 4
4: repita
5:   se B > C
6:     A ← A+B
7:   senão
8:     A ← A+C
9:   fim{se}
10: até A > (2×(B+C))
11: escreva (A + B + C)
```

Ex. 17

```
1: J ← 9
2: B ← 2
3: C ← 3
4: para J de 1 ate 13 passo 2
5:   para B de 15 ate 5 passo -6
6:     C ← J+B
7:   fim{para}
8: fim{para}
9: J ← J + 2
10: B ← B - 3
11: escreva (J + B + C)
```

Ex. 18

```
1: J ← 3
2: se J > 2
3:   repita
4:     J ← j + 3
```

```
5:   até J > 16
6: senão
7:   enquanto J >15
8:     J ← J - 7
9:   fim{enquanto}
10: fim{se}
11: escreva (J)
```

Ex. 19

```
1: X ← 9
2: Y ← 35
3: GAMA ← 4
4: enquanto X < Y
5:   X ← X + GAMA
6:   se ((GAMA mod 2) ≠ 0)
7:     GAMA ← GAMA + 2
8:   fim{se}
9:   GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 20

```
1: Z ← 2
2: F ← 2
3: para UBA de 10 ate 20 passo 2
4:   para UCA de 1 ate 5
5:     Z ← Z + UCA
6:   fim{para}
7:   Z ← Z - 1
8: fim{para}
9: Z ← Z + F
10: F ← 9
11: escreva (Z + F)
```

Ex. 21

```
1: A ← 1
2: B ← 23
3: enquanto A ≤ B
4:   se ((B mod 2) = 0) ∨ ((A mod 3) = 0)
5:     B ← B - 2
6:   fim{se}
7:   A ← A + 2
8:   B ← B - 2
9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Respostas

1.	2.	3.	SOMA →
4.	5.	6.	SOMA →
7.	8.	9.	SOMA →
10.	11.	12.	SOMA →
13.	14.	15.	SOMA →
16.	17.	18.	SOMA →
19.	20.	21.	SOMA →



Comandos de repetição

Ex. 1

```
1: A ← 6
2: enquanto A ≤ 16
3:   B ← 21
4:   enquanto B ≥ 17
5:     C ← 8
6:     se (A + B + C) > 20
7:       C ← C + 1
8:   senão
9:     C ← C - 1
10:  fim{se}
11:  A ← A + 1
12:  se B > 15
13:    A ← A + 1
14:  fim{se}
15:  B ← B - 3
16:  fim{enquanto}
17:  A ← A + 3
18: fim{enquanto}
19: escreva(A + B + C)
```

Ex. 2

```
1: A ← 13
2: B ← 2
3: C ← 13
4: repita
5:   C ← C - 3
6:   enquanto A < 36
7:     B ← B + 3
8:     se A < 18
9:       B ← B + 2
10:    senão
11:      C ← C - 1
12:    fim{se}
13:    se ((C mod 2) = 0)
14:      B ← B + 1
15:    fim{se}
16:    A ← A + 5
17:  fim{enquanto}
18: até C < 0
19: escreva (A+B+C)
```

Ex. 3

```
1: X ← 1
2: Y ← 29
3: Z ← 11
4: para I de 9 ate 15 passo 4
5:   X ← X + I
6:   para J de 23 ate 2 passo -8
7:     Y ← Y - I
8:     Z ← Z + J
9:   fim{para}
10:  se (Z × 2) > 12
11:    X ← ⌊ ( X ÷ 2)
12:  senão
13:    Y ← Y + 1
14:  fim{se}
15: fim{para}
16: se X < 11
17:   Z ← Z + 3
18: fim{se}
19: escreva (Y + Z)
```

Ex. 4

```
1: K ← 5
2: L ← 12
3: M ← 24
4: CT ← 0
5: enquanto K < 18
6:   CT ← CT + 1
7:   enquanto L > 1
8:     CT ← CT + 2
9:     enquanto M ≥ 12
10:      CT ← CT + 3
11:      M ← M - 6
12:    fim{enquanto}
13:    L ← L - 8
14:  fim{enquanto}
15:  K ← K + 4
16: fim{enquanto}
17: CT ← CT + 4
18: K ← K - 6
19: escreva (K + CT)
```

Ex. 5

```
1: A ← 7
2: se ((A mod 2) = 0)
3:   J ← 1
4:   enquanto J < 16
5:     se ((J mod 3) = 0) ∨ (J > 0)
6:       K ← 4
7:     fim{se}
8:     J ← J + 4
9:   fim{enquanto}
10: senão
11:   K ← 7
12:   repita
13:     se ((K mod 3) ≠ 0) ∨ (K > 0)
14:       J ← 5
15:     fim{se}
16:     K ← K + 4
17:     até K > 13
18:   fim{se}
19: escreva (A + K + J)
```

Ex. 6

```
1: A ← 2
2: enquanto A ≤ 17
3:   B ← 23
4:   enquanto B ≥ 12
5:     C ← 6
6:     se (A + B + C) > 20
7:       C ← C + 1
8:   senão
9:     C ← C - 1
10:  fim{se}
11:  A ← A + 1
12:  se B > 16
13:    A ← A + 1
14:  fim{se}
15:  B ← B - 3
16:  fim{enquanto}
17:  A ← A + 4
18: fim{enquanto}
19: escreva(A + B + C)
```

Ex. 7

```
1: A ← 12
2: B ← 3
3: C ← 16
4: repita
5:   C ← C - 4
6:   enquanto A < 35
7:     B ← B + 3
8:     se A < 20
9:       B ← B + 3
10:  senão
11:    C ← C - 1
12:  fim{se}
13:  se ((C mod 2) = 0)
14:    B ← B + 1
15:  fim{se}
16:  A ← A + 4
17:  fim{enquanto}
18: até C < 0
19: escreva (A+B+C)
```

Ex. 8

```
1: X ← 8
2: Y ← 26
3: Z ← 16
4: para I de 5 ate 11 passo 5
5:   X ← X + I
6:   para J de 27 ate 8 passo -6
7:     Y ← Y - I
8:     Z ← Z + J
9:   fim{para}
10:  se (Z × 2) > 19
11:    X ← ⌊ ( X ÷ 2)
12:  senão
13:    Y ← Y + 1
14:  fim{se}
15: fim{para}
16: se X < 13
17:   Z ← Z + 3
18: fim{se}
19: escreva (Y + Z)
```

Ex. 9

```
1: K ← 1
2: L ← 11
3: M ← 29
4: CT ← 0
5: enquanto K < 14
6:   CT ← CT + 1
```

```
7:   enquanto L > 3
8:     CT ← CT + 2
9:     enquanto M ≥ 11
10:      CT ← CT + 3
11:      M ← M - 7
12:    fim{enquanto}
13:    L ← L - 5
14:  fim{enquanto}
15:  K ← K + 9
16: fim{enquanto}
17: CT ← CT + 4
18: K ← K - 6
19: escreva (K + CT)
```

Ex. 10

```
1: A ← 5
2: se ((A mod 2) = 0)
3:   J ← 5
4:   enquanto J < 21
5:     se ((J mod 3) = 0) ∨ (J > 0)
6:       K ← 6
7:     fim{se}
8:     J ← J + 5
9:   fim{enquanto}
10: senão
11:   K ← 1
12:   repita
13:     se ((K mod 3) ≠ 0) ∨ (K > 0)
14:       J ← 1
15:     fim{se}
16:     K ← K + 5
17:     até K > 11
18:   fim{se}
19: escreva (A + K + J)
```

Ex. 11

```
1: A ← 2
2: enquanto A ≤ 11
3:   B ← 24
4:   enquanto B ≥ 11
5:     C ← 8
6:     se (A + B + C) > 20
7:       C ← C + 1
8:   senão
9:     C ← C - 1
10:  fim{se}
11:  A ← A + 1
12:  se B > 16
13:    A ← A + 1
14:  fim{se}
15:  B ← B - 1
16:  fim{enquanto}
17:  A ← A + 3
18: fim{enquanto}
19: escreva(A + B + C)
```

Ex. 12

```
1: A ← 16
2: B ← 8
3: C ← 15
4: repita
5:   C ← C - 3
6:   enquanto A < 31
7:     B ← B + 5
8:     se A < 17
9:       B ← B + 3
10:  senão
11:    C ← C - 1
12:  fim{se}
13:  se ((C mod 2) = 0)
14:    B ← B + 1
15:  fim{se}
16:  A ← A + 5
17:  fim{enquanto}
18: até C < 0
19: escreva (A+B+C)
```

Ex. 13

```
1: X ← 5
2: Y ← 29
3: Z ← 13
4: para I de 7 ate 11 passo 4
5:   X ← X + I
6:   para J de 23 ate 5 passo -5
7:     Y ← Y - I
8:     Z ← Z + J
9:   fim{para}
10:  se (Z × 2) > 14
11:    X ← ⌊ ( X ÷ 2)
12:  senão
13:    Y ← Y + 1
14:  fim{se}
15: fim{para}
16: se X < 16
17:   Z ← Z + 3
18: fim{se}
19: escreva (Y + Z)
```

Ex. 14

```
1: K ← 2
2: L ← 12
3: M ← 24
4: CT ← 0
5: enquanto K < 17
6:   CT ← CT + 1
7:   enquanto L > 3
8:     CT ← CT + 2
9:     enquanto M ≥ 17
10:      CT ← CT + 3
11:      M ← M - 9
12:    fim{enquanto}
13:    L ← L - 6
14:  fim{enquanto}
15:  K ← K + 6
16: fim{enquanto}
17: CT ← CT + 4
18: K ← K - 6
19: escreva (K + CT)
```

Respostas

1.	2.	3.	SOMA →
4.	5.	6.	SOMA →
7.	8.	9.	SOMA →
10.	11.	12.	SOMA →
13.	14.	xxx	SOMA →



Seguir o código

Nos exercícios a seguir, faça três coisas: a. Siga o código e ache o resultado impresso. b. Depois, traduz este código para C++ e mande executar. c. Só depois que obtiver uma coincidência de resultado, responda esta folha e entregue-a.

```
-----1-----
1: int main() {
2:   int A,B,C,D;
3:   A=6,B=3,C=1,D=5;
4:   A=A-4;
5:   if (C==5){
6:     D=C+6;
7:     if (A=3){
8:       B=C+3;
9:       C=A-8;
10:    }
11:    C=B+5;
12:    }
13:    cout<<(A+B+C+D);
14: }
```

```
-----2-----
1: int main() {
2:   int A,B,C,D;
3:   A=7,B=8,C=8,D=3;
4:   D=D+2;
5:   if ((C==6) || (B<8)){
6:     C=D+3;
7:     C=B+3;
8:   }else{
9:     B=C+7;
10:    if ((C!=7) || (C!=5)){
11:      A=D-9;
12:      A=B+9;
13:    }
14:    A=B+6;
15:    }
16:    cout<<(A+B+C+D);
17: }
```

```
-----3-----
1: int main() {
2:   int A,B,C,D;
3:   A=1,B=4,C=2,D=6;
4:   C=A+8;
5:   if (C>6){
6:     B=A+8;
7:     if (D<8){
8:       D=D+6;
9:       C=B+7;
10:    }
11:    B=A-6;
12:    }
13:    cout<<(A+B+C+D);
14: }
```

```
-----4-----
1: int main() {
2:   int A,B,C,D;
3:   A=2,B=1,C=4,D=3;
4:   C=B-5;
5:   if (D>6){
6:     D=C-4;
7:     C=B-3;
8:   }
9:   cout<<(A+B+C+D);
10: }
```

```
-----5-----
1: int main() {
2:   int A,B,C,D;
3:   A=7,B=8,C=9,D=9;
4:   C=A-9;
5:   if (D!=4){
6:     A=D+7;
7:     B=A-5;
8:   }
9:   cout<<(A+B+C+D);
10: }
```

```
-----6-----
1: int main() {
2:   int A,B,C,D;
3:   A=4,B=7,C=5,D=6;
4:   D=B+4;
5:   if (C>3){
6:     B=D+3;
7:     if (D>9){
8:       D=C-7;
9:       if ((C==8) && (B<=2)){
10:        D=C-1;
11:        if ((A<=5) || (D>=3)){
12:          D=C+1;
13:          D=C+2;
14:          }else{
15:            C=D-9;
16:            if (A==4){
17:              A=C+8;
18:              C=C-1;
19:            }
20:            A=C-4;
21:            }
22:            A=C-6;
23:            }
24:            C=B-8;
25:            }
26:            C=A+8;
27:            }else{
28:              B=A+8;
29:              if (A<2){
30:                D=D-7;
31:                A=B+7;
32:              }
33:              B=C-4;
34:              }
35:              cout<<(A+B+C+D);
36: }
```

```
-----7-----
1: int main() {
2:   int A,B,C,D;
3:   A=7,B=5,C=2,D=6;
4:   D=B-6;
5:   if (D==5){
6:     A=C+4;
7:     if (C<5){
8:       C=C+5;
9:       D=C+2;
10:    }
11:    D=B+2;
12:    }else{
13:      B=C-9;
14:      if ((B<5) || (A<=6)){
15:        C=A-1;
16:        A=B+3;
17:      }
18:      C=A-9;
19:      }
20:      cout<<(A+B+C+D);
21: }
```

```
-----8-----
1: int main() {
2:   int A,B,C,D;
3:   A=4,B=9,C=7,D=3;
4:   C=B-6;
5:   if ((B==4) || (C!=3)){
6:     C=D+2;
7:     C=A-7;
8:   }else{
9:     D=A+4;
10:    if ((C==7) && (C<8)){
11:      B=C+3;
12:      B=C-6;
13:    }
14:    C=A+4;
15:    }
16:    cout<<(A+B+C+D);
17: }
```

```
-----9-----
1: int main() {
2:   int A,B,C,D;
3:   A=7,B=4,C=4,D=6;
4:   B=B-1;
5:   if ((D!=5) && (D<=5)){
6:     D=A+1;
7:     if (A>9){
8:       D=C+8;
9:       B=C-9;
10:    }
11:    D=C+9;
12:    }else{
13:      C=A-3;
14:      if (C==2){
15:        D=B-7;
16:        if ((D>6) && (D<=6)){
17:          C=A-6;
18:          D=D+1;
19:        }else{
20:          B=D+9;
21:          if ((D==4) && (A>=8)){
22:            A=C-4;
23:            if (A!=6){
24:              D=A-6;
25:              if (C>1){
26:                B=D+2;
27:                A=A+7;
28:              }
29:              A=D-6;
30:              }else{
31:                C=B+7;
32:                if (D>=9){
33:                  C=A+4;
34:                  B=C-8;
35:                }
36:                C=B-5;
37:              }
38:              B=A+4;
39:              }
40:              A=D-6;
41:              }
42:              D=A+1;
43:              }
44:              D=D-8;
45:              }
46:              cout<<(A+B+C+D);
47: }
```

```
-----10-----
1: int main() {
2:   int A,B,C,D;
3:   A=2,B=4,C=4,D=1;
4:   C=A-3;
5:   if (B<7){
6:     D=D-4;
7:     A=B+5;
8:   }
9:   cout<<(A+B+C+D);
10: }
```

```
-----11-----
1: int main() {
2:   int A,B,C,D;
3:   A=3,B=2,C=4,D=9;
4:   C=C-1;
5:   if (C>9){
6:     C=B+3;
7:     C=B+4;
8:   }
9:   cout<<(A+B+C+D);
10: }
```

```
-----12-----
1: int main() {
2:   int A,B,C,D;
3:   A=5,B=1,C=7,D=8;
4:   C=D-1;
5:   if ((A!=1) && (A==5)){
6:     D=C-1;
7:     if (C==7){
8:       C=B-9;
9:       if ((D<6) && (C<4)){
10:        C=D-9;
11:        if ((B==4) || (B>=9)){
12:          B=A+2;
13:          D=D+5;
14:          }
15:          A=B+4;
16:          }
17:          A=C+5;
18:          }
19:          D=B-6;
20:          }
21:          cout<<(A+B+C+D);
22: }
```

```
-----13-----
1: int main() {
2:   int A,B,C,D;
3:   A=3,B=6,C=2,D=9;
4:   A=B+9;
5:   if (B>=1){
6:     B=B+3;
7:     B=C-8;
8:   }else{
9:     C=C-9;
10:    if (D>=4){
11:      A=B+6;
12:      A=A-1;
13:    }
14:    C=C+1;
15:    }
16:    cout<<(A+B+C+D);
17: }
```

```
-----14-----
1: int main() {
2:   int A,B,C,D;
3:   A=3,B=8,C=8,D=4;
4:   D=A-5;
5:   if ((D==4) || (C==1)){
6:     B=C+7;
7:     C=C+9;
8:   }else{
9:     D=A+6;
10:    if ((C==5) && (B==7)){
11:      B=C+9;
12:      B=C+1;
13:    }
14:    C=B-4;
15:    }
16:    cout<<(A+B+C+D);
17: }
```

```
-----15-----
1: int main() {
2:   int A,B,C,D;
3:   A=2,B=3,C=5,D=7;
4:   C=D-4;
5:   if ((C>=6) || (D==6)){
6:     D=B-5;
7:     D=B-3;
8:   }else{
9:     A=B-3;
10:    if ((D<=5) && (B>8)){
11:      D=B-6;
12:      if (B==2){
13:        C=B+9;
14:        D=A-7;
15:      }
16:      A=B-5;
17:      }
18:      D=A-2;
19:      }
20:      cout<<(A+B+C+D);
21: }
```



Seguir o código

Nos exercícios a seguir, faça três coisas: a. Siga o código e ache o resultado impresso. b. Depois, traduz este código para C++ ou para Python (como queira) e mande executar. c. Só depois que obtiver uma coincidência de resultado, responda esta folha e entregue-a.

```
-----1-----
1:funcao AAAA
2:A = 6
3:B = 5
4:C = 1
5:D = 4
6:D = B+ 5
7:se (C < 2)&&(A >= 2)
8:A = C+ 6
9:D = B- 3
10:senao
11:A = C- 3
12:se (A <= 1)|| (C <= 7)
13:D = A+ 7
14:B = D- 6
15:fim{se}
16:D = B+ 7
17:fim{se}
18:imprima (A+B+C+D)
19:fim{funcao}
-----2-----
1:funcao AAAA
2:A = 9
3:B = 9
4:C = 2
5:D = 7
6:C = C- 3
7:se B > 3
8:D = D+ 9
9:se (A >= 2)|| (C >=7)
10:C = C- 3
11:B = C- 7
12:fim{se}
13:D = B- 3
14:fim{se}
15:imprima (A+B+C+D)
16:fim{funcao}
-----3-----
1:funcao AAAA
2:A = 3
3:B = 7
4:C = 4
5:D = 7
6:A = A+ 1
7:se A > 9
8:D = B- 7
9:se (B < 3)&&(A <=8)
10:C = C+ 3
11:se D >= 4
12:B = C- 2
13:C = A- 2
14:fim{se}
15:D = B- 9
16:fim{se}
17:D = D+ 8
18:fim{se}
19:imprima (A+B+C+D)
20:fim{funcao}
-----4-----
1:funcao AAAA
2:A = 3
3:B = 4
4:C = 2
5:D = 1
6:A = C- 8
7:se B < 2
8:A = B- 8
9:se (D >= 3)&&(A >= 9)
10:B = C- 4
11:se (A != 1)|| (D <= 2)
12:B = A+ 4
13:se (A >= 5)&&(A < 9)
14:B = A- 1
15:A = D+ 1
16:senao
17:B = B+ 9
18:se C != 8
19:B = C- 3
20:C = B- 8
21:fim{se}

22:A = B- 2
23:fim{se}
24:D = A- 2
25:senao
26:C = A+ 3
27:se (C <= 7)|| (D == 3)
28:C = D- 2
29:C = A+ 6
30:fim{se}
31:B = A+ 2
32:fim{se}
33:D = B+ 1
34:fim{se}
35:B = D+ 1
36:senao
37:B = D- 6
38:se A != 1
39:A = B+ 9
40:D = B+ 2
41:fim{se}
42:A = B+ 8
43:fim{se}
44:imprima (A+B+C+D)
45:fim{funcao}
-----5-----
1:funcao AAAA
2:A = 5
3:B = 5
4:C = 8
5:D = 7
6:A = D- 4
7:se C <= 5
8:C = C- 6
9:B = B- 7
10:fim{se}
11:imprima (A+B+C+D)
12:fim{funcao}
-----6-----
1:funcao AAAA
2:A = 3
3:B = 3
4:C = 2
5:D = 8
6:A = B+ 1
7:se D < 5
8:B = B+ 1
9:A = C+ 8
10:fim{se}
11:imprima (A+B+C+D)
12:fim{funcao}
-----7-----
1:funcao AAAA
2:A = 4
3:B = 9
4:C = 6
5:D = 9
6:A = A+ 6
7:se (C <= 4)|| (D !=6)
8:A = D+ 8
9:D = B+ 8
10:fim{se}
11:imprima (A+B+C+D)
12:fim{funcao}
-----8-----
1:funcao AAAA
2:A = 8
3:B = 2
4:C = 7
5:D = 8
6:B = B+ 3
7:se B == 8
8:B = C- 6
9:se (D > 3)|| (A <6)
10:D = D- 7
11:se C < 6
12:B = D- 1
13:se D >= 1
14:A = D+ 7
15:A = B- 2
16:fim{se}

17:B = A+ 4
18:fim{se}
19:D = C- 2
20:fim{se}
21:C = C- 6
22:fim{se}
23:imprima (A+B+C+D)
24:fim{funcao}
-----9-----
1:funcao AAAA
2:A = 3
3:B = 7
4:C = 6
5:D = 8
6:C = A- 2
7:se (C >= 7)|| (A <8)
8:D = A- 2
9:C = C+ 8
10:fim{se}
11:imprima (A+B+C+D)
12:fim{funcao}
-----10-----
1:funcao AAAA
2:A = 1
3:B = 1
4:C = 5
5:D = 5
6:C = D+ 4
7:se (C == 8)&&(C >=2)
8:A = C- 2
9:D = D+ 5
10:senao
11:B = D+ 7
12:se (D <= 7)|| (A <=3)
13:D = C+ 2
14:se (B >= 3)&&(B !=6)
15:D = C- 5
16:se C == 8
17:C = D- 1
18:A = D- 2
19:fim{se}
20:B = C+ 2
21:fim{se}
22:A = D+ 5
23:senao
24:C = A- 2
25:se D > 3
26:C = B- 2
27:se (C >= 9)&&(A !=7)
28:C = D- 7
29:C = B- 2
30:senao
31:C = B- 6
32:se C > 7
33:C = A- 1
34:B = A+ 8
35:fim{se}
36:C = B+ 4
37:fim{se}
38:B = C+ 8
39:fim{se}
40:B = B+ 1
41:fim{se}
42:C = B+ 9
43:fim{se}
44:imprima (A+B+C+D)
45:fim{funcao}
-----11-----
1:funcao AAAA
2:A = 7
3:B = 4
4:C = 4
5:D = 4
6:B = B+ 3
7:se (A != 4)&&(B !=8)
8:D = C- 6
9:se B < 6
10:A = D+ 5
11:A = D+ 4
12:senao

13:D = A- 2
14:se A < 7
15:B = D- 4
16:se B <= 6
17:D = B- 4
18:C = C+ 7
19:fim{se}
20:C = D- 7
21:fim{se}
22:A = C- 9
23:fim{se}
24:C = D+ 6
25:senao
26:A = A+ 5
27:se B < 4
28:A = C- 9
29:A = B+ 2
30:senao
31:D = B- 7
32:se (C <= 5)|| (D >9)
33:C = B- 3
34:D = B- 8
35:fim{se}
36:A = D- 4
37:fim{se}
38:C = B+ 1
39:fim{se}
40:imprima (A+B+C+D)
41:fim{funcao}
-----12-----
1:funcao AAAA
2:A = 3
3:B = 4
4:C = 1
5:D = 1
6:C = C- 4
7:se (C <= 7)|| (A >=7)
8:A = C- 3
9:B = B- 1
10:fim{se}
11:imprima (A+B+C+D)
12:fim{funcao}
-----13-----
1:funcao AAAA
2:A = 9
3:B = 9
4:C = 1
5:D = 3
6:A = A- 6
7:se A > 7
8:D = C+ 8
9:se (D != 5)|| (D <3)
10:D = D+ 5
11:se (B != 8)&&(C >=1)
12:B = A+ 4
13:D = C- 3
14:senao
15:D = B+ 2
16:se D >= 3
17:A = C- 7
18:A = A+ 3
19:fim{se}
20:B = C- 8
21:fim{se}
22:C = A- 3
23:fim{se}
24:C = B+ 9
25:fim{se}
26:imprima (A+B+C+D)
27:fim{funcao}
```



Qual função faz o quê - C++

Nesta folha você receberá 10 enunciados, devidamente numerados de *a* a *j*, e também 10 funções numeradas de 1 a 10 e deverá estabelecer a relação entre os números e as letras, a depender de qual função obedece a qual enunciado.

As funções

- Escreva um programa na linguagem especificada que leia dois números e imprima o Mínimo Múltiplo Comum entre eles. Por exemplo, ao ler 12, 20 deve imprimir 60.
- Escreva um programa na linguagem especificada que leia 4 números e imprima a média dos 3 menores.
- Escreva um programa na linguagem especificada que leia um número e informe se ele é um quadrado ou não. Por exemplo, 100 é quadrado e 101 não é.
- Escreva um programa na linguagem especificada que leia 6 números (em ordem crescente) e imprima a quantidade de diferentes.
- Escreva um programa que leia um número e diga se ele é ou não é primo.
- Escreva um programa na linguagem especificada que leia um número e imprima a soma de seus divisores (incluindo a unidade e ele mesmo).
- Escreva um programa na linguagem especificada que leia 4 números e imprima a média dos 3 maiores.
- Escreva um programa na linguagem especificada que leia 10 números (5 pares) e imprima a média de cada par.
- Escreva um programa na linguagem especificada que leia *m* e *n* e imprima a soma dos números entre *m* e *n* (inclusive) que forem divisíveis por 7. Por exemplo, lendo 10,30, imprimirá 14+21+28=63.
- Escrever um programa na linguagem especificada que leia um número *k* e imprima a média entre o primo anterior a *k* e o posterior a *k*. Por exemplo, se ler 20, deve imprimir 21, que é (19 + 23)/2.

As respostas

-----1-----

```
int main(){
    int p,q,mi;
    cin>>p>>q;
    mi=max(p,q);
    while (1==1){
        if (((mi%p)==0)&&((mi%q)==0)){
            cout<<mi;
            return;
        }
        mi++;
    }
}
```

-----2-----

```
int main() {
    int a,b,c,d,som;
    cin>>a>>b>>c>>d;
    som=a+b+c+d;
    som=som-max(max(max(a,b),c),d);
    cout<<som/3.0;
}
```

-----3-----

```
int primo(int n){
    int dv=2;
    while (dv<n){
        if ((n%dv)==0){return 0;}
        dv++;
    }
    return 1;
}

int main(){
    int n,antes,nwork;
    cin>>n;
    nwork=n;
    while (!primo(nwork)){
        nwork--;
    }
    cout<<nwork<<endl;
    antes=nwork;
    nwork=n;
    while (!primo(nwork)){
        nwork++;
    }
    cout<<nwork<<endl;
    cout<<(nwork+antes)/2.0;
}
```

-----4-----

```
int main(){
    int a,b,c,d,e,f,qtd=1;
    cin>>a>>b>>c>>d>>e>>f;
    if (a!=b){qtd++;}
    if (b!=c){qtd++;}
    if (c!=d){qtd++;}
    if (d!=e){qtd++;}
    if (e!=f){qtd++;}
    cout<<qtd;
}
```

-----5-----

```
int main(){
    int a,b,c,d,som;
    cin>>a>>b>>c>>d;
    som=a+b+c+d;
    som=som-min(min(min(a,b),c),d);
    cout<<som/3.0;
}
```

-----6-----

```
int main(){
    int n,divs=0,cand=2;
    cin>>n;
    while (cand<n){
        if ((n%cand)==0){
            divs++;
        }
        cand++;
    }
    if (divs==0){
        cout<<"PRIMO";
    }
}
```

-----7-----

```
int main(){
    int n,i;
    cin>>n;
    i=1;
    while (i<=n){
        if ((i*i)==n){
            cout<<"sim"; return;
        }
        i++;
    }
    return;
}
```

-----8-----

```
int main(){
    int a,b,sd=0;
    cin>>a>>b;
    while (a<=b){
        if ((a%7)==0){
            sd=sd+a;
        }
        a++;
    }
}
```

```
}
    cout<<sd;
}

-----9-----
int main(){
    int n, sd=0, d=1;
    cin>>n;
    while (d<=n){
        if ((n%d)==0){
            sd=sd+d;
        }
        d++;
    }
    cout<<sd;
}
```

-----10-----

```
int main(){
    int n1,n2,n3,n4,n5,n6,n7,n8,n9,n10;
    cin>>n1>>n2>>n3>>n4>>n5>>
        n6>>n7>>n8>>n9>>n10;
    cout<<(n1+n2)/2.0<<endl;
    cout<<(n3+n4)/2.0<<endl;
    cout<<(n5+n6)/2.0<<endl;
    cout<<(n7+n8)/2.0<<endl;
    cout<<(n9+n10)/2.0<<endl;
}
```

👉 Para você fazer

Responda aqui:

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	



- 1 -

Qual função faz o quê - Python

Nesta folha você receberá 10 enunciados, devidamente numerados de *a* a *j*, e também 10 funções numeradas de 1 a 10 e deverá estabelecer a relação entre os números e as letras a depender de qual função obedece a qual enunciado.

As funções

- Escreva um programa na linguagem especificada que leia 4 números e imprima a média dos 2 menores.
- Escreva um programa na linguagem especificada que leia 6 números (em ordem crescente) e imprima a quantidade de diferentes.
- Escreva um programa na linguagem especificada que leia um número e informe se ele é um quadrado ou não. Por exemplo, 100 é quadrado e 101 não é.
- Escreva um programa na linguagem especificada que leia 3 números *a*, *b* e *c*, positivos e imprima a maior media possível entre 2 deles. $(a+b)/2$, ou $(a+c)/2$ ou $(b+c)/2$, qual for maior.
- Escreva um programa na linguagem especificada que leia 3 números: *p*, *q*, e *r* e imprima a quantidade de números iguais lidos.
- Escreva um programa na linguagem especificada que leia um número e imprima a tabuada desse número começando em 1 e terminando nele.
- Escreva um programa na linguagem especificada que leia um número e imprima a soma dos seus divisores que são múltiplos de 3.
- Escreva um programa na linguagem especificada que leia 10 números (5 pares) e imprima a média de cada par.
- Escreva um programa na linguagem especificada que leia um número *t* e imprima a soma dos quadrados $1^2 + 2^2 + 3^2 + \dots + t^2$. Por exemplo, se ler 3, $(1^2 + 2^2 + 3^2 = 14)$ deve imprimir 14.
- Suponha que você tem um computador muito pobre, que só tem as operações de contagem, adição e subtração. Escreva um programa na linguagem solicitada que leia *a* e *b* e imprima o resultado da divisão inteira de *a* por *b*, usando apenas as operações existentes. Por exemplo, ao ler 10,2 deve imprimir 5. Lendo 20,7, deve imprimir 2.

As respostas

-----1-----

```
def func(a):  
    divs=a  
    som=0  
    while (divs>=1):  
        if a%divs==0 and divs%3==0:  
            som=som+divs  
            divs=divs-1  
    print('som=',som)  
#func(6)
```

-----2-----

```
def func(a):  
    i=1  
    while i<a:  
        if i**2==a:
```

```
        print('sim')  
        i=i+1  
#func(123)  
-----3-----  
def func(a,b,c,d):  
    li=[a,b,c,d]  
    m1=sorted(li)  
    print((m1[0]+m1[1])/2)  
#func(1,10,2,22)  
-----4-----  
def func(a,b,c,d,e,f,g,h,i,j):  
    print((a+b)/2)  
    print((c+d)/2)  
    print((e+f)/2)  
    print((g+h)/2)  
    print((i+j)/2)  
#func(1,2,3,4,5,6,7,8,9,10)  
-----5-----
```

```
def func(a,b,c):  
    q=0  
    if a==b:  
        q=q+1  
    if a==c:  
        q=q+1  
    if b==c:  
        q=q+1  
    print(q)  
#func(1,1,1)  
-----6-----  
def func(a,b,c):  
    m1=(a+b)/2  
    m2=(a+c)/2  
    m3=(b+c)/2  
    if m1>=m2 and m1>=m3:  
        print(m1)  
    elif m2>=m1 and m2>=m3:  
        print(m2)  
    else:  
        print(m3)  
#func(11,4,1)
```

```
-----7-----  
def func(a):  
    j=1  
    while j<=a:  
        i=1  
        while i<=a:  
            print(j,i,j*i)  
            i=i+1  
        j=j+1  
#func(17)
```

```
-----8-----  
def func(a,b,c,d,e,f):  
    qtd=1  
    if a!=b:  
        qtd=qtd+1  
    if b!=c:  
        qtd=qtd+1  
    if c!=d:  
        qtd=qtd+1  
    if d!=e:  
        qtd=qtd+1  
    if e!=f:  
        qtd=qtd+1  
    print(qtd)  
#func(1,2,10,22,22,23)  
-----9-----
```

```
def func(a,b):  
    r=0  
    while a>=b:  
        r=r+1  
        a=a-b  
    print(r)  
#func(20,1)  
-----10-----  
def func(a):  
    som=0  
    while a>=1:  
        som=som+a**2  
        a=a-1  
    print(som)
```

#func(7)

🔗 Para você fazer

Responda aqui:

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	



- 1 -

 Para você fazer

Imprima ou escreva em uma folha os seus codigos e entregue junto com esta folha. Por favor, identifique bem qual programa atende a qual enunciado e não esqueça de se identificar colocando seu nome. Obrigado.



- 1 -

Escreva 10 programas em _____

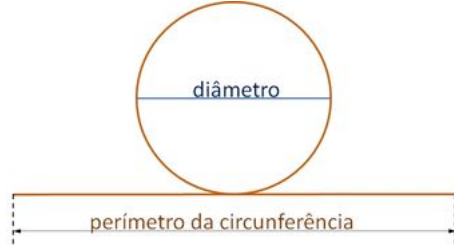
Nesta folha você receberá 10 enunciados, devidamente numerados de a a j e deverá escrever os códigos que atendem à solicitação.

Os enunciados

- Escreva um programa na linguagem especificada que leia 3 números e imprima a média dos 2 mais próximos.
- Suponha que você tem um computador muito pobre, que só tem as operações de contagem, adição e subtração. Escreva um programa na linguagem solicitada que leia a e b e imprima o resultado da divisão inteira de a por b , usando apenas as operações existentes. Por exemplo, ao ler 10,2 deve imprimir 5. Lendo 20,7, deve imprimir 2.
- Escreva um programa na linguagem especificada que leia dois números e imprima o Máximo Divisor Comum entre eles. Por exemplo, ao ler 12, 20 deve imprimir 4.
- Escreva um programa na linguagem especificada que leia 6 números (em ordem crescente) e imprima a quantidade de diferentes.
- Escreva um programa na linguagem dada que leia 2 horários de um mesmo dia (4 dígitos inteiros) no formato h_1, m_1, h_2, m_2 , sendo que o segundo horário é maior ou igual ao primeiro. O programa deve imprimir a quantidade de minutos entre eles. Por exemplo ao ler 9,10,10,0 deve responder 50.
- Escrever um programa na linguagem especificada que leia dois valores x e m e imprima a quantidade de múltiplos de 11 entre $x - m$ e $x + m$. Considere que sempre $x \geq m$.
- Escreva um programa na linguagem especificada que leia dois números inteiros a e b , e usando apenas multiplicação e divisão calcule a^b . Lembre que b pode eventualmente ser negativo, já a é sempre positivo. Por exemplo ao ler 2 e 3, deve imprimir 8 e ao ler 2, -3 a impressão deve ser 0.125. Não é permitido chamar qualquer função exponencial.
- Escreva um programa na linguagem especificada que leia 2 números a e b e calcule a^b e b^a imprimindo o maior deles, sem usar nenhuma função exponencial. Por exemplo, lendo 2,3 deve imprimir 9, Se ler 3,4 deve imprimir 81.
- Escreva um programa na linguagem especificada que leia um número t e imprima a soma dos quadrados $1^2 + 2^2 + 3^2 + \dots + t^2$. Por exemplo, se ler 3, ($1^2 + 2^2 + 3^2 = 14$) deve imprimir 14.
- Escreva um programa na linguagem especificada que leia 4 números e imprima a média dos 3 maiores.

O calculo de π

Como se sabe, π é um número irracional ou seja, não pode ser escrito como m/n , $m, n \in \mathbb{Z}$. Se pudesse, bastava descobrir m e n ou então descobrir a parte fracionária que se repete na divisão m/n . Sua definição mais simples é a divisão do comprimento de uma circunferência pelo seu diâmetro:



Para aplicações práticas (uma piscina, uma praça, ...) pode-se usar $\pi = 3.14159265$ que está perfeitamente dentro dos limites do nosso meio ambiente computacional. Por exemplo, uma praça grande com raio igual a 100 m, teria área de 31415.9265 m^2 . Se a área fosse calculada com $\pi = 3.1415$ seu valor seria de 31415 m^2 . Pesquisadores sugerem que com 39 dígitos de π é possível calcular o diâmetro de um átomo considerando o tamanho total do universo.

Mas, independentemente de sua necessidade, assiste-se a uma corrida para calcular mais e mais dígitos de π . Em 2021 o recorde era de 62,8 trilhões de dígitos. E a corrida segue em alta velocidade.

Porque ?

Pode-se especular porque os matemáticos/programadores (pessoas sérias!) continuam gastando tempo e neurônios para calcular mais dígitos de π . (a propósito, neurônios não se gastam, eles são lustrados).

Possível distribuição de dígitos em π : Muita gente boa gastou tutano estudando os dígitos em busca de algum possível padrão de formação. Até hoje não se achou nada, o que não deixa de ser uma baita resposta: π é fonte inesgotável de dígitos aleatórios. A propósito nos primeiros 1000 dígitos de π as ocorrências de 0,1,2,...,9 são 93, 116, 103, 103, 93, 97, 94, 95, 101 e 106.

Passar da matemática teórica para a matemática computacional: Uma coisa é a matemática real, que não tem falta de espaço (este é infinito) nem pressa (tem-se todo o tempo do mundo). Outra, é usar um computador para calcular coisas. Aqui, tempo e espaço são bens preciosos. A passagem de um ambiente para o outro não é trivial, e estudar este método muito bem conhecido ajuda a efetuar a transição em outras aplicações/momentos em que a ignorância é maior.

Aprender software e hardware básicos: A indústria (de TI) usa e abusa do conceito de *benchmark* que vem a ser um bloco de computação feito em diversos hardwares/plataformas/softwares para comparar desempenho e exatidão. π se ajusta como uma luva nesta demanda.

Porque ele está lá: Esta resposta pega uma carona na história de Edmond Hillary, o neozelandês que primeiro subiu ao Everest. Quando perguntado porque ele ia arriscar a vida subindo, respondeu: *porque ele está lá*. Esta, no fundo, talvez seja a melhor resposta.

Como ?

A seguir algumas possibilidades de calcular o valor de π . Existem muitas mais, além destas.

Trigonometria simples: Existem muitas (muitas!) expressões envolvendo π que podem ser usadas para fazer este cálculo. A mais simples que conheço é $\tan(45^\circ) = 1$ que é diretamente extraída da trigonometria mais pé-no-chão possível. Passando para radianos tem-se $\tan(\frac{\pi}{4}) = 1$

e $\arctan(1) = \frac{\pi}{4}$. Agora nos socorremos da fórmula de Leibnitz para o arco tangente que é $\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} + \dots (-1 \leq x \leq 1)$. E juntando as duas expressões nos dá: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \dots$. Eis um Python simples para isso

```
import math
def pi_gregory(n):
    result = 0
    divisor = 1
    sign = 1
    for i in range(n):
        result = result + (sign / divisor)
        divisor = divisor + 2
        sign = -sign # troca o sinal
    return 4 * result

for p in range(1,8):
    n = 10**p
    res = pi_gregory(n)
    err = res - math.pi
    print("%8d = %.10f erro %.10f" % (n,res,err))

    10 = 3.0418396189 erro -0.0997530347
    100 = 3.1315929036 erro -0.0099997500
    1000 = 3.1405926538 erro -0.0009999997
    10000 = 3.1414926536 erro -0.0001000000
    100000 = 3.1415826536 erro -0.0000100000
    1000000 = 3.1415916536 erro -0.0000010000
    10000000 = 3.1415925536 erro -0.0000001000
```

Este algoritmo implica em 10 vezes mais iterações que o anterior para achar o próximo dígito de π . Pense nisso para 1 trilhão de dígitos... Sem chance, precisamos algo melhor.

Arquimedes: Em 200 AEC, Arquimedes pensou em um círculo de raio 1 e inscreveu nele um polígono de n lados. À medida em que n cresce, o perímetro do polígono se aproxima da medida da circunferência do círculo. Deixando de lado um pouco de geometria, até por falta de espaço aqui, mas que pode ser vista em

<https://www.craig-wood.com/nick/articles/pi-archimedes/> tem-se

```
def pi_archimedes(n):
    poly = 2.0
    sides = 4
    for i in range(n):
        poly = 2 - 2 * math.sqrt(1 - poly / 4)
        sides = sides * 2
    return sides * math.sqrt(poly) / 2

for n in range(16):
    res = pi_archimedes(n)
    err = res - math.pi
    print("%8d = %.10f err %.10f" % (n,res,err))

    0 = 2.8284271247 err -0.3131655288
    2 = 3.1214451523 err -0.0201475013
    4 = 3.1403311570 err -0.0012614966
    6 = 3.1415138011 err -0.0000788524
    8 = 3.1415877253 err -0.0000049283
    10 = 3.1415923456 err -0.0000003080
    12 = 3.1415926335 err -0.0000000201
    13 = 3.1415926548 err 0.0000000012
    14 = 3.1415926453 err -0.0000000083
    15 = 3.1415926074 err -0.0000000462
==== algumas linhas omitidos por economia ====
```

Calculou-se aqui π com 8 casas em 13 iterações. Mas, a partir daqui o erro começa a aumentar. Isso ocorre porque estamos presos ao limite de precisão do ponto flutuante de Python (mais ou menos 17 dígitos) e os erros começam a se acumular.

Algoritmo de Chudnovsky Usa uma fórmula derivada dos trabalhos de Ramanujan, um indiano que viveu no começo do século 20 e deixou milhares de pérolas em cadernos manuscritos. Morreu jovem, com 32 anos em 1920. Em 1997 surgiu o *The Ramanujan Journal* publicação seriada das mais importantes no atual panorama da matemática para entre outras coisas, estudar os cerca de 3900 resultados de Ramanujan.

Há uma anedota conhecida envolvendo Ramanujan que foi mostrada no filme *The Man Who Knew Infinity*, refere-se ao fato de que quando Hardy foi visitar Ramanujan no hospital, disse *Eu havia viajado no táxi número 1729 e observei que o número me parecia um tanto enfadonho e que esperava que não fosse um presságio desfavorável.*

"Não", respondeu ele, "é um número muito interessante; é o menor número que pode ser expresso como a soma de dois cubos de duas maneiras diferentes." São elas $1729 = 1^3 + 12^3 = 9^3 + 10^3$.

Uma das pérolas de Ramanujan, que foi desenvolvido pelos irmãos Chudnovsky por volta de 1988 é

$$\frac{1}{\pi} = 12 \sum_{k=0}^{\infty} \frac{(-1)^k (6k)! (13591409 + 545140134k)}{(3k)! (k!)^3 640320^{3k+3/2}}$$

que devidamente transplantado para python é

```
from decimal import *
def fatorial(n):
    if n<1:
        return 1
    else:
        return n * fatorial(n-1)

def chudnovsky(n):
    getcontext().prec = n
    pi = Decimal(0)
    k = 0
    while k < n:
        pi = pi + (Decimal(-1)**k) * \
            (Decimal(fatorial(6*k)) \
            / ((fatorial(k)**3) * (fatorial(3*k))) * \
            (13591409+545140134*k) / (640320**(3*k)))
        k = k + 1
    pi = pi * Decimal(10005).sqrt() / 4270934400
    pi = pi**(-1)
    return pi
print(chudnovsky(60))
```

3.14159265358979323846264338327
 950288419716939937510582097492

Algoritmo BBP: A Fórmula BBP é uma fórmula de cálculo do número π , descoberta em 1995 por David Harold Bailey, em colaboração com Peter Borwein e Simon Plouffe. Essa fórmula permite calcular facilmente a n ésima decimal binária ou hexadecimal de π sem ter que calcular as decimais precedentes. Usa a fórmula

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

Em python

```
from decimal import Decimal
from decimal import getcontext
def pi(prec):
    getcontext().prec=prec
    aa=0
    for k in range(prec):
        aa=aa+(1/Decimal(16)**k *
            (Decimal(4)/(8*k+1) -
            Decimal(2)/(8*k+4) -
            Decimal(1)/(8*k+5) -
            Decimal(1)/(8*k+6)))
    return aa
print(pi(150))
```

3.14159265358979323846264338327950288419716939
 9375105820974944592307816406286208998628034825
 3421170679821480865132823066470938446095505822
 3172535940812

👉 Para você fazer

Utilize o método que quiser e ache os dígitos de π correspondendo às seguintes posições (O 3 inicial é a posição 1, o primeiro 1 é a segunda posição, o 4 é a terceira e assim por diante...)

1. 244
2. 394
3. 624
4. 888

1	2	3	4
1			



Condicionalis em C++

Nos exercícios a seguir, você deve simular o compilador C++ e descobrir qual valor vai ser impresso ao final. Considere todas as variáveis como sendo do tipo **int**. Todos os códigos estao sintaticamente corretos. A sugestão é que **depois** de ter calculado o valor na ponta do lápis, você confira o resultado com um computador real.

🔗 Para você fazer

Exercício 1

```
int main() {
    int A,B,C,D,F,G;
    A=2,B=9,C=2,D=8,F=8,G=3;
    if (G!=6){
        G=A-G;
        G=C+G;
        G=G*D;
    }
    cout<<F+C+A-B;
}
```

Informe no quadro próprio o valor que foi impresso.

Exercício 2

```
int main() {
    int A,B,C,D,F,G;
    A=8,B=2,C=7,D=6,F=6,G=6;
    if (F>=2){
        C=C+G;
        C=C-F;
    }else{
        C=A+B;
    }
    cout<<A+F+C-B;
}
```

Exercício 3

```
int main() {
    int A,B,C,D,F,G;
    A=4,B=3,C=9,D=5,F=5,G=8;
    if ((B<=4) || (!(D!=2))){
        G=D+D;
        A=B-C;
    }else{
        B=G+B;
    }
    cout<<A+G+F-B;
}
```

Exercício 4

```
int main() {
    int A,B,C,D,F,G;
    A=9,B=8,C=7,D=4,F=5,G=6;
    if ((F>=3) && (A>=8)){
        G=B*G;
    }else{
        B=B-B;
        A=B*D;
    }
    cout<<A+F+G-C;
}
```

Exercício 5

```
int main() {
    int A,B,C,D,F,G;
    A=8,B=9,C=6,D=8,F=7,G=7;
    if ((F<=8) && (F!=5)){
        A=F*F;
        if (A<9){
            C=A-G;
        }else{
            D=A+A;
        }
    }else{
        D=F-C;
        C=C+B;
    }
    cout<<A+B+C-D;
}
```

Exercício 6

```
int main() {
    int A,B,C,D,F,G;
    A=4,B=3,C=8,D=4,F=4,G=4;
    if ((D<2) && (B<4)){
        A=D*A;
        if (G<4){
            B=F*A;
        }else{
            F=D*A;
        }
    }else{
        if (C<3){
            G=A+C;
        }
        C=A*G;
    }
    cout<<B+D+F-A;
}
```

Exercício 7

```
int main() {
    int A,B,C,D,F,G;
    A=8,B=3,C=9,D=9,F=5,G=7;
    if ((D!=6) && (D==2)){
        D=G+A;
        if (C<=8){
            G=B+G;
        }else{
            if ((D>6) && (!(F!=5))){
                A=G-D;
            }
        }
    }else{
        if (B!=2){
            D=G-B;
        }
        F=B+F;
    }
    cout<<A+D+G+F;
}
```

Exercício 8

```
int main() {
    int A,B,C,D,F,G;
    A=9,B=5,C=8,D=6,F=4,G=6;
    if ((G<=9) || (F<=6)){
        G=D*F;
        if (G<=3){
            F=G*C;
        }else{
            if ((A!=9) || (C==4)){
                B=A-D;
            }
        }
    }else{
        if (G>6){
            B=C+F;
        }
        D=D*G;
    }
    cout<<A+B+D-C;
}
```

Exercício 9

```
int main() {
    int A,B,C,D,F,G;
    A=4,B=9,C=4,D=7,F=8,G=8;
    if ((D==3) && (B>=8)){
        F=A+B;
        if (D<6){
            F=C*D;
        }else{
            B=G-B;
        }
    }else{
        D=A*D;
        A=F-G;
    }
    cout<<D+A+F-G;
}
```

Exercício 10

```
int main() {
    int A,B,C,D,F,G;
    A=7,B=5,C=2,D=5,F=8,G=2;
    if ((G<5) || (G>9)){
        F=D+G;
        if (A!=7){
            A=C*F;
        }
    }
}
```

```
        }else{
            B=F-C;
        }
    }else{
        if (F==2){
            F=B+C;
        }
        F=D+F;
    }
    cout<<A+D+B-G;
}
```

Exercício 11

```
int main() {
    int A,B,C,D,F,G;
    A=4,B=2,C=8,D=9,F=5,G=3;
    if ((!(D>=8)) || (A>6)){
        G=G-B;
        if (B==2){
            G=C-F;
        }else{
            if ((B<4) && (C!=6)){
                D=G+F;
            }
        }
    }else{
        if (G>7){
            C=A*B;
        }
        F=D+G;
    }
    cout<<D+A+B+C;
}
```

Exercício 12

```
int main() {
    int A,B,C,D,F,G;
    A=5,B=5,C=8,D=4,F=2,G=3;
    if ((F<=8) || (A!=7)){
        A=A*A;
        if (C<=3){
            G=B*D;
        }else{
            if ((C>=3) || (G<3)){
                C=D-D;
            }
        }
    }else{
        if (A>5){
            C=G*F;
        }
        F=C*F;
    }
    cout<<G+A+B-D;
}
```

Responda aqui:

1	2	3	4
5	6	7	8
9	10	11	12



Condicionalis em C++ - II

Nos exercícios a seguir, você deve simular o compilador C++ e descobrir qual valor vai ser impresso ao final. Considere todas as variáveis como sendo do tipo **int**. Todos os códigos estao sintaticamente corretos. A sugestão é que **depois** de ter calculado o valor na ponta do lápis, você confira o resultado com um computador real.

👉 Para você fazer

Exercício 1

```
int main() {
    int A,B,C,D,F,G;
    A=3,B=4,C=4,D=7,F=9,G=2;
    if(C!=2){
        D=C-C;
        if((G>=2)|| (A>=2)){
            G=F-A;
        }
        G=D*G;
    }else{
        C=A+A;
        if((!(G!=5))||(D==7)){
            C=A+A;
        }else{
            if(F<8){
                D=B+A;
            }else{
                D=A+D;
            }
            C=A*G;
        }
        B=G*C;
    }
    C=B-D;
    cout<<A+G+C-F;
}
```

Informe no quadro próprio o valor que foi impresso.

Exercício 2

```
int main() {
    int A,B,C,D,F,G;
    A=3,B=9,C=3,D=9,F=6,G=3;
    if(A>3){
        if((!(C>=6))||(B==3)){
            C=D-A;
        }
        if(B<=8){
            G=G+G;
        }
        C=G*A;
    }else{
        F=A+F;
        if((F==8)&&(B<7)){
            A=C+F;
            if(F>5){
                C=G*F;
            }
        }else{
            if((C>2)&&(F>9)){
                C=F-F;
            }else{
                G=F-B;
            }
            D=C-A;
        }
        B=C-B;
    }
    cout<<D+C+G-A;
}
```

Exercício 3

```
int main() {
    int A,B,C,D,F,G;
    A=8,B=7,C=6,D=7,F=7,G=2;
    if(A==5){
        C=B*D;
    }
    if((D<=4)&&(D<=4)){
        if(F>7){
            C=C*A;
        }
    }
    F=G-G;
    if((C>=4)&&(G!=3)){
        D=C*A;
        if(A>5){
            G=G+F;
        }else{
            D=G-C;
        }
    }else{
        if((C!=8)&&(D>5)){
            C=G-C;
        }else{
            F=F-A;
        }
        C=G+C;
    }
    D=A+C;
    cout<<F+B+A-C;
}
```

Exercício 4

```
int main() {
    int A,B,C,D,F,G;
    A=9,B=9,C=9,D=9,F=7,G=6;
    if(C<9){
        A=C-C;
    }
    if((B!=2)|| (C==6)){
        if(B<6){
            C=F+G;
        }else{
            if((B<2)|| (C>=7)){
                F=C-F;
            }else{
                if(A!=4){
                    G=D+D;
                }
            }
            G=C+B;
        }
        F=C+G;
    }
    A=C*F;
    if((B==7)&&(G==9)){
        C=B+G;
        if(G<=3){
            G=C-D;
        }else{
            C=C*F;
        }
    }else{
        if((B>=6)|| (G>=9)){
            C=C*G;
        }else{
            D=D-A;
        }
        F=A*B;
    }
    B=B+G;
    cout<<C+B+A-F;
}
```

Exercício 5

```
int main() {
    int A,B,C,D,F,G;
    A=7,B=6,C=7,D=3,F=5,G=9;
    if(D!=4){
        if((F!=6)&&(C==4)){
            if(F>3){
                if((B<3)&&(D<5)){
                    G=G+D;
                }else{
                    G=C+B;
                }
            }
            C=B-D;
        }else{
            C=G*F;
        }
        D=G+G;
    }else{
```

```
        D=G-D;
    }
    D=D+B;
}else{
    if(F<2){
        if((F==4)|| (B<=3)){
            if(A!=4){
                B=D-F;
            }else{
                G=F+A;
            }
        }else{
            A=F-C;
        }
    }
    C=A*F;
}
B=D+A;
cout<<G+F+B-C;
}
```

Exercício 6

```
int main() {
    int A,B,C,D,F,G;
    A=7,B=9,C=8,D=3,F=7,G=3;
    if(F>6){
        D=G-B;
        if((C>=2)|| (G<=7)){
            G=A-G;
        }
        B=B*G;
    }else{
        G=C-D;
        if((F<=5)|| (G<=7)){
            G=C-D;
        }else{
            if(B>9){
                C=C-A;
            }else{
                D=D-B;
            }
            B=C*A;
        }
        D=B-D;
    }
    A=G-G;
    cout<<B+F+D-A;
}
```

Responda aqui:

1	2	3
4	5	6


- 1 - /

Condicionalis em Python

Nos exerc cios a seguir, voc  deve simular o inter-
pretador Python e descobrir qual valor vai ser im-
presso ao final. Considere todas as vari veis como
sendo do tipo **inteiro**. Todos os c digos est o sin-
taticamente corretos. A sugest o   que **depois** de
ter calculado o valor na ponta do l pis, voc  confira
o resultado com um computador real.

Para voc  fazer
Exerc cio 1

```
def AA():  
    A=4;B=5;C=8;D=6;F=6;G=6  
    if F!=2:  
        C=A+F  
        F=D+B  
        C=C-C  
    print(A+G+D-C)  
AA()
```

Exerc cio 2

```
def AA():  
    A=5;B=3;C=4;D=9;F=3;G=9  
    if F>=5:  
        G=G+C  
        G=G-C  
    else:  
        G=F*G  
    print(C+F+D-A)  
AA()
```

Exerc cio 3

```
def AA():  
    A=4;B=9;C=9;D=5;F=4;G=3  
    if (D>7)or(not(C>=7)):  
        G=B*G  
        F=C+G  
    else:  
        G=B+C  
    print(G+C+D+F)  
AA()
```

Exerc cio 4

```
def AA():  
    A=4;B=6;C=4;D=2;F=8;G=9  
    if (F<3)or(not(F>2)):  
        G=C-B  
    else:  
        A=A+G  
        G=F+F  
    print(F+B+C-G)  
AA()
```

Exerc cio 5

```
def AA():  
    A=5;B=2;C=8;D=3;F=4;G=4  
    if (B<4)or(A<3):  
        D=B*D  
        if C>=7:  
            B=G*G  
        else:  
            C=C+A  
    else:  
        B=F*G  
        G=G*F  
    print(C+D+B-G)  
AA()
```

Exerc cio 6

```
def AA():  
    A=9;B=8;C=8;D=5;F=2;G=7  
    if (not(D==8))or(not(G>=2)):  
        D=C-A  
        if G!=6:  
            F=G*B  
        else:  
            B=F-D  
    else:  
        if G>2:  
            C=F+A  
            D=G*G  
    print(F+C+D+B)  
AA()
```

Exerc cio 7

```
def AA():  
    A=9;B=9;C=6;D=6;F=9;G=5  
    if (A!=4)or(C!=7):  
        B=D*B  
        if D<5:  
            D=B*B  
        else:  
            if (not(G>=4))or(C==8):  
                C=G+F  
    else:  
        if A==4:  
            C=G*D  
            F=F*F  
    print(B+C+A-F)  
AA()
```

Exerc cio 8

```
def AA():  
    A=9;B=4;C=3;D=2;F=6;G=6  
    if (D>8)or(not(A>=6)):  
        B=G*B  
        if F<=5:  
            F=A+D  
        else:  
            if (A>3)and(F>=5):  
                D=F-D  
    else:  
        if C==2:  
            F=B+A  
            G=G*A  
    print(A+F+B+D)  
AA()
```

Exerc cio 9

```
def AA():  
    A=7;B=9;C=5;D=5;F=9;G=2  
    if B<9:  
        F=F*F  
        F=C*C  
        B=F+B  
    print(G+B+C-D)  
AA()
```

Exerc cio 10

```
def AA():  
    A=2;B=3;C=9;D=7;F=8;G=7  
    if D==6:  
        C=C+A  
        C=A+G  
    else:  
        C=A+F  
    print(B+F+A-D)  
AA()
```

Exerc cio 11

```
def AA():  
    A=3;B=9;C=7;D=8;F=3;G=4  
    if (F<2)or(C<5):  
        G=C*G  
        F=F*G  
    else:  
        B=A*B  
    print(D+G+F+A)  
AA()
```

Exerc cio 12

```
def AA():  
    A=2;B=9;C=9;D=7;F=3;G=7  
    if (G!=2)or(D!=6):  
        A=D+G  
    else:  
        G=A*G  
        A=C*G  
    print(F+B+D-G)  
AA()
```

Exerc cio 13

```
def AA():  
    A=5;B=9;C=8;D=6;F=2;G=6  
    if (F!=5)or(not(B<3)):  
        G=C+B  
        if G==3:  
            B=F+C  
        else:  
            G=C+A  
    else:  
        G=A+G  
        F=B+C  
    print(G+C+A-B)  
AA()
```

Exerc cio 14

```
def AA():  
    A=8;B=4;C=4;D=9;F=8;G=8  
    if (C<7)or(F>2):  
        F=F+B  
        if D>=2:  
            G=A-F  
        else:  
            D=A*F  
    else:  
        if A!=8:  
            G=B*B  
            G=B+D  
    print(A+C+B-G)  
AA()
```

Exerc cio 15

```
def AA():  
    A=6;B=7;C=8;D=5;F=8;G=7  
    if (B==7)and(B>=7):  
        D=D*B  
        if B>5:  
            A=A+D  
        else:  
            if (B<=9)and(B<2):  
                D=A+F  
    else:  
        if D<5:  
            A=C*B  
            F=A*C  
    print(D+F+C-A)  
AA()
```

Exerc cio 16

```
def AA():  
    A=4;B=5;C=6;D=3;F=5;G=5  
    if (not(C<8))and(C<3):  
        G=G-B  
        if D>=3:  
            D=D*C  
        else:  
            if (F<8)or(D==4):  
                G=A+D  
    else:  
        if D>=3:  
            F=G-D  
            A=B+B  
    print(B+D+C+A)  
AA()
```

Responda aqui:
Some os 4 valores de cada coluna na vertical. O
gabarito contempla apenas a soma e n o os 4 valo-
res originais.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
Σ	Σ	Σ	Σ

Condicionalis em Python - II

Nos exercícios a seguir, você deve simular o interpretador Python e descobrir qual valor vai ser impresso ao final. Considere todas as variáveis como sendo do tipo **inteiro**. Todos os códigos estão sintaticamente corretos. A sugestão é que **depois** de ter calculado o valor na ponta do lápis, você confira o resultado com um computador real.

Um exemplo

```
def BB():
    A=4;B=1;C=5;D=6
    if A<D and C!=4:
        if A==C-1:
            A=A+1
            D=D+2
        else:
            C=C*2
    else:
        A=A+3
    C=C+2
    print(A+B-C)
BB()
```

Deu como resposta -1.

Mais um

```
def CC():
    C=1;B=2
    if not (C!=1) and B==2:
        B=B*2
    else:
        C=C+10
    C=C+2
    print(C+B)
CC()
```

Deu como resposta 7.

👉 Para você fazer

Exercício 1

```
def AA():
    A=5;B=3;C=3;D=4;F=5;G=3
    if A>4:
        G=C+C
        if (F<=2) or (D>=2):
            D=C*B
        A=A*A
    else:
        C=G-C
        if (G<9) and (G==6):
            C=G-C
        else:
            if D>7:
                G=C-D
            else:
                B=F-G
                F=A+D
            D=G+B
    print(A+G+C-B)
AA()
```

Exercício 2

```
def AA():
    A=5;B=8;C=5;D=4;F=3;G=9
    if D<=8:
        if (C!=9) or (not (C>2)):
            G=F*G
        if A==5:
            C=D*B
        B=G+B
    else:
        D=D*B
        if (C<=4) and (D>=3):
            G=D*D
            if D>=7:
                C=F*A
        else:
            if (C>=5) and (A<3):
                A=A-F
            else:
```

```
            D=F-A
            G=A-D
            G=F*C
            print(G+C+F+D)
AA()
```

Exercício 3

```
def AA():
    A=3;B=6;C=4;D=6;F=6;G=7
    if G==8:
        C=C-D
    if (A<=2) or (D>6):
        if C>=5:
            F=C+F
        A=C*G
    if (D!=8) or (G!=7):
        D=F*F
        if G<=5:
            F=B-G
        else:
            A=F-B
    else:
        if (G==4) and (B>8):
            C=G*B
        else:
            F=F*C
            G=D-D
            F=A+D
            print(A+D+F-G)
AA()
```

Exercício 4

```
def AA():
    A=2;B=4;C=5;D=8;F=8;G=3
    if F==4:
        B=D-A
    if (D>=9) and (B==3):
        if A<=7:
            C=F+B
        else:
            if (B>=3) and (A>4):
                F=F-G
            else:
                if D<=3:
                    G=F*A
                D=A+F
                F=C*G
            F=C*G
    if (B>8) or (G!=7):
        F=G*C
    if F>=5:
        C=C+B
    else:
        F=F-D
    else:
        if (not (A==2)) or (C<=6):
            G=D+B
        else:
            B=C*D
            F=G*B
            B=B*A
            print(G+D+A-F)
AA()
```

Exercício 5

```
def AA():
    A=8;B=6;C=9;D=5;F=2;G=2
    if D>4:
        if (F>3) and (G<4):
            if F==8:
                if (G>6) and (B<7):
                    F=C*A
                else:
                    F=B+C
                    G=A+B
            else:
                A=C+G
                D=G*A
        else:
            F=C-D
        F=G-A
    else:
        if F>9:
            if (B>9) and (not (C<=7)):
                if G>9:
                    G=A-C
                else:
                    G=D-B
            else:
                B=G-G
                F=G*A
                C=D*D
```

```
            print(C+G+D-A)
AA()

def AA():
    A=5;B=4;C=3;D=9;F=5;G=8
    if F==3:
        G=F*D
        if (G>4) or (not (C!=5)):
            A=B-B
            C=B+D
    else:
        A=G+G
        if (D==7) or (D>3):
            A=G+G
        else:
            if B==2:
                C=F-D
            else:
                C=F*A
                G=C-G
            G=C*G
            F=B+A
            print(C+B+F-G)
AA()
```

Exercício 7

```
def AA():
    A=9;B=6;C=5;D=3;F=5;G=5
    if G>4:
        if (B<=3) and (D<=6):
            C=B*D
        if C<=7:
            F=C+G
            A=F+F
        else:
            G=A+C
            if (B>=9) and (D==9):
                B=D-B
                if A!=5:
                    C=F-G
            else:
                if (C!=3) and (A>3):
                    C=B-D
                else:
                    F=B+F
                    G=D+C
                    C=A+F
            print(B+F+C-G)
AA()
```

Exercício 8

```
def AA():
    A=6;B=9;C=4;D=4;F=8;G=3
    if B!=3:
        B=B*G
    if (A>=7) or (F>6):
        if A<=3:
            A=F*G
        A=B+G
    if (F<6) and (F==2):
        D=G+B
        if C==9:
            B=D+A
        else:
            B=G+B
    else:
        if (not (F!=8)) and (F<3):
            C=D+C
        else:
            A=D+A
            B=A-B
            B=F+F
            print(B+D+F-A)
AA()
```

Responda aqui:

1	2	3	4
5	6	7	8

Exercícios de repetição Nos exercícios a seguir, é dado um trecho de código em pseudo-código. Logo abaixo dele, 3 conjuntos completos de dados de entrada MAIS os dados de saída são apresentados. Você deve estudar o código, os dados de entrada e de saída e deve decidir se aquele conjunto é possível (respondendo 1) ou é impossível (respondendo 0). Ao final, os resultados numéricos (1 ou 0) devem ser agrupados formando um número binário, que será a resposta.

Acompanhe no exemplo: Seja o trecho de código

```
1: Z ← A
2: X ← B
3: enquanto X ≤ C
4:   Z ← Z + D
5:   X ← X + E
6: fim{enquanto}
7: escreva (Z)
```

Se os dados de entrada forem A=3 B=4 C=13 D=3 E=2 e o dado de saída for Z=18, devidamente aplicados ao algoritmo darão um resultado correto. Deve portanto ser assinalado com “1”. O segundo conjunto, A=3 B=3 C=14 D=2 E=3 Z=13, dá um resultado errado, pois se aplicados no algoritmo gerarão como saída 11 e não 13. Este resultado, portanto, é zero. O terceiro A=5 B=4 C=15 D=3 E=3 Z=17, também está correto sendo assinalado com “1”. Para responder deve-se converter o binário 1 0 1 em decimal o que dá “5”, que é a resposta correta. Note que a grade de correção ao final da folha, dá ajuda para fazer a transformação binário → decimal. Outro detalhe importante é que o esforço computacional é sempre pequeno, o que garante ser simples resolver este exercício com lápis e papel. As variáveis cujos nomes tem as letras iniciais do alfabeto, devem ser substituídas no código pelos valores informados, EM TODOS OS LOCAIS DO CODIGO (às vezes é mais de um local).

Exercício 1

```
1: Z ← A
2: X ← B
3: enquanto X ≤ C
4:   Z ← Z + D
5:   X ← X + E
6: fim{enquanto}
7: escreva (Z)
```

Para este exercício os valores são:

A	B	C	D	E	Z	1/0
3	3	14	2	2	13	
3	5	17	3	3	21	
5	4	15	2	2	17	

Exercício 2

```
1: K ← A
2: L ← B
3: M ← C
4: enquanto K + L > D
5:   M ← M - E
6:   K ← K + F
7:   L ← L - G
8: fim{enquanto}
9: escreva (K + M)
```

Para este exercício os valores são:

A	B	C	D	E	F	G	K+M	1/0
3	4	5	21	3	3	2	10	
5	3	4	24	3	2	3	10	
4	5	3	22	3	3	2	8	

Exercício 3

```
1: T ← A
2: Y ← B
3: repita
4:   R ← C + T + Y
5:   Y ← Y + D
6: até R > E
7: escreva (Y)
```

Para este exercício os valores são:

A	B	C	D	E	Y	1/0
4	3	1	6	13	15	
5	5	3	4	13	14	
4	4	1	5	28	28	

Exercício 4

```
1: F ← A
2: para I ← B até C passo D
3:   F ← F + I + E
4: fim{para}
5: escreva (F)
```

Para este exercício os valores são:

A	B	C	D	E	F	1/0
4	4	16	5	8	55	
4	4	13	5	11	38	
4	3	16	3	3	64	

Exercício 5

```
1: Q ← A
2: R ← B
3: S ← C
4: enquanto (Q + S) ≤ R
5:   Q ← Q + D
6:   S ← S - E
7: fim{enquanto}
8: escreva (21+Q-S)
```

Para este exercício os valores são:

A	B	C	D	E	21+Q-S	1/0
3	4	5	7	3	18	
4	4	6	4	2	19	
2	4	7	6	2	16	

Exercício 6

```
1: P ← A
2: J ← B
3: J ← J + A + 25
4: enquanto J > C
5:   P ← P + D
6:   J ← J - E
7: fim{enquanto}
8: escreva (P)
```

Para este exercício os valores são:

A	B	C	D	E	P	1/0
2	2	15	4	3	19	
2	2	21	3	2	14	
4	1	22	5	1	44	

Exercício 7

```
1: W ← A
2: repita
3:   W ← W + B
4: até W > C
5: escreva (W+C)
```

Para este exercício os valores são:

A	B	C	W+C	1/0
3	5	19	40	
3	3	26	53	
4	3	27	55	

Exercício 8

```
1: ALFA ← A
2: BETA ← B
3: enquanto ((ALFA+BETA) mod 11) ≠ 0
4:   ALFA ← ALFA + 2
5:   BETA ← BETA - 1
6: fim{enquanto}
7: escreva (10+ALFA-BETA)
```

Para este exercício os valores são:

A	B	10+ALFA-BETA	1/0
8	12	15	
8	10	20	
11	4	40	

Exercício 9

```
1: T ← A
2: Y ← B
3: repita
4:   R ← C + T + Y
5:   Y ← Y + D
6: até R > E
7: escreva (Y)
```

Para este exercício os valores são:

A	B	C	D	E	Y	1/0
3	5	1	4	19	22	
4	5	3	6	27	29	
4	5	4	5	12	8	

Exercício 10

```
1: F ← A
2: para I ← B até C passo D
3:   F ← F + I + E
4: fim{para}
5: escreva (F)
```

Para este exercício os valores são:

A	B	C	D	E	F	1/0
4	4	16	4	15	102	
3	3	15	6	13	70	
3	3	12	6	5	24	

Exercício 11

```
1: Q ← A
2: R ← B
3: S ← C
4: enquanto (Q + S) ≤ R
5:   Q ← Q + D
6:   S ← S - E
7: fim{enquanto}
8: escreva (21+Q-S)
```

Para este exercício os valores são:

A	B	C	D	E	21+Q-S	1/0
4	3	6	7	1	19	
4	2	4	7	1	21	
2	4	5	7	3	16	

Exercício 12

```
1: P ← A
2: J ← B
3: J ← J + A + 25
4: enquanto J > C
5:   P ← P + D
6:   J ← J - E
7: fim{enquanto}
8: escreva (P)
```

Para este exercício os valores são:

A	B	C	D	E	P	1/0
4	1	16	3	3	16	
2	1	21	5	3	17	
4	2	15	5	2	45	

📝 Para você responder

1.1	1.2	1.3	2.1	2.2	2.3	questão
2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	soma 1.

3.1	3.2	3.3	4.1	4.2	4.3	questão
2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	soma 2.

5.1	5.2	5.3	6.1	6.2	6.3	questão
2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	soma 3.

7.1	7.2	7.3	8.1	8.2	8.3	questão
2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	soma 4.

9.1	9.2	9.3	10.1	10.2	10.3	questão
2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	soma 5.

11.1	11.2	11.3	12.1	12.2	12.3	questão
2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	soma 6.



Repetição em C++

Nos exercícios a seguir, você deve simular o compilador C++ e descobrir qual valor vai ser impresso ao final. Considere todas as variáveis como sendo do tipo **int**. Todos os códigos estão sintaticamente corretos. A sugestão é que **depois** de ter calculado o valor na ponta do lápis, você confira o resultado com um computador real.

Exemplos

Seja o programa

```
int main() {
    int A,B,C,D;
    A=5,B=2,C=7;
    while(B<15){
        C=A+5;
        A=A+4;
        B=B+1;
    }
    cout<<C+A+B;
}
```

Cujo resultado é 130 e seja o seguinte código

```
int main() {
    int A,B,C,D;
    A=4,B=1,C=2;
    if(C<=12){
        while(B<=12){
            C=A+3;
            B=B+4;
        }
    }else{
        C=C-8;
        B=B+3;
    }
    cout<<C+A+B;
}
```

cujo resultado é 24.

👉 Para você fazer

Exercício 1

```
int main() {
    int A,B,C,D;
    A=8,B=6,C=4;
    while(B<19){
        A=A-8;
        A=A+4;
        B=B+4;
    }
    cout<<A+A+B;
}
```

Informe no quadro próprio o valor que foi impresso.

Exercício 2

```
int main() {
    int A,B,C,D;
    A=1,B=7,C=3;
    if(C<18){
        while(B<=18){
            C=C+6;
            B=B+4;
        }
    }else{
        C=C+8;
        B=B+4;
    }
    cout<<C+C+B;
}
```

Exercício 3

```
int main() {
    int A,B,C,D;
    A=1,B=3,C=3;
    if(A>=19){
        while(B<=19){
            A=C-3;
            B=B+2;
        }
    }
```

```
}else{
    while(B<=18){
        A=A-2;
        B=B+2;
    }
}
cout<<A+C+B;
}
```

Exercício 4

```
int main() {
    int A,B,C,D;
    A=7,B=2,C=6;
    D=5;
    A=A+6+2;
    while(A>6){
        B=B+2;
        while(C<=18){
            D=B+3;
            C=C+5;
        }
        D=B-16;
        A=A-4;
    }
    cout<<D+C+B;
}
```

Exercício 5

```
int main() {
    int A,B,C,D;
    A=7,B=2,C=4;
    D=6;
    C=C+4+2;
    while(C>2){
        B=B+3;
        if(A<7){
            while(A<=17){
                B=D+6;
                A=A+4;
            }
            B=D+19;
        }else{
            D=D+5;
        }
        C=C-4;
    }
    cout<<B+C;
}
```

Exercício 6

```
int main() {
    int A,B,C,D;
    A=8,B=2,C=3;
    D=4;
    B=B+3+2;
    while(B>7){
        A=A+3;
        if(C>7){
            while(D<=18){
                D=D+4;
            }
            C=C+4;
        }else{
            C=A+12;
        }
        if(A<=6){
            C=C+4;
        }else{
            C=C+7;
        }
        B=B-2;
    }
    cout<<B+A;
}
```

Exercício 7

```
int main() {
    int A,B,C,D;
    A=4,B=1,C=1;
    while(C<=18){
        B=B+5;
        B=A+3;
        C=C+4;
    }
    cout<<B+B+C;
}
```

Exercício 8

```
int main() {
    int A,B,C,D;
    A=8,B=1,C=4;
    if(C<20){
        while(A<20){
            C=B+2;
            A=A+4;
        }
    }else{
        B=C-1;
        A=A+1;
    }
    cout<<C+B+A;
}
```

Exercício 9

```
int main() {
    int A,B,C,D;
    A=5,B=4,C=8;
    if(C==13){
        while(A<13){
            C=C+1;
            A=A+1;
        }
    }else{
        while(A<18){
            B=B+7;
            A=A+1;
        }
    }
    cout<<C+C+A;
}
```

Exercício 10

```
int main() {
    int A,B,C,D;
    A=4,B=8,C=6;
    D=7;
    A=A+6+8;
    while(A>=2){
        C=A-6;
        if(C>=8){
            while(B<=16){
                C=D-8;
                B=B+5;
            }
            C=C+18;
        }else{
            C=C+9;
        }
        A=A-2;
    }
    cout<<D+B;
}
```

Responda aqui:

1	2
3	4
5	6
7	8
9	10



Repetição em Python

Como qualquer linguagem estruturada moderna, Python tem as ferramentas básicas para interar. São elas: while, for, bem como as saídas break e continue. A mais amigável é while. Já a a mais potente é for, pois funciona como um interador, que é uma extensão e tanto ao conceito de indexador. Já os comandos break permitem o abandono de um ciclo, enquanto continue serve para abandonar a interação atual. Veja alguns exemplos:

```
i=3
while i<10:
    print(i,' ')
    if i%7==0:
        break
    i=i+1
print('acabou')
```

```
i=3
while i<10:
    print(i,' ')
    if i%7==0:
        continue
    i=i+1
print('acabou')
```

```
s="Curitiba - Paraná - Brasil"
qtd=0
for c in s:
    if c=="a":
        qtd=qtd+1
print("achei ",qtd," a")
```

Exercícios Nos exercícios a seguir, você deve simular o interpretador Python e descobrir qual valor vai ser impresso ao final. Todos os códigos estão sintaticamente corretos. A sugestão é que depois de ter calculado o valor na ponta do lápis, você confira o resultado com um computador real.

Exemplos

Seja o programa

```
def AA():
    A=5;B=5;C=3
    if C>=18:
        while B<=18:
            C=C-3
            B=B+4
    else:
        C=C+4
        B=B+2
    print(C+C+B)
AA()
```

Cujo resultado é 21 e seja o seguinte código

```
def AA():
    A=4;B=3;C=6
    D=1
    C=C+6+3
    while C>6:
        B=D+7
        while A<=14:
            D=B+4
            A=A+2
            D=B+19
            C=C-1
        print(C+B+B)
AA()
```

cujo resultado é 438.

👉 Para você fazer
Exercício 1

```
def AA():
    A=3;B=1;C=4
    while C<13:
        B=B-4
        B=B+6
        C=C+4
    print(B+B+C)
AA()
```

Informe no quadro próprio o valor que foi impresso.

Exercício 2

```
def AA():
    A=6;B=1;C=5
    if A>18:
        while B<18:
            A=C+1
            B=B+4
        else:
            C=A+1
            B=B+1
    print(A+C+B)
AA()
```

Exercício 3

```
def AA():
    A=1;B=1;C=3
    if A>=16:
        while C<16:
            A=A+7
            C=C+3
        else:
            while C<18:
                B=B+3
                C=C+2
            print(A+A+C)
AA()
```

Exercício 4

```
def AA():
    A=8;B=4;C=3
    D=2
    B=B+3+4
    while B>3:
        A=B-5
        while D<14:
            A=C+5
            D=D+4
            A=C-17
            B=B-2
        print(B+B+D)
AA()
```

Exercício 5

```
def AA():
    A=2;B=1;C=7
    D=3
    B=B+7+1
    while B>=3:
        C=A-7
        if D>4:
            while A<13:
                D=D-2
                A=A+5
                D=D+13
            else:
                D=C+6
                B=B-3
        print(B+C)
AA()
```

Exercício 6

```
def AA():
    A=1;B=7;C=2
    D=5
    A=A+2+7
    while A>3:
        C=B+2
        if C<8:
            while B<14:
                B=B+4
                C=D+4
            else:
                C=D-13
        if D<8:
            C=C+6
        else:
            D=C-7
            A=A-1
    print(C+B)
AA()
```

Exercício 7

```
def AA():
    A=6;B=1;C=2
    D=4
    A=A+2+1
    while A>6:
        B=A+7
        if D<=10:
```

```
        while D<=17:
            D=D+4
            C=C+1
        else:
            B=C+14
        if C>10:
            B=B+10
        else:
            C=C+4
            A=A-4
    print(C+D)
AA()
```

Exercício 8

```
def AA():
    A=2;B=8;C=1
    D=5
    C=C+1+8
    while C>3:
        A=C+3
        if B<5:
            while D<=19:
                A=B+8
                D=D+5
                B=B-19
            else:
                A=B-5
                C=C-1
        print(A+C)
AA()
```

Exercício 9

```
def AA():
    A=7;B=5;C=3
    D=1
    C=C+3+5
    while C>=5:
        A=B-8
        while D<=12:
            B=B+2
            D=D+1
        A=A+16
        C=C-1
    print(C+C+A)
AA()
```

Exercício 10

```
def AA():
    A=7;B=5;C=8
    if A=17:
        while B<=17:
            A=A-1
            B=B+4
        else:
            while B<=12:
                C=C+1
                B=B+3
            print(A+A+B)
AA()
```

Responda aqui:

1	2
3	4
5	6
7	8
9	10



Primeiros passos em programação - C++

Esta folha tem como objetivo ajudar a dar os primeiros passos na programação. Por ser uma atividade humana complexa e que exige muito de abstração, a melhor maneira de avançar é com calma e sem pressa. O pré-requisito para concluir esta atividade com sucesso é ter diante de si um computador (ou tablet ou celular) com um C++ funcional. Se ainda não tem, veja na documentação disponibilizada (em <http://algoritmovivo.com> aba atividades) como obter este ambiente.

1. Alô Mundo Tradicionalmente, o primeiro programa funcional que se produz em qualquer plataforma nova tem este nome e não faz quase nada: apenas mostra uma saudação. O objetivo de sua produção é apenas conhecer, manipular e certificar o processo completo de produzir, compilar, testar e executar qualquer programa.

```
#include<iostream>
using namespace std;
int main(){
    cout<<"Alo mundo"<<endl;
}
```

Observação: nos próximos exemplos, vão ser omitidos os 2 comandos iniciais que sempre devem estar presentes.

2. Soma de 5 números O objetivo é escrever um programa que leia 5 números e imprima a sua soma. Eis uma possível solução

```
int main(){
    int a,b,c,d,e,s;
    cout<<"Informe os 5 numeros ";
    cin>>a>>b>>c>>d>>e;
    s=a+b+c+d+e;
    cout<<s<<endl;
}
```

3. Média de n números Agora o problema é mais complexo, pois não se sabe quantos números serão informados. Há necessidade de se usar uma *sentinela* que vem a ser um valor que sinaliza alguma condição: neste caso o final dos dados. Vamos usar a sentinela valendo -1, e enquanto este número não for digitado o programa segue lendo dados. Veja uma possível solução

```
int main(){
    int a,sentinela=0, contador=0;
    float soma=0, media;
    while (1==1){
        cout<<"Informe ";
        cin>>a;
        if (a==--1){
            break;
        }
        contador++;
        soma=soma+a;
    }
    media=soma/contador;
    cout<<"Media "<<media;
}
```

4. Desvio padrão O desvio padrão é uma medida sobre o afastamento médio de um conjunto de valores em relação à média desses mesmos valores. Tem como fórmula

$$\sigma = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

onde $\bar{x}$ é a média dos valores x . Há aqui uma dificuldade que é a necessidade de processar 2 vezes cada valor (a primeira para calcular a média e depois para achar a distância da média). Então, na leitura dos dados, eles terão que ser guardados para uso posterior. Veja

```
#include<cmath>
int main(){
    int i,a,sent=0,ctd=0;
    float media,soma=0,b,dp;
    int dados[1000];
    while(1==1){
        cout<<"Informe ";
        cin>>a;
        if (a==--1){
            break;
        }
        dados[ctd]=a;
        soma=soma+a;
        ctd++;
    }
    media=soma/ctd;
    soma=0;
    i=0;
    while (i<ctd){
        b=dados[i]-media;
        soma=soma+b*b;
        i++;
    }
    dp=soma/ctd;
    dp=pow(dp,0.5);
    cout<<"dp= "<<dp;
}
```

5. Simulador de dados Nosso problema agora é apostar na lota. Como estamos sem inspiração, vamos construir um programa que sugira uma aposta. Como seria bom se ele sempre acertasse... Nesta hora precisamos definir um número aleatório ou no caso da programação um número pseudo-aleatório: Trata-se de um número gerado ao acaso. Há muitas maneiras de fazer isto em C++. Acompanhe

```
#include<stdlib.h>
#include<algorithm>
#include<ctime>
int main(){
    srand(time(0));
    int i=0;
    int n[6];
    while (i<6){
        n[i]=(rand()%60)+1;
        i++;
    }
    sort(n,n+6);
    for (i=0;i<6;i++){
        cout<<n[i]<<" ";
    }
}
```

6. Amplitude Suponha que você vai receber as temperaturas ao meio dia, de 7 dias seguidos em Curitiba. Você quer determinar a amplitude térmica que vem a ser a maior temperatura - menor temperatura no período. Acompanhe

```
int main(){
    float temp[7];
    float maior=-999999, menor=999999, ampl;
    int i;
    for (i=0;i<7;i++){
        cout<<"Temperatura ";
        cin>>temp[i];
    }
    for (i=0;i<7;i++){
        if (temp[i]>maior){
            maior=temp[i];
        }
        if (temp[i]<menor){
            menor=temp[i];
        }
    }
    ampl=maior-menor;
    cout<<"ampl = "<<ampl;
}
```

7. Quantas vogais ? Agora o objetivo é examinar uma frase e contar quantas vogais há nela. Veja

```
#include<string.h>
int main(){
    int i,ct=0;
    string frase;
    cout<<"Informe a frase ";
    getline(cin,frase);
    for (i=0;i<frase.size();i++){
        if ((frase[i]=='a') || (frase[i]=='e') ||
            frase[i]=='i' || (frase[i]=='o') ||
            (frase[i]=='u'))){

```

```
                ct++;
            }
        }
        cout<<ct;
    }
}
```

Para você fazer

Use o programa 2 para somar os números abaixo

839 132 378 44 78

e transcreva o resultado do programa na tabela abaixo

Use o programa 3 para achar a média entre os valores

84 59 835 937 496 774 410 807 773

Ache o desvio padrão desta distribuição (programa 4)

79 621 86 955 288 934 655 339 388

Ache a amplitude das 7 temperaturas (programa 6)

24.2 20.3 26.2 15.8 25.7 21.4 17.4

Agora é com você Agora você deve exercitar seus poderes de criação e escrever os programas abaixo pedidos.

8. Qual o maior salário ? Suponha que dada uma certa coleção de salários, você precisa escrever um programa de computador que informe o valor do maior salário. Escreva o programa, teste-o e depois execute-o com os dados abaixo

2791 3163 2580 2131 3598 2755

Lance o valor informado na tabela abaixo e ENTREGUE JUNTO A LISTAGEM DO PROGRAMA QUE VOC ESCREVEU.

9. Quantos Pês há na frase Escreva um programa que conte quantas letras P (maiúsculas ou minúsculas) há numa frase. Execute o programa com a frase

Senti pelas faces Vaguei pelas serras

Lance o valor informado na tabela abaixo e ENTREGUE JUNTO A LISTAGEM DO PROGRAMA QUE VOC ESCREVEU.

10. Desconto na loja Uma loja no Shopping aqui ao lado bolou a seguinte promoção. Se o valor de uma compra está acima ou igual a 200 e abaixo de 500 terá um desconto de 15% e se é maior ou igual a 500 reais terá um desconto de 20%. Escreva um programa que receba o valor total de uma compra e calcule o novo valor a pagar, depois de aplicado o desconto. A compra tem um valor de

640

Lance o valor real a pagar (depois do desconto) no lugar indicado e apresente a listagem do programa.

11. Volume Suponha uma caixa de sapatos (eventualmente meio grande) cujas dimensões dadas em centímetros são

380 608 201

Escreva um programa que receba esses 3 valores e calcule o volume em m^3 . Lance o valor correto abaixo e entregue a listagem do seu programa junto. Responda aqui:

2)soma	3)média	4)d.p.	6)amplit
8)salario	9)P's	10)compra	11)caixa

Não esqueça de anexar a listagem dos 4 últimos programas devidamente identificada com o seu nome e com o número 1



Primeiros passos programação II - C++

Continuando as tarefas de introdução à programação, vamos a mais alguns casos.

1. Uma lista está em ordem crescente? Dada uma lista numérica, como por exemplo 1, 7, 23, 55, 61, 61, 70, 99, a pergunta é se ela está em ordem crescente ou não. Olhando a definição diz-se que a lista X está em ordem crescente sse

$$x_i \geq x_j \quad \forall \quad i > j$$

É mais uma tarefa para usar uma sentinela, já que todos os pares possíveis precisarão ser testados e basta que um par não seja verdadeiro, para a lista deixar de estar em ordem. Acompanhe:

```
#include<iostream>
using namespace std;
int orcre(int tamanho, int x[]) {
    int i=1,senti=1;
    while ((i<tamanho) && (senti==1)){
        if (x[i]<x[i-1]){
            senti=0;
        }
        i++;
    }
    return senti;
}
int main(){
    int y[100]={1,2,4,33,5,6};
    cout<<orcre(5,y); }
```

Observação: nos próximos exemplos desta folha vai-se deixar de incluir os 2 comandos iniciais, por falta de espaço, mas eles sempre estarão lá.

2. determinante de matriz 3×3 Para este exercício precisa-se introduzir o conceito de matriz. É um arranjo bidimensional de dados. Cada elemento vai ser individualizado pelo uso de 2 índices: o da linha e o da coluna, nesta ordem. Uma matriz é criada em C++ como segue. Acompanhe

```
float a[3][3]={1,2,3},{6,7,8},{10,11,12};
```

Este exercício pede que você receba uma matriz assim formada (3×3), calcule e devolva o valor do determinante da matriz. Use a regra de Sarrus, como mostrado aqui:



```
float a[3][3]={2,4,5},{3,7,0},{1,1,5};
float sarrus(float x[3][3]){
    float a1,a2;
    a1=x[0][0]*x[1][1]*x[2][2];
    a1=a1+x[0][2]*x[1][0]*x[2][1];
    a1=a1+x[0][1]*x[1][2]*x[2][0];
    a2=x[0][2]*x[1][1]*x[2][0];
    a2=a2+x[0][1]*x[1][0]*x[2][2];
    a2=a2+x[0][0]*x[1][2]*x[2][1];
    return a1-a2;
}
main() {
    cout<<sarrus(a); }
```

3. Dia mais frio Suponha que você tem uma coleção de 10 temperaturas mínimas em Curitiba referentes aos primeiros dez dias do mês de julho último. A pergunta que se quer responder é qual foi o dia mais frio nessa dezena. Então, primeiro precisa-se descobrir qual a temperatura mínima da dezena e depois em que dia isso ocorreu. Veja

```
int dmfrio(int x[10]){
    int i=0, mini=+9999999,dmin;
    while (i<10){
        if (x[i]<mini){
            mini=x[i];

```

```
            dmin=i;
        }
        i++;
    }
    return dmin+1;
}
int main(){
    int te[10]={12,17,13,14,8,19,26,10,10,10};
    cout<<dmfrio(te); }
```

4. Quantos estados ? No modelo tributário brasileiro, os estados se endividam junto à União, e mesmo quando o agente emprestado é outro, a União fica como avalista. Desse modo, vários devem muito dinheiro à União. Suponha uma lista com n valores correspondendo à dívida de cada estado e do DF com o Governo Central. n pode ser menor que 27, já que há estados que nada devem. A pergunta a responder é quantos estados respondem por 50% da dívida ? Para responder, primeiro há que ordenar a lista em ordem decrescente, somar tudo e depois ir contando até ultrapassar a metade desse valor. Observe

```
#include<algorithm>
int metadiv(int tam,float x[28]){
    float som=0,met;
    int i;
    sort(x,x+tam);
    for (i=0;i<tam;i++){
        som=som+x[i];
    }
    met = som /2.0;
    i=tam;
    som=0;
    while (som<met){
        som=som+x[i];
        i--;
    }
    return tam-(i+1);
}
int main(){
    float di[28]={20,33,54,90,100,10,10,10};
    cout<<metadiv(8,di); }
```

5. Plataforma de petróleo O fundo da baía de Guaratuba foi mapeado em células de $100m \times 100m$. Em cada célula dessa um número real indica a chance de encontrar petróleo cavando no centro da célula. A Petrobras achou uma sonda que é capaz de extrair o óleo da célula onde está e dos seus até 8 vizinhos. Supondo que você tem a matriz (imagina 8 linhas por 10 colunas) com tais números, a pergunta é onde (L,C) pôr a sonda ?

```
int sonda(int tl, int tc, float fu[100][100]){
    float maxi=-999999,soma;
    int i,j,mai,maj;
    for (i=0;i<tl;i++){
        for (j=0;j<tc;j++){
            soma=fu[i-1][j-1]+fu[i-1][j]+fu[i-1][j+1];
            soma=soma+fu[i][j-1]+fu[i][j]+fu[i][j+1];
            soma=soma+fu[i+1][j-1]+fu[i+1][j]+fu[i+1][j+1];
            if (soma>maxi){
                maxi=soma;
                mai=i;
                maj=j;
            }
        }
    }
    return ((mai+1)*1000)+maj;
}
int main(){
    float f[100][100]={1,3,3,5},{7,4,3,1},
        {10,1,1,2},{7,3,2,8},{9,8,2,1};
    cout<<sonda(4,5,f); }
```

6. Jambo e Ruivão Há dois insetos na natureza: O jambo que tem um ciclo de vida de 13 anos (passa 12 anos hibernando e surge aos milhões no 13º ano) e o ruivão cujo ciclo é de 17 anos (idem idem). O jambo se alimenta preferencialmente de ruivões e quando não os encontra tem muita dificuldade de sobreviver: mais de 90% dos jambos que nascem não chegam à idade adulta por falta de comida. A pergunta é de quanto em quanto tempo, há uma explosão de jambos ? Obviamente isso só vai ocorrer quando ambos nascerem no mesmo ano.

```
int jambo(int x, int y){
    int qt;
    qt=x;
    while (i==1){
        if (qt%y==0){
            return qt;

```

```
        }
        qt=qt+x;
    }
}
int main(){
    cout<<jambo(13,17); }
```

Para você fazer

Para os exercícios 1 a 6, você deve executar os programas acima sobre a seguinte coleção de dados

- 1 11 49 54 65 77 81 85 90 99
- 45 6 9
8 25 8
3 4 30
- 19 18 16.7 16 19.2 14 12 19.9 13 10.7
- 72 79 458 188 834 680 343 999
- 371 216 706 213 785 599 203 144 857
658 592 298 859 430 747 954 733 334
784 666 863 185 81 487 101 939 704
418 878 190 84 123 413 571 269 797
421 546 549 366 896 870 133 789 763
- 9 10

Para os 3 programas a seguir, você deve construir o programa pedido e depois executá-lo com os dados entregues, achando o número em questão e também entregando a listagem do programa que você escreveu.

7. Maiores que os vizinhos A questão é contar quantos elementos em uma lista numérica dada são maiores que os seus vizinhos. Note que o primeiro e o último elementos não tem um dos vizinhos e fica a pergunta: o que fazer com eles ? Cada problema dá a sua solução. Neste caso, os elementos inicial e final não serão testados, isto é: devem ser desconsiderados. Execute seu programa com estes dados

- 477 892 793 628 876 92 476 117 767 319

8. x é primo ? Da definição matemática, o inteiro positivo x é primo se ele é maior do que 1 e só tem 2 divisores: a unidade e o próprio x . Implementando exatamente esta definição, como ficaria ? Execute seu programa com o número

- 788677

9. Funcionário da quinzena Os 11 funcionários de uma lanchonete do Bloco de Exatas de importante universidade são identificados por um número único e fixo: 1..11. A cada dia a lanchonete escolhe o funcionário do dia (aquele que foi melhor avaliado nesse dia) e registra seu número em uma lista. No final da quinzena, essa lista contém 15 números entre 1 e 11. A pergunta que se quer responder é qual o funcionário da quinzena ? Ou seja, aquele que mais vezes foi citado. Se houver empate, deve ser mostrado o que tiver o maior número.

- 5 10 7 3 11 3 9 9 2 8 8 4 3 1 11

Responda aqui:

1) cresce?	2) det	3) dia frio	4) quantos est?
5) L. petr	5) C. petr	6) jambo	7) > viz?
8) primo?	9) func?	////// ////// ////// ////// //////	////// ////// ////// ////// //////

observações

1. Não esqueça de anexar a listagem dos 3 últimos programas devidamente identificada com o seu nome e com o número 1
2. Em todas as contagens desta folha considere a origem dos índices como UM e não como zero.



Primeiros passos em programação-python

Esta folha tem como objetivo ajudar a dar os primeiros passos na programação. Por ser uma atividade humana complexa e que exige muito de abstração, a melhor maneira de avançar é com calma e sem pressa. O pré-requisito para concluir esta atividade com sucesso é ter diante de si um computador (ou tablet ou celular) com um Python funcional. Se ainda não tem, veja na documentação disponibilizada (em <http://algoritmovivo.com> aba atividades) como obter este ambiente.

1. Alô Mundo Tradicionalmente, o primeiro programa funcional que se produz em qualquer plataforma nova tem este nome e não faz quase nada: apenas mostra uma saudação. O objetivo de sua produção é apenas conhecer, manipular e certificar o processo completo de produzir, compilar, testar e executar qualquer programa.

1. Localize o Python
2. Abra um (ou o) editor de textos e transcreva

```
def alo(): # definindo f. de nome alo
    print("Alo mundo") #a f. imprime a msg
alo() # agora executa-se a função alo()
```

3. Pressione F5 (ou salve ou arquivo)
4. dê a ele o nome alo (no disco ficará alo.py)
5. veja o resultado. Se não funcionar, veja o erro e corrija-o

2. Soma de 5 números O objetivo é escrever um programa que leia 5 números e imprima a sua soma. Eis uma possível solução

```
def s5n():
    a=int(input("Informe primeiro numero "))
    b=int(input("Informe segundo numero "))
    c=int(input("Informe terceiro numero "))
    d=int(input("Informe quarto numero "))
    e=int(input("Informe quinto numero "))
    s=a+b+c+d+e
    print(s)
s5n()
```

3. Média de n números Agora o problema é mais complexo, pois não se sabe quantos números serão informados. Há necessidade de se usar uma *sentinela* que vem a ser um valor que sinaliza alguma condição: neste caso o final dos dados. Vamos usar a sentinela valendo -1, e enquanto este número não for digitado o programa segue lendo dados. Veja uma possível solução

```
def media():
    sentinela=0
    soma=0
    contador=0
    while i==1:
        a=int(input("informe "))
        if a==--1:
            break
        contador=contador+1
        soma=soma+a
        media=soma/contador
        print(media)
media()
```

4. Desvio padrão O desvio padrão é uma medida sobre o afastamento médio de um conjunto de valores em relação à média desses mesmos valores. Tem como fórmula

$$\sigma = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

onde $\bar{x}$ é a média dos valores x . Há aqui uma dificuldade que é a necessidade de processar 2 vezes cada valor (a primeira para calcular a média e depois para achar a distância da média). Então, na leitura dos dados, eles terão que ser guardados para uso posterior. Veja

```
def depad():
    sentinela=0
    soma=0
    contador=0
    dados=[]
    while i==1:
        a=int(input("informe "))
        if a==--1:
            break
        dados=dados+[a]
        contador=contador+1
        soma=soma+a
        media=soma/contador
        i=0
        soma=0
        while i<contador:
            soma=soma+(dados[i]-media)**2
            i=i+1
        dp=soma/contador
        dp=dp**0.5
        print(dp)
depad()
```

5. Simulador de dados Nosso problema agora é apostar na lota. Como estamos sem inspiração, vamos construir um programa que sugira uma aposta. Como seria bom se ele sempre acertasse... Nesta hora precisamos definir um número aleatório ou no caso da programação um número pseudo-aleatório: Trata-se de um número gerado ao acaso. Há muitas maneiras de fazer isto em Python, mas a mais imediata é carregar o pacote numpy com o nome np e do sub-pacote random, chamar a função randint(n) onde n é o limite máximo (inalcançável) do inteiro a gerar. Acompanhe

```
import numpy as np
def loto():
    a=np.random.randint(1,61,6)
    a.sort()
    print(a)
loto()
```

6. Amplitude Suponha que você vai receber as temperaturas ao meio dia, de 7 dias seguidos em Curitiba. Você quer determinar a amplitude térmica que vem a ser a maior temperatura - menor temperatura no período. Acompanhe

```
def ampli():
    tem=[0]*7
    i=0
    while i<7:
        t=float(input("temperatura "))
        tem[i]=t
        i=i+1
    maior=999999
    i=0
    while i<7:
        if tem[i]>maior:
            maior=tem[i]
            i=i+1
    menor=-999999
    i=0
    while i<7:
        if tem[i]<menor:
            menor=tem[i]
            i=i+1
    ampl=maior-menor
    print(ampl)
ampli()
```

7. Quantas vogais ? Agora o objetivo é examinar uma frase e contar quantas vogais há nela. Veja

```
def vogais():
    a=input("Informe a frase ")
    i=0
    ct=0
    while i<len(a):
        if a[i]=="a" or a[i]=="e" or a[i]=="i" \
           or a[i]=="o" or a[i]=="u":
            ct=ct+1
            i=i+1
        print(ct)
vogais()
```

Para você fazer

Use o programa 2 para somar os números abaixo

520 287 437 511 314

e transcreva o resultado do programa na tabela abaixo

Use o programa 3 para achar a média entre os valores

804 237 656 256 112 434 94 615 184

Ache o desvio padrão desta distribuição (programa 4)

945 770 952 274 172 738 615 575 42

Ache a amplitude das 7 temperaturas (programa 6)

28.2 22.3 17.5 15.5 20.5 20.7 26.6

Agora é com você Agora você deve exercitar seus poderes de criação e escrever os programas abaixo pedidos.

8. Qual o maior salário ? Suponha que dada uma certa coleção de salários, você precisa escrever um programa de computador que informe o valor do maior salário. Escreva o programa, teste-o e depois execute-o com os dados abaixo

2357 2942 3557 2903 3553 3238

Lance o valor informado na tabela abaixo e ENTREGUE JUNTO A LISTAGEM DO PROGRAMA QUE VOC ESCREVEU.

9. Quantos Pês há na frase Escreva um programa que conte quantas letras P (maiúsculas ou minúsculas) há numa frase. Execute o programa com a frase

Com placido rosto, Sereno e composto,

Lance o valor informado na tabela abaixo e ENTREGUE JUNTO A LISTAGEM DO PROGRAMA QUE VOC ESCREVEU.

10. Desconto na loja Uma loja no Shopping aqui ao lado bolou a seguinte promoção. Se o valor de uma compra está acima ou igual a 200 e abaixo de 500 terá um desconto de 15% e se é maior ou igual a 500 reais terá um desconto de 20%. Escreva um programa que receba o valor total de uma compra e calcule o novo valor a pagar, depois de aplicado o desconto. A compra tem um valor de

947

Lance o valor real a pagar (depois do desconto) no lugar indicado e apresente a listagem do programa.

11. Volume Suponha uma caixa de sapatos (eventualmente meio grande) cujas dimensões dadas em centímetros são

402 669 616

Escreva um programa que receba esses 3 valores e calcule o volume em m^3 . Lance o valor correto abaixo e entregue a listagem do seu programa junto. Responda aqui:

2)soma	3)média	4)d.p.	6)amplit
8)salario	9)P's	10)compra	11)caixa

Não esqueça de anexar a listagem dos 4 últimos programas devidamente identificada com o seu nome e com o número 1



- 1 - /

Primeiros passos programação II-
python

Continuando as tarefas de introdução à programa-
ção, vamos a mais alguns casos.

1. Uma lista está em ordem cres-
cente? Dada uma lista numérica, como por
exemplo [1, 7, 23, 55, 61, 61, 70, 99], a pergunta é
se ela está em ordem crescente ou não. Olhando a
definição diz-se que a lista X está em ordem cres-
cente sse

$$x_i \geq x_j \quad \forall \quad i > j$$

É mais uma tarefa para usar uma sentinela, já que
todos os pares possíveis precisarão ser testados e
basta que um par não seja verdadeiro, para a lista
deixar de estar em ordem. Acompanhe:

```
def orcre(x):  
    i=1  
    sentinela=True  
    while i<len(x) and sentinela==True:  
        if x[i]<x[i-1]:  
            sentinela=False  
            i=i+1  
    return sentinela  
print(orcre([1, 7, 23, 55, 61, 61, 70, 99]))
```

2. determinante de matriz 3 x 3
Para este exercício precisa-se introduzir o conceito
de matriz. É um arranjo bidimensional de dados.
Cada elemento vai ser individualizado pelo uso de
2 índices: o da linha e o da coluna, nesta ordem.
Uma matriz é criada em Python chamando-se o
pacote Numpy (Numerical Python). Acompanhe

```
>>> import numpy as np  
>>> a = np.array([[1,2,3],[6,7,8],[10,11,12]])  
>>> print(a)  
[[ 1  2  3]  
 [ 6  7  8]  
 [10 11 12]]
```

Este exercício pede que você receba uma matriz as-
sim formada (3 x 3), calcule e devolva o valor do
determinante da matriz. Use a regra de Sarrus,
como mostrado aqui:



```
def sarrus(x):  
    a1=x[0,0]*x[1,1]*x[2,2]  
    a1=a1+x[0,2]*x[1,0]*x[2,1]  
    a1=a1+x[2,0]*x[0,1]*x[1,2]  
    a2=x[0,2]*x[1,1]*x[2,0]  
    a2=a2+x[0,1]*x[1,0]*x[2,2]  
    a2=a2+x[0,0]*x[2,1]*x[1,2]  
    return a1-a2  
import numpy as np  
a=np.array([[2,4,5],[3,7,0],[1,1,5]])  
print(sarrus(a))  
print(np.linalg.det(a))
```

Note que o valor pode ser verificado com o comando
linalg.det(x) que já está programado em numpy e
vale para qualquer matriz quadrada.

3. Dia mais frio Suponha que você tem
uma coleção de 10 temperaturas mínimas em Cu-
ritiba referentes aos primeiros dez dias do mês de
julho último. A pergunta que se quer responder é
qual foi o dia mais frio nessa dezena. Então, pri-
meiro precisa-se descobrir qual a temperatura mí-
nima da dezena e depois em que dia isso ocorreu.
Veja

```
def dmfrio(x):  
    i=0  
    mini=+9999999  
    while i<len(x):
```

```
        if x[i]<mini:  
            mini=x[i]  
            dmin=i+1  
            i=i+1  
    return dmin  
print(dmfrio([12,17,13,14,8,19,26]))
```

4. Quantos estados ? No modelo tri-
butário brasileiro, os estados se endividam junto à
União, e mesmo quando o agente emprestado é ou-
tro, a União fica como avalista. Desse modo, vários
devem muito dinheiro à União. Suponha uma lista
com n valores correspondendo à dívida de cada es-
tado e do DF com o Governo Central. n pode ser
menor que 27, já que há estados que nada devem.
A pergunta a responder é quantos estados respon-
dem por 50% da dívida ? Para responder, primeiro
há que ordenar a lista em ordem decrescente, so-
mar tudo e depois ir contando até ultrapassar a
metade desse valor. Observe

```
def metadiv(x):  
    x.sort(reverse=True)  
    met=sum(x)/2  
    som=0  
    i=0  
    while som<met:  
        som=som+x[i]  
        i=i+1  
    return i  
print(metadiv([10,20,5,15,49]))  
print(metadiv([1,2,3,4,5,6,7]))
```

5. Plataforma de petróleo O fundo
da baía de Guaratuba foi mapeado em células de
100m x 100m. Em cada célula dessa um número
real indica a chance de encontrar petróleo cavando
no centro da célula. A Petrobras achou uma sonda
que é capaz de extrair o óleo da célula onde está e
dos seus até 8 vizinhos. Supondo que você tem a
matriz (imagina 8 linhas por 10 colunas) com tais
números, a pergunta é onde (L,C) pôr a sonda ?

```
def sonda(x):  
    maxi=-9999999  
    i=1  
    while i<len(x)-1:  
        j=1  
        while j<len(x[0])-1:  
            soma=x[i-1,j-1]+x[i-1,j]+x[i-1,j+1]  
            soma=soma+x[i,j-1]+x[i,j]+x[i,j+1]  
            soma=soma+x[i+1,j-1]+x[i+1,j]+x[i+1,j+1]  
            if soma>maxi:  
                maxi=soma  
                mai=i  
                maj=j  
            j=j+1  
            i=i+1  
        return [mai+1,maj+1]  
import numpy as np  
a=np.array([[1,4,3,5],[7,4,3,1],[10,1,1,2],  
            [7,3,2,8],[9,8,2,1]])  
print(sonda(a))
```

6. Jambo e Ruivão Há dois insetos na
natureza: O jambo que tem um ciclo de vida de 13
anos (passa 12 anos hibernando e surge aos milhões
no 13º ano) e o ruivão cujo ciclo é de 17 anos (idem
idem). O jambo se alimenta preferencialmente de
ruivões e quando não o encontra tem muita di-
ficuldade de sobreviver: mais de 90% dos jambos
que nascem não chegam à idade adulta por falta de
comida. A pergunta é de quanto em quanto tempo,
há uma explosão de jambos ? Obviamente isso só
vai ocorrer quando ambos nascerem no mesmo ano.

```
def jambo(x,y):  
    qt=x  
    while i==1:  
        if qt%y==0:  
            return qt  
        qt=qt+x  
    print(jambo(13,17))  
    print(jambo(8,6))
```

📖 Para você fazer

Para os exercícios 1 a 6, você deve executar os pro-
gramas acima sobre a seguinte coleção de dados

- 1. 13 24 27 34 47 49 50 51 67 84
- 2. 24 5 4
2 27 9

- 4 6 12
- 3. 16 11 13 14.1 20 14.1 18.5 16 18.5 15
- 4. 640 248 361 575 507 876 357 560
- 5. 802 593 429 121 627 213 253 122 792
671 759 737 93 483 197 977 723 819
617 839 861 25 973 436 854 770 150
83 767 125 385 503 511 387 130 664
314 524 586 625 874 347 470 928 148

6. 6 9

Para os 3 programas a seguir, você deve construir o
programa pedido e depois executá-lo com os dados
entregues, achando o número em questão e tam-
bém entregando a listagem do programa que você
escreveu.

7. Maiores que os vizinhos A questão
é contar quantos elementos em uma lista numérica
dada são maiores que os seus vizinhos. Note que
o primeiro e o último elementos não tem um dos
vizinhos e fica a pergunta: o que fazer com eles ?
Cada problema dá a sua solução. Neste caso, os
elementos inicial e final não serão testados, isto é:
devem ser desconsiderados. Execute seu programa
com estes dados

7. 788 690 275 392 284 521 804 793 42 277 504

8. x é primo ? Da definição matemática, o
inteiro positivo x é primo se ele é maior do que 1 e
só tem 2 divisores: a unidade e o próprio x. Imple-
mentando exatamente esta definição, como ficaria
? Execute seu programa com o número

8. 306083

9. Funcionário da quinzena Os 11
funcionário de uma lanchonete do Bloco de Exa-
tas de importante universidade são identificados
por um número único e fixo: 1..11. A cada dia
a lanchonete escolhe o funcionário do dia (aquele
que foi melhor avaliado nesse dia) e registra seu
número em uma lista. No final da quinzena, essa
lista contém 15 números entre 1 e 11. A pergunta
que se quer responder é qual o funcionário da quin-
zena ? Ou seja, aquele que mais vezes foi citado.
Se houver empate, deve ser mostrado o que tiver o
maior número.

9. 11 10 7 8 7 8 11 5 3 11 5 10 6 10 3

Responda aqui:

1)cresce?	2)det	3)dia frio	4)quantos est?
5)L. petr	5)C. petr	6)jambo	7)> viz?
8)primo?	9)func?	//////// //////// //////// //////// ////////	//////// //////// //////// //////// ////////

observações

- 1. Não esqueça de anexar a listagem dos 3 úl-
timos programas devidamente identificada
com o seu nome e com o número 1
- 2. Em todas as contagens desta folha considere
a origem dos índices como UM e não como
zero.



Gerador de movimentos de xadrez

O xadrez tem um papel fundamental na Inteligência Artificial, e por conseguinte na ciência da computação. Ele é suficientemente compacto para poder ser manuseado dentro de um computador, mesmo um computador da década de 50, com poucas centenas de bytes de memória. Mas, é abrangente e complexo para representar o grau máximo de abstração e simbolismo de que o ser humano é capaz. Turing, o genial criador desta ciência, sugeriu em 1951 que “em 50 anos, um computador será capaz de vencer o campeonato mundial de xadrez”. Ele errou por pouco: 1951+50=2001. Em 1997, Deep Blue, da IBM, venceu Gary Kasparov que era o campeão mundial. Que outra previsão – no mundo da computação – tem, teve ou terá tal grau de acerto? Já foi chamado de *drosóphila melanogaster* da IA, similar à importância da mosca doméstica na genética.

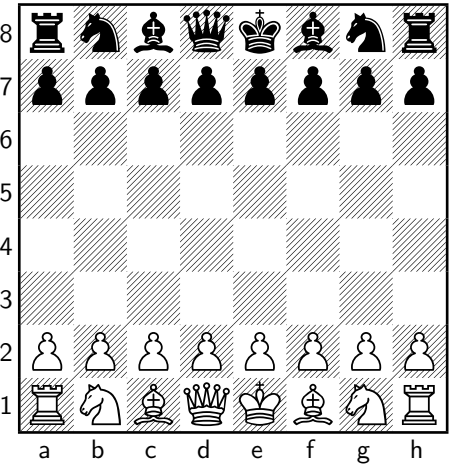
Abstraindo a computação, ainda assim o xadrez é um mundo maravilhoso. Aqui no Brasil, não temos grande tradição nele, mas em uma boa parte do mundo, as crianças aprendem a jogar e jogam durante toda sua vida escolar e por conseguinte muitos adultos seguem jogando vida afora. Isto, desde o século XII, que registra as primeiras manifestações do jogo, que por sinal é muito mais antigo do que isso.

Ele representa 2 exércitos que se opõem em um tabuleiro limitado (e pequeno) de 64 casas, e o único objetivo do jogo é vencer o exército inimigo. Esse fato é representado pela captura do rei oposto. Compõe o exército, as forças usuais: a infantaria, representada pelos peões em número de 8. As máquinas de guerra, as torres em número de 2. A igreja, personagem sempre influente, pelos seus 2 bispos. A cavalaria com 2 cavalos. Finalmente o casal real: a rainha ou dama, a peça mais poderosa no tabuleiro e o rei, limitado em movimentos, mas a peça em volta da qual todos operam.

Jogar xadrez é uma viagem intelectual indescritível, puro prazer. É superado pela construção de um programa de computador que jogue xadrez. Não é tarefa simples, mas deveria ser um investimento intelectual de todos quantos pretendam ser programadores plenos de computadores.

Como sempre, um programa grande e complexo precisa ser dividido em funções menores em tamanho e escopo. Uma parte importante em qualquer jogador é a análise e descoberta de todas as jogadas possíveis de uma determinada peça. Este exercício vai pedir que você programe uma pequena parte desta função.

Preliminares Antes de começar, é preciso se familiarizar com um tabuleiro armado.



Tal disposição é boa para humanos, mas computadores precisam outra coisa: uma matriz numérica de 8 x 8, com casa brancas e negras alternadas sendo que a casa embaixo, mais à direita, é branca. Por ser numérica, a matriz usa números para identificar as peças. Primeiro, a cor. Peças brancas são positivas e negras, negativas. Os valores: 1=peão, 2=torre, 3=cavalo, 4=bispo, 5=dama

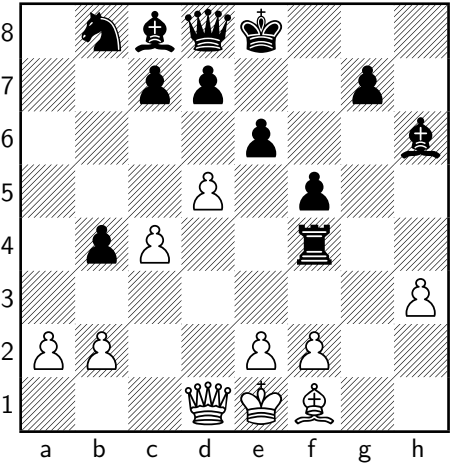
e 6=rei. Note que a célula linha 0, coluna 0, é a superior esquerda da matriz. Nesses termos, o tabuleiro acima corresponde a

	0	1	2	3	4	5	6	7
0	-2	-3	-4	-5	-6	-4	-3	-2
1	-1	-1	-1	-1	-1	-1	-1	-1
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	1
7	2	3	4	5	6	4	3	2

Você deve escrever um programa que vai receber um tabuleiro preenchido com uma situação de meio-jogo e também uma linha (0-7) e coluna (0-7). Você deve olhar o tabuleiro fornecido e ver qual é a peça que está lá. Identificada a peça e a sua cor, deve descobrir quantas casas essa peça pode ocupar num eventual próximo lance. Lembrando que as casas examinadas:

- se contiverem zero, podem ser ocupadas.
- se contiverem valor de sinal oposto ao da peça podem ser ocupadas, mas este valor implica parar na busca nessa direção (exceto o cavalo).
- se contiverem valor de mesmo sinal, não podem ser ocupadas e determinam final da busca.
- A casa não deve ser considerada se algum índice (linha ou coluna ou ambos) cair fora do tabuleiro.

Finalmente, não se assuste se aparecer um tabuleiro com alguma situação muito infrequente do ponto de vista enxadístico: o gerador usa e abusa de números aleatórios. Vejamos um exemplo



Se o endereço a pesquisar fosse 4,5 a resposta deveria ser 7. Se fosse 1,2 a resposta deveria ser 2. Se fosse 0,1 a resposta deveria ser 2.

Movimento do peão Se na casa inicial, pode adiantar 1 ou 2 casas. Se em outras casas, só pode andar 1 posição. Pode tomar em diagonal, à direita ou à esquerda. Não se implementa aqui a tomada *en passant*, pois ela depende da jogada anterior, que neste caso é desconhecida.

```
int peao(int x[8][8], int lin, int col){
    int casa = 0;
    if (x[lin][col]==1) {
        if (lin > 0 && x[lin-1][col]==0) {
            casa++;
        }
        if (lin==6 && x[4][col]==0 && x[5][col]==0){
            casa++;
        }
        if (lin>0 && col<7 && x[lin-1][col+1]<0) {
            casa++;
        }
        if (lin>0 && col>0 && x[lin-1][col-1]<0) {
            casa++;
        }
    }
    else {
        if (lin < 7 && x[lin+1][col]==0) {
            casa++;
        }
        if (lin==1 && x[2][col]==0 && x[3][col]==0){
            casa++;
        }
        if (lin<7 && col<7 && x[lin+1][col+1]<0) {
            casa++;
        }
        if (lin<7 && col>0 && x[lin+1][col-1]<0) {
            casa++;
        }
    }
    return casa;
}
```

Torre

```
int torre(int x[8][8], int lin, int col){
    int casa = 0;
    int nl = lin-1;
    int nc = col;
    while (nl>=0){
        if (x[nl][nc]*x[lin][col]==0) {
            casa++;
        }
        if (x[nl][nc]*x[lin][col]<0) {
            casa++; nl=0;
        }
        if (x[nl][nc]*x[lin][col]>0) {
            nl=0;
        }
        nl++;
        nl=lin+1;
        // ... nl<= 7...
        // ... nc<= 7...
        // ... nc>= 7...
        return casa;
    }
```

Cavalo

```
int cavalo(int x[8][8], int lin, int col){
    int i,nl,nc,casa = 0;
    int dlin[8]={-2,-1,1,2,2,1,-1,-2};
    int dcol[8]={1,2,2,1,-1,-2,-2,-1};
    for (i=0;i<8;i++){
        nl=lin+dlin[i];
        nc=col+dcol[i];
        if (nl<=7 && nl>=0 && nc<=7 && nc>=0){
            if (x[nl][nc]==0|x[nl][nc]*x[lin][col]<0){
                casa++;
            }
        }
    }
    return casa;
}
```

Bispo

```
int bispo(int x[8][8], int lin, int col){
    int nl,nc,casa=0;
    nl=lin-1;
    nc=col-1;
    while (nl>=0 && nc>=0){
        if (x[nl][nc]==0) {
            casa++;
        }
        if (x[nl][nc]*x[lin][col]<0){
            casa++; nl=0;
        }
        if (x[nl][nc]*x[lin][col]>0){
            nl=0;
        }
        nl--;
        nc--;
    }
    nl=lin+1;
    nc=col+1;
    // while nl>=0 && nc<=7...
    // while nl<=7 && nc<=7...
    // while nl<=7 && nc<=7...
    return casa;
}
```

Dama e Rei Fazendo os devidos ajustes aos controles de torre e bispo, é quase trivial. Ficam para você fazer, por falta de espaço.

Para você fazer

A seguir um tabuleiro em situação de meio jogo, seguido do endereço de 3 peças. Escreva um programa capaz de descobrir quantas jogadas cada peça individualizada pode realizar na jogada a seguir.

	0	1	2	3	4	5	6	7
0	-2	-5	-4	0	-6	0	-3	-3
1	-1	-1	0	0	2	0	0	0
2	0	0	0	-1	0	-1	-1	0
3	0	0	2	0	0	0	0	-1
4	0	0	1	0	0	0	0	1
5	1	0	0	0	1	0	0	0
6	0	0	0	1	0	1	1	0
7	0	3	4	5	6	0	0	0

Descubra quantas jogadas podem ser feitas pelas peças localizadas em: 1) 1 4, 2) 2 6 e 3) 3 2 .

Responda aqui:

1	2	3



Gerador de movimentos de xadrez

O xadrez tem um papel fundamental na Inteligência Artificial, e por conseguinte na ciência da computação. Ele é suficientemente compacto para poder ser manuseado dentro de um computador, mesmo um computador da década de 50, com poucas centenas de bytes de memória. Mas, é abrangente e complexo para representar o grau máximo de abstração e simbolismo de que o ser humano é capaz. Turing, o genial criador desta ciência, sugeriu em 1951 que “em 50 anos, um computador será capaz de vencer o campeão mundial de xadrez”. Ele errou por pouco: 1951+50=2001. Em 1997, Deep Blue, da IBM, venceu Gary Kasparov que era o campeão mundial. Que outra previsão – no mundo da computação – tem, teve ou terá tal grau de acerto? Já foi chamado de *drosóphila melanogaster* da IA, similar à importância da mosca doméstica na genética.

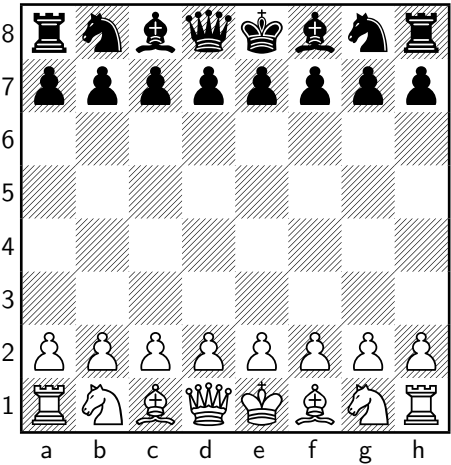
Abstraindo a computação, ainda assim o xadrez é um mundo maravilhoso. Aqui no Brasil, não temos grande tradição nele, mas em uma boa parte do mundo, as crianças aprendem a jogar e jogam durante toda sua vida escolar e por conseguinte muitos adultos seguem jogando vida afora. Isto, desde o século XII, que registra as primeiras manifestações do jogo, que por sinal é muito mais antigo do que isso.

Ele representa 2 exércitos que se opõem em um tabuleiro limitado (e pequeno) de 64 casas, e o único objetivo do jogo é vencer o exército inimigo. Esse fato é representado pela captura do rei oposto. Compõe o exército, as forças usuais: a infantaria, representada pelos peões em número de 8. As máquinas de guerra, as torres em número de 2. A igreja, personagem sempre influente, pelos seus 2 bispos. A cavalaria com 2 cavalos. Finalmente o casal real: a rainha ou dama, a peça mais poderosa no tabuleiro e o rei, limitado em movimentos, mas a peça em volta da qual todos operam.

Jogar xadrez é uma viagem intelectual indescritível, puro prazer. É superado pela construção de um programa de computador que jogue xadrez. Não é tarefa simples, mas deveria ser um investimento intelectual de todos quantos pretendam ser programadores plenos de computadores.

Como sempre, um programa grande e complexo precisa ser dividido em funções menores em tamanho e escopo. Uma parte importante em qualquer jogador é a análise e descoberta de todas as jogadas possíveis de uma determinada peça. Este exercício vai pedir que você programe uma pequena parte desta função.

Preliminares Antes de começar, é preciso se familiarizar com um tabuleiro armado.



Tal disposição é boa para humanos, mas computadores precisam outra coisa: uma matriz numérica de 8 x 8, com casas brancas e negras alternadas sendo que que a casa embaixo, mais à direita, é branca. Por ser numérica, a matriz usa números para identificar as peças. Primeiro, a cor. Peças brancas são positivas e negras, negativas. Os valores: 1=peão, 2=torre, 3=cavalo, 4=bispo, 5=dama

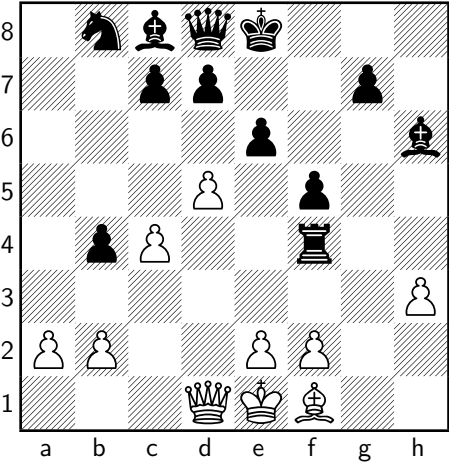
e 6=rei. Note que a célula linha 0, coluna 0, é a superior esquerda da matriz. Nesses termos, o tabuleiro acima corresponde a

	0	1	2	3	4	5	6	7
0	-2	-3	-4	-5	-6	-4	-3	-2
1	-1	-1	-1	-1	-1	-1	-1	-1
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	1
7	2	3	4	5	6	4	3	2

Você deve escrever um programa que vai receber um tabuleiro preenchido com uma situação de meio-jogo e também uma linha (0-7) e coluna (0-7). Você deve olhar o tabuleiro fornecido e ver qual é a peça que está lá. Identificada a peça e a sua cor, deve descobrir quantas casas essa peça pode ocupar num eventual próximo lance. Lembrando que as casas examinadas:

- se contiverem zero, podem ser ocupadas.
- se contiverem valor de sinal oposto ao da peça podem ser ocupadas, mas este valor implica parar na busca nessa direção (exceto o cavalo).
- se contiverem valor de mesmo sinal, não podem ser ocupadas e determinam final da busca.
- A casa não deve ser considerada se algum índice (linha ou coluna ou ambos) cair fora do tabuleiro.

Finalmente, não se assuste se aparecer um tabuleiro com alguma situação muito infrequente do ponto de vista enxadrístico: o gerador usa e abusa de números aleatórios. Vejamos um exemplo



Se o endereço a pesquisar fosse 4,5 a resposta deveria ser 7. Se fosse 1,2 a resposta deveria ser 2. Se fosse 0,1 a resposta deveria ser 2.

Movimento do peão Se na casa inicial, pode adiantar 1 ou 2 casas. Se em outras casas, só pode andar 1 posição. Pode tomar em diagonal, à direita ou à esquerda. Não se implementa aqui a tomada *en passant*, pois ela depende da jogada anterior, que neste caso é desconhecida.

```
import numpy as np
def peao(x,lin,col):
    casa=0
    if x[lin,col]==1: #branca: de baixo p/ cima
        if lin>0 and x[lin-1,col]==0:
            casa=casa+1
        if lin==6 and x[4,col]==0 and x[5,col]==0:
            casa=casa+1
        if lin>0 and col<7 and x[lin-1,col+1]<0:
            casa=casa+1
        if lin>0 and col>0 and x[lin-1,col-1]<0:
            casa=casa+1
    else: #preta: de cima para baixo
        if lin<7 and x[lin+1,col]==0:
            casa=casa+1
        if lin==1 and x[2,col]==0 and x[3,col]==0:
            casa=casa+1
        if lin<7 and col<7 and x[lin+1,col+1]<0:
            casa=casa+1
        if lin<7 and col>0 and x[lin+1,col-1]<0:
            casa=casa+1
    return casa
```

Torre

```
def torre(x,lin,col):
    casa=0
    nl=lin-1
    nc=col
    while nl>=0:
        if x[nl,nc]==0:
            casa=casa+1
        if x[nl,nc]*x[lin,col]<0:
            casa=casa+1
            nl=0
        if x[nl,nc]*x[lin,col]>0:
            nl=0
            nl=nl-1
    nl=lin+1
    # ... nl<=7 ...
    # ... nc<=7 ...
    # ... nc>=0 ...
```

Cavalo

```
def cavalo(x,lin,col):
    casa=0
    dlin=[-2,-1,1,2,2,1,-1,-2]
    dcol=[1,2,2,1,-1,-2,-2,-1]
    for i in range(8):
        nl=lin+dlin[i]
        nc=col+dcol[i]
        if nl<=7 and nl>=0 and nc<=7 and nc>=0:
            if x[nl,nc]==0 or x[nl,nc]*x[lin,col]<0:
                casa=casa+1
    return casa
```

Bispo

```
def bispo(x,lin,col):
    casa=0
    nl=lin-1
    nc=col-1
    while nl>=0 and nc>=0:
        if x[nl,nc]==0:
            casa=casa+1
        if x[nl,nc]*x[lin,col]<0:
            casa=casa+1
            nl=0
        if x[nl,nc]*x[lin,col]>0:
            nl=0
            nl=nl-1
            nc=nc-1
    nl=lin-1
    nc=col+1
    # ... while nl>=0 and nc<=7...
    # ... while nl<=7 and nc>=0...
    # ... while nl<=7 and nc<=7...
```

Dama e Rei Fazendo os devidos ajustes aos controles de torre e bispo, é quase trivial. Ficam para você fazer, por falta de espaço.

Para você fazer

A seguir um tabuleiro em situação de meio jogo, seguido do endereço de 3 peças. Escreva um programa capaz de descobrir quantas jogadas cada peça individualizada pode realizar na jogada a seguir.

	0	1	2	3	4	5	6	7
0	0	0	0	0	-6	0	2	0
1	-1	-1	0	-3	-1	-4	0	-1
2	0	-2	0	0	0	0	0	0
3	0	0	-4	3	0	0	-1	0
4	0	0	0	0	0	0	0	-3
5	0	0	0	1	0	0	1	0
6	0	0	0	0	1	1	0	1
7	0	3	0	0	6	0	-2	2

Descubra quantas jogadas podem ser feitas pelas peças localizadas em: 1) 1 3, 2) 1 1 e 3) 3 2 .

Responda aqui:

1	2	3



Prática de programação - C++

Esta folha pede para o aluno se ambientar com a programação. Os dados de entrada estão aqui listados, mas se o interessado quiser os MESMOS dados também estão publicados no local de sempre de onde podem ser capturados e colados (CTRL-C e CTRL-V) no seu código.

Exercícios

1. Defina no seu programa o vetor AA, contendo 120 inteiros, e criado através do comando

```
int AA[120]={
158,173,165,13,90,80,134,146,49,136,139,47,
52,42,144,119,128,69,129,115,19,194,105,54,
125,55,154,131,143,81,186,188,16,159,7,57,
183,32,137,4,15,5,123,195,133,25,113,130,
111,10,9,112,3,178,106,120,100,31,104,96,
181,56,200,182,110,176,45,67,193,8,169,14,
35,43,40,94,108,150,121,184,199,148,60,38,
83,117,101,58,88,164,142,175,29,145,61,75,
138,198,127,62,50,162,185,151,59,27,41,28,
82,21,135,72,34,92,20,86,2,109,1,12}
;
```

Agora escreva um programa em C++ que calcule e imprima a quantidade dos números acima que são maiores ou iguais a 21 e menores do que 175 . Responda no local correto.

2. Defina uma matriz quadrada de 12 × 12 usando os seguintes dados

```
int BB[12][12]= {
{40,33,95,8,118,88,68,60,4,34,112,65},
{6,40,72,21,57,85,79,77,59,63,56,74},
{64,98,118,12,41,62,33,100,100,37,82,12},
{22,98,19,52,58,89,17,42,65,82,64,92},
{85,87,26,9,10,70,9,49,118,89,88,112},
{105,21,76,71,11,32,109,29,93,79,61,113},
{92,9,115,46,107,76,103,107,24,91,98,60},
{57,19,45,2,25,54,43,60,1,58,83,95},
{45,58,21,14,18,117,12,83,34,30,83,56},
{69,49,82,58,34,114,82,67,102,106,7,83},
{31,88,54,93,45,75,70,20,100,52,81,112},
{101,71,117,17,94,10,112,28,74,28,33,40}}
;
```

escreva um programa que calcule a diferença entre a soma da diagonal principal (\) e a soma da diagonal secundária (/). Transcreva o valor no local correto.

3. Suponha uma lista das temperaturas máximas em Curitiba no último quadrimestre (120 dias), que está na variável CC.

```
float CC[120]={
11.2,18.8,15.9,13.3,10.3,10.4,14.1,11.7,17.4,15,
11.8,10.4,10.2,12.1,11.2,16.4,15.6,18.1,10.6,15.2,
15.8,11.8,19.6,14.4,19.5,17.4,16.6,13.8,19.4,16.9,
15.8,12.6,19.2,19.9,21,24.7,25,24.2,14,17.3,
11.1,22.9,17.8,19.5,10.1,22.9,17.6,12.1,22,12.2,
23.5,20.6,12.1,16.1,23.5,10.4,21.1,20.6,10.4,13.2,
25.2,29.8,22.1,17.2,22.7,20.4,22.3,24.9,29.6,20.4,
13,28.8,20.7,13.8,23.2,14.4,23.3,28.3,16.7,17.9,
27.9,26.9,11.6,15.2,22.5,17.7,29.9,25.3,29.7,24,
10.7,16.9,26.3,11.4,28.8,20.3,13,27,17.5,21.2,
15.4,35,13.6,18,21.7,28.1,23.2,27.6,33.3,14.5,
33.1,28.3,28.6,16.8,27.3,30.6,30,31,12.1,20.3}
;
```

Agora

- Descubra a amplitude térmica no quadrimestre
- Divida tal amplitude em 4 parcelas lineares
- Descubra quantos dias estão presentes na primeira parcela (a mais baixa dos 4). Considere a pertinência como igual ou maior e igual ou menor.

Responda no local correto.

4. A seguir, uma coleção de 195 inteiros, na forma de um vetor. Escreva um programa que ache a média daqueles números que são primos. Um número primo é aquele que só tem 2 divisores: ele próprio e a unidade. Pode haver números duplicados, e eles serão tratados normalmente, como se não fossem duplicados.

```
int DD[195]={
109,32,48,149,16,65,84,88,145,12,71,80,174,
70,51,33,83,107,36,33,48,107,107,162,108,46,
112,3,68,25,26,132,22,40,63,60,174,74,67,
81,135,20,37,25,150,30,81,109,85,128,170,23,
4,100,119,29,27,145,75,155,70,164,97,178,137,
131,175,55,96,74,40,67,24,88,98,111,118,58,
31,96,166,29,127,83,149,94,22,168,88,97,104,
104,73,69,57,30,176,57,170,100,43,23,79,67,
4,133,57,143,26,157,118,82,154,117,82,93,176,
62,7,160,94,12,106,18,47,29,41,156,109,108,
20,65,143,167,78,118,54,10,44,46,46,108,107,
96,82,18,94,20,52,136,1,97,179,174,95,176,
50,38,155,93,54,22,96,106,65,113,68,174,110,
148,126,19,127,160,36,153,143,163,13,144,9,92,
147,132,152,60,95,47,48,144,96,141,81,55,34}
;
```

5. A seguir uma matriz de 30 × 13 números inteiros menores do que 150. Descubra quantos dos números que aparecem na matriz são únicos (isto é, só ocorrem 1 vez).

```
int EE[30][13]={
{144,66,145,131,145,97,92,67,113,2,64,8,124},
{45,86,90,10,150,68,15,54,49,65,21,111,67},
{90,12,143,142,91,137,66,112,23,31,36,71,101},
{108,129,123,43,75,37,24,31,45,133,99,116,56},
{14,68,82,107,78,20,105,51,125,146,137,17,8},
{62,97,8,150,121,145,82,37,26,5,32,60,17},
{33,11,89,48,141,41,18,87,28,54,112,25,58},
{67,60,117,38,139,121,71,134,46,63,76,31,115},
{14,95,61,21,130,2,18,95,67,146,25,142,84},
{19,103,149,138,4,43,136,32,16,5,134,105,112},
{149,4,53,15,61,50,140,132,130,80,140,58,137},
{103,20,107,17,47,13,122,41,54,44,81,65,128},
{126,13,14,66,88,141,111,19,7,46,71,98,30},
{30,28,23,32,42,86,105,12,142,135,145,136,73},
{126,53,54,115,14,81,142,39,120,121,58,94,125},
{97,4,85,21,38,103,11,101,102,127,67,115,39},
{23,141,66,70,122,134,86,102,148,141,148,17,40},
{144,142,61,61,111,116,76,29,44,85,89,10,81},
{124,61,64,129,106,48,116,110,4,88,77,52,66},
{86,89,30,9,39,19,111,90,28,101,24,122,13},
{48,36,58,110,18,95,25,14,31,91,66,39,138},
{3,71,61,7,142,104,117,81,73,31,139,113,12},
{139,111,141,32,85,94,114,2,74,11,132,52,86},
{109,44,53,54,93,29,46,14,138,109,15,100,47},
{108,148,53,49,67,133,140,147,49,147,64,86,110},
{8,6,71,125,59,25,57,93,80,98,54,52,57},
{140,85,119,121,82,124,86,118,62,37,146,39,23},
{30,150,41,42,39,27,115,21,109,41,135,40,25},
{126,127,57,94,20,13,90,12,118,73,26,97,115},
{10,75,147,82,141,150,11,120,79,88,149,73,114}}
;
```

Exemplo No mesmo local está o arquivo exemp_1c.myd cujas respostas são: 69 -15 42 75.17 30.

🔗 Para você fazer

Se não quiser transcrever os dados acima, procure no lugar de costume o arquivo cujo nome é

C051001.myd

e utilize os dados lá gravados para resolver os 5 exercícios aqui propostos. Não se esqueça de transcrever no verso os programas utilizados (inéditos!). Se preferir, anexe **listagem** impressa dos mesmos. Transcreva as respostas:

1	2	3	4	5



Prática de programação

Esta folha pede para o aluno se ambientar com o acesso a um arquivo de dados que contém os dados de entrada para todos os exercícios da folha (são 10).

Uma dificuldade adicional é que não há um número fixo de dados por algoritmo. Então, vai-se trabalhar com este conceito. Um trecho que vai carregar em variáveis diferentes os dados de cada exercício pode ser (em python)

```
import numpy,math
e1=[]
e2=[]
...
ref=open("c:/apl2/F51lixo.myd","r")
lin=ref.readline()
while (0!=len(lin)):
    lin=lin.split()
    if lin[0]=="F01":
        exercicio = 1
        i = 1
        while i<len(lin) and exercicio == 1:
            e1.append(int(lin[i]))
            i=i+1
    if lin[0]=="F02":
        if exercicio == 1:
            exercicio = 2
            i = 1
            while i<len(lin) and exercicio == 2:
                e2.append(int(lin[i]))
                i=i+1
    if lin[0]=="F03":
        ...
    lin=ref.readline()
```

Ao final deste trecho e1 será uma lista que conterà os dados do exercício 1. e2 conterà os dados do exercício 2 e assim por diante.

Exercícios

1. Leia no arquivo citado os registros contendo "F01" nas posições 1 a 3 e imprima o maior - menor número ali existente. Este conceito é conhecido como amplitude do conjunto de números. Por exemplo, se o conjunto de números for

F01 55 66 33 12 68 918 2 3 55 3 121
F01 167 322 889 32

O maior número é 918 e o menor é 2, logo a amplitude é 916.

2. Leia no arquivo citado os registros contendo "F02" e imprima a quantidade de números existentes que é múltiplo de 7. Por exemplo, se o conjunto de números for

F02 931 852 900 105 542 459 594 769 744
F02 677 159 88 27 400 58

a quantidade de multiplos de 7 é 2 (são múltiplos o primeiro 931 e o quarto 105).

3. Leia no arquivo os registros contendo F03 e descubra quantos números são maiores do que seu vizinho da esquerda e menores do que seu vizinho da direita. Nesta contagem não entram os extremos, já que eles não têm um dos vizinhos. Por exemplo, se a entrada for

F03 494 450 820 973 146 194 825 568
F03 843 13 900 461 66 477 701

a resposta deveria ser 3, a saber: 820, 194 e 477.

4. Vamos definir o conceito de número gordo (aquele que tem mais do que 4 divisores) e número magro (aquele que tem menos do que 4 divisores). Note que o número que tem exatamente 4 divisores não é nem gordo nem magro. Leia no arquivo registros contendo F04 e descubra a quantidade de gordos menos a quantidade de magros no arquivo. Por exemplo, se os números forem

F04 171 737 482 649 183 760 675 935 39 916
F04 795 274 150 635 424 726 298 583 37 674

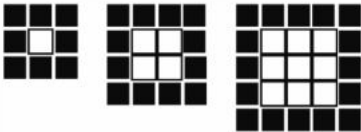
A resposta deve ser 8, já que os que tem mais do que 4 divisores são 171 760 675 935 916 795 150 424 726 e que apenas o 37 tem menos do que 4 divisores.

5. Agora você deve ler os registros identificados com F05 e reunir os valores nele constantes em uma matriz de 25 x 25 (25 linhas e 25 colunas). Depois disso deve calcular a diferença entre a soma da diagonal principal e a da soma da diagonal secundária ($|\sum principal - \sum secundaria|$). Note que $|x|$ é o valor absoluto de x. Por exemplo, numa matriz 3 x 3 como esta:

3 5 9
2 12 11
1 1 5

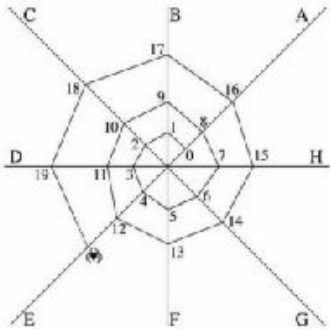
O número procurado deve ser 3+12+5 = 20 menos 9+12+1 = 22 ou $|20-22| = |-2| = 2$. A resposta procurada deve ser 2.

6. Para o próximo exercício suponha um piso formado por lajotas brancas e pretas, todas do mesmo tamanho. O primeiro caso é formado por um azulejo branco cercado de pretos. O segundo tem 4 brancos também cercados de pretos e assim sucessivamente conforme a figura.



No exercício, o único registro identificado por F06 contém a quantidade de azulejos pretos do piso. Você deve Calcular a quantidade de azulejos brancos necessários para a cobertura do piso usando esta regra. Por exemplo, se o número fornecido fosse 12, a resposta deveria ser 4.

7. Para o próximo exercício examine o trabalho de Roberta a aranha trabalhadeira:



No registro identificado por F07 você vai encontrar o nome (a letra) do fio onde a Roberta deve terminar seu serviço. Imediatamente ao lado, haverá a quantidade mínima de trechos (arestas) que a teia deve ter. Você precisa calcular a quantidade real de arestas que Roberta precisará produzir. Por exemplo, se o eixo pedido fosse C e a quantidade mínima de arestas fosse 13, a resposta real seria 18.

8. No exercício a seguir, leia os números acompanhados de F08. Disponha-os na maior matriz quadrada que os possa conter. Assim por exemplo se houver 110 números, a maior matriz quadrada é de ordem 10 ($10^2 = 100$ e $11^2 = 121$). Como $121 > 110$, a matriz quadrada escolhida é 10×10 . Neste caso pode haver necessidade de desprezar os últimos números lidos. Depois de fazer isso, ache a coluna da matriz que contém o maior número na matriz. Havendo empate, responda a maior coluna. Por exemplo, se os números lidos fossem

F08 74 32 73 84 17 63 35 31 93 97 61
F08 14 80 51 42 11 83 79

a matriz quadrada a ser formada seria 4 x 4:

74 32 73 84
17 63 35 31
93 97 61 14
80 51 42 11

E o maior número nela é 97, que ocupa a coluna 2. 2 é a resposta esperada.

9. Deve-se ler os números acompanhados de F09 e dispô-los segundo uma matriz cuja ordem é única e na qual o número de colunas é maior do que o número de linhas, sem desprezar nenhum número da entrada. Note que o número de linhas e o de colunas são ambos números primos. Ambos são maiores do que 4 e menores do que 30.

Disposta a matriz, achar o número da linha que tenha mais múltiplos de 3. Havendo empate, informar a maior linha.

Por exemplo, se os números forem:

F09 46 7 45 54 40 57 73 95 21 86 31 13 23 44 43
F09 33 78 34 71 52 28 87 84 17 59 77 80 62 99 5
F09 72 29 60 30 38

Como a quantidade de números é 35, as dimensões da matriz são 5 x 7. A quantidade de múltiplos de 3 é: 3, 1, 2, 2 e 4. Assim, a linha que mais múltiplos tem é a 4.

10. Definiremos **números bacanas** como sendo aqueles que estão no meio de dois primos gêmeos na sequência dos naturais. Assim, por exemplo, 12 é bacana, pois 11 e 13 são primos. E 11 e 13 são primos gêmeos, pois estão separados por 2 unidades na sequência dos naturais. Você vai ler uma sequência de números acompanhados de F10 e deve contar quantos bacanas há nessa sequência. Por exemplo, se fosse lido

F10 135 1950 149 1320 1372 72 1659
F10 436 1649 770 60 1608 833 1656

Nesta série há 5 bacanas: 60, 72, 1320, 1608 e 1950. Logo a resposta deveria ser 5.

Para Testar

Peça ao professor 3 arquivos de teste para verificar seu código e observar se ele atua corretamente. Os nomes dos arquivos são:

EXEMPL01.myd:
998 11 16 28 1719 7225 1745 11 2 17
EXEMPL02.myd:
965 12 19 25 121 6084 1265 4 5 15
EXEMPL03.myd:
978 13 11 30 2051 5625 1245 10 12 21

Para você fazer

Procure no site www.pkantek.com.br o arquivo cujo nome é

F051001.myd

e utilize os dados lá gravados para resolver os 10 exercícios aqui propostos. Transcreva as respostas:

1	2	3	4	5
6	7	8	9	10



- 1 - /

Prática de programação II

Esta folha pede para o aluno se ambientar com o acesso a um arquivo de dados que contém os dados de entrada para os exercícios da folha. Há um número fixo de dados no arquivo para cada exercício.

1. Suponha que você deve simular o comportamento do robot Emengardo que distribui café na sua empresa. O Emengardo é capaz de receber e obedecer a 5 tipos de ordens. Quando ele é ligado, todo dia às 08h00, ele se imagina no ponto 0,0 do barracão da empresa e direcionado ao Norte. Daí ele vai receber 150 ordens ao longo do dia. Você precisa descobrir qual a distância em linha reta desse ponto final ao ponto inicial, onde ele começou o dia. Cada ordem é composta de 2 números inteiros, o primeiro indica em que direção ele vai andar e o segundo indica quantos metros ele vai andar. O primeiro número pode ser 1=norte, 2=leste, 3=sul, 4=oeste e 5=a mesma em que ele já estava. Seja o exemplo:

```
Se ele começa em 0,0; após o comando:
1 10   ele vai para o ponto 10,0. Depois de:
2 20   ele vai para o ponto 10,20. Depois de:
3 30   ele vai para -20,20. Depois de:
5 10   ele vai para -30,20. Agora:
1 20   ele vai para -10,20.
```

E, se encontra à distância de 22.36m do início. Mas, para responder o exercício tome o chão do valor, ou seja 22m.
Leia no arquivo citado, 150 linhas, contendo cada uma delas uma ordem para o Emengardo e informe o chão da distância em que ele pára no final do dia.

2. Suponha uma máquina caça níquel com 3 bobinas girando de maneira independente. Quando a máquina pára, cada bobina pode parar em mimosa, banana, morango ou jaca, com igual probabilidade (por incrível que pareça, a máquina é honesta). O esquema de pagamentos dessa máquina para cada aposta de 3 moedas é:

mimosa	mimosa	mimosa	20 moedas
banana	banana	banana	15 moedas
morango	morango	morango	5 moedas
jaca	jaca	jaca	3 moedas
jaca	jaca	?	2 moedas
jaca	?	?	1 moeda

O símbolo ? indica que não importa o que apareceu. Havendo 2 prêmios devidos, apenas o maior é pago. Qualquer combinação não listada aqui, significa que a máquina engole o dinheiro apostado. Cada jogada que você faz custa 3 moedas. Supondo que você entra no cassino com 82 moedas para gastar nessa máquina e que ela vai apresentar nos próximos 200 lances as configurações dispostas no arquivo (1=mimosa, 2=banana, 3=morango e 4=jaca), a pergunta que você deve responder é quantas jogadas poderá fazer antes do seu dinheiro acabar ? (se é que ele vai acabar...) Por exemplo, se você entrar no cassino com 15 moedas e a máquina diz

```
1 2 3 - você perde 3 moedas e fica com 12
4 2 2 - você ganha 1 moeda e fica com 10
3 4 1 - você perde e fica com 7
2 2 3 - idem, 4
3 3 1 - idem, 1
3 2 1 - não há mais como jogar. A resposta é 4.
...
```

No exemplo acima, você conseguiu jogar 4 vezes, que é a resposta. Se seu dinheiro não acabar, responde 200.

3. Neste exercício você vai ler um string de 2000 caracteres, composto de 50 mensagens cada uma com 40 caracteres. Deve localizar a posição inicial da primeira ocorrência de

GHCUYUP

no conjunto. Lembre que a primeira posição corresponde a zero. Por exemplo, se o string original fosse

```
0123456789012345678901234567890123456789
-----
KJHSJLEIUIOPIOKLHJDGKJHDJUIIUSIKSLGTAH <=msg
AAAJKLJSLKJSLJSLKHDASLKJJKJSLKJSLHGGGKHDA <=msg
```

E o subconjunto a pesquisar fosse KHDA, a resposta deverá ser 56. Observação importante: se o subconjunto não existir, a resposta deverá ser 0.

4. Dois irmãos, Alfredo e Bernarda são muito companheiros e se gostam muito. Mas, recentemente receberam uma fortuna herdada de uma tia velha que morreu e deixou 2000 jóias de alto valor. O advogado contratado dividiu essa bolada em 200 partidas de 10 jóias cada partida. Cada jóia é identificada dentro da partida pelo seu valor em reais (um valor inteiro). Os 10 valores de cada partida já estão em ordem decrescente para facilitar a sua vida. O que você precisa decidir é das 200 partidas quantas podem ser igualmente divididas entre os 2 irmãos sem que haja nenhum real de diferença na divisão. Veja o exemplo:

```
200 50 30 2 2 2 2 1 1 1
50 20 20 20 5 5 5 5 5 5
1 1 1 1 1 1 1 1 1 1
800 10 10 10 10 10 10 10 10 10
```

Nestes exemplos, o primeiro não pode ser dividido, pois a soma do lote é ímpar. O segundo pode ser dividido e cada irmão vai receber jóias de 70 reais. O terceiro também e nele cada irmão recebe jóias de 5 reais. O último não pode ser dividido, não há como o lado que receber a jóia de 800 reais receber o mesmo que o outro irmão. A resposta final do problema exemplo é 2 partidas. Uma observação importante é que as jóias não podem ser reordenadas no lote. Por exemplo, um lote 4,3,2,1 deveria ser testado como 4|321 ou 43|21 ou 432|1 resultando impossível sua divisão, sem considerar a hipótese 41|32.

5. Existe um país numa região afastada do mundo chamado Borgúncia Sobeedesce. É um país muito pequeno mas coalhado de vilas e pequenas cidades. Você decidiu visitar integralmente o país, de bicicleta, mas como é muito esperto, estabeleceu que nunca vai subir, encarando apenas estradas e ruas que tenham descida ou no máximo que estejam no mesmo nível. No arquivo há 100 cidades, identificadas pela sua altitude em relação ao mar (números inteiros). Supondo que sua viagem vai começar na primeira cidade do arquivo e que você deve – em tese – seguir a ordem das cidades no arquivo, a pergunta a responder é quantas cidades conseguirá visitar, sem nunca subir na altitude. Por exemplo, se a lista de cidades fosse

```
854 758 900 758 601 200 444 85 100 1
```

você conseguiria visitar as cidades 1(854), 2(758), 4(758), 5(601), 6(200) 8(85) e 10(1). Totalizando 7 cidades.

6. Um grupo de amigos seus inventou um jogo que foi batizado de “poker catarinense”. Nele usa-se um baralho padrão com 13 cartas, de As até King, mas aqui o naipe não importa. Nesse jogo, dadas 5 cartas ao jogador, ele deve avaliar quantos pontos conseguiu, a partir das seguintes regras:

- 1. Se as cinco cartas estão em sequência a partir da carta x (ou seja, os valores das cartas são $x, x + 1, x + 2, x + 3$ e $x + 4$), a pontuação é $x + 200$ pontos. Por exemplo, se as cartas recebidas são 10, 9, 8, 11 e 12, a pontuação é 208 pontos.
- 2. Se há quatro cartas iguais a x (uma quadra, ou seja, os valores das cartas são x, x, x, x e y), a pontuação é $x + 180$ pontos. Por exemplo, se as cartas recebidas são 1, 1, 1, 1 e 10, a pontuação é 181 pontos.
- 3. Se há três cartas iguais a x e duas outras cartas iguais y (uma trinca e um par, ou seja, os valores das cartas são x, x, x, y e y), a pontuação é $x + 160$ pontos. Por exemplo, se as cartas recebidas são 4, 4, 4, 10 e 10, a pontuação é 164 pontos.

- 4. Se há três cartas iguais a x e duas outras cartas diferentes y e z (uma trinca, ou seja, os valores das cartas são x, x, x, y e z), a pontuação é $x + 140$ pontos. Por exemplo, se as cartas recebidas são 2, 2, 2, 3 e 13, a pontuação é 142 pontos.
- 5. Se há duas cartas iguais a x , duas outras cartas iguais a y ($x \neq y$) e uma outra carta distinta z (dois pares, ou seja, os valores das cartas são x, x, y, y e z), a pontuação é $3 \times x + 2 \times y + 20$ pontos, em que $x > y$. Por exemplo, se as cartas recebidas são 7, 7, 8, 12 e 12, a pontuação é 70 pontos.
- 6. Se há apenas duas cartas iguais x e as outras são todas distintas (um par, ou seja, os valores das cartas são x, x, y, z e t), a pontuação é x pontos. Por exemplo, se as cartas recebidas são 5, 8, 12, 13 e 13, a pontuação é 13 pontos.
- 7. Se todas as cartas são distintas, não há pontuação.

Para sua ajuda, as 5 cartas virão obrigatoriamente ordenadas em ordem crescente. Sua tarefa é ler no arquivo 200 jogadas e informar ao final do ciclo quantos pontos obteve nessas 200 rodadas de poker catarinense.

Exemplo No site da disciplina existem 3 arquivos de exemplo de nomes EXEMPLE1E.myd, EXEMPL2E.myd e EXEMPL3E.myd cujas execuções para estes enunciados resultaram os seguintes valores

```
66 KXDJIZYQRY 107 51 1219 22 33 9039
65 FCLXZJTJWJGL 20 55 1657 17 33 10203
72 COBHHCFWBM 212 48 0 17 22 10828
```

Observação: o primeiro número é a quantidade de dinheiro no cassino, a mensagem depois é o subconjunto a ser pesquisado e os 6 números seguintes são as respostas do exercício.

Para ler o arquivo

```
import numpy as np
def f051e():
    a=open('c:/apl2/nome.myd','r')
    f1=np.zeros((150,2))
    for i in range(150):
        xx=a.readline()
        f1[i]=[int(x) for x in xx.split()]
    f2=np.zeros((200,3))
    for i in range(200):
        xx=a.readline()
        f2[i]=[int(x) for x in xx.split()]
    f3=[]
    for i in range(50):
        xx=a.readline()
        xx=xx.rstrip()
        f3=f3+list(xx)
# ...
```

Para você fazer

Procure no site usual o arquivo cujo nome é

F51E0001.myd

e utilize os dados lá gravados para resolver os 6 exercícios aqui propostos. Transcreva as respostas:

1	2	3
4	5	6

Prática de programação - Python

Esta folha pede para o aluno se ambientar com a programação. Os dados de entrada estão aqui listados, mas se o interessado quiser os MESMOS dados também estão publicados no local de sempre de onde podem ser capturados e colados (CTRL-C e CTRL-V) no seu código.

Exercícios

1. Defina no seu programa o vetor AA, contendo 120 inteiros, e criado através do comando

```
aa=[
97,46,172,3,98,197,79,133,12,51,27,87,
23,84,49,73,24,103,105,29,113,167,153,48,
135,141,157,17,50,47,59,104,72,92,95,77,
32,69,22,162,136,142,42,66,15,114,75,200,
117,5,109,67,76,151,180,125,185,148,68,34,
91,52,166,174,168,115,178,121,40,43,65,62,
70,132,21,150,137,102,145,124,192,184,10,111,
74,85,53,182,55,35,83,36,116,189,7,45,
93,196,86,198,175,106,191,11,190,126,186,155,
1,2,89,195,161,156,154,144,78,112,169,152]
```

Agora escreva um programa em Python que calcule e imprima a quantidade dos números acima que são maiores ou iguais a 59 e menores do que 133 Responda no local correto.

2. Defina uma matriz quadrada de 12 × 12 usando os seguintes dados

```
import numpy as np
BB = np.array([
[33,113,36,10,110,73,26,35,100,49,74,11],
[52,81,81,70,4,112,100,107,59,16,60,51],
[65,75,31,13,83,40,23,64,48,26,113,107],
[38,94,63,19,45,24,31,99,68,59,39,71],
[23,51,85,73,100,43,76,3,15,85,29,116],
[115,44,87,89,98,47,28,49,38,95,64,66],
[23,46,57,119,82,119,55,3,79,12,21,70],
[9,34,50,58,3,10,108,95,50,12,40,113],
[40,65,51,94,55,4,94,114,38,67,35,42],
[66,113,118,74,26,33,96,15,119,40,24,14],
[69,19,113,46,108,5,21,55,76,37,11,90],
[108,100,33,63,3,87,28,100,10,89,90,32]]
)
```

escreva um programa que calcule a diferença entre a soma da diagonal principal (\) e a soma da diagonal secundária (/). Transcreva o valor no local correto.

3. Suponha uma lista das temperaturas máximas em Curitiba no último quadrimestre (120 dias), que está na variável CC.

```
CC=[
13.1,21.17.2,20,25,24.8,30.7,34.1,20,28.8,
13.4,29.5,27.2,27.6,20,29.1,27.4,29.5,20.5,23.8,
23.1,30.5,34.3,22.5,27.9,32.9,24.4,27.1,12.2,31.7,
13.5,22.6,27.8,27.6,20.4,22.5,29,15.1,16.5,13.1,
20.2,12.7,24.3,14.2,14.5,30,18.7,19.5,20,25.2,
16.9,30,17.6,26.1,10.5,23.8,28.5,24,12.3,29.5,
22.7,13,10.6,13.1,12.1,21.1,10.3,10.3,24.5,11,
14.7,19.1,14.8,12,20.5,23.7,21.8,15,12.6,19.1,
17.9,14.9,15.5,22,21.9,14.4,23.6,16.4,17.7,23.8,
16.8,15.3,19.8,11.5,10.9,13.5,16.5,16.8,10.5,13.8,
18.4,18.3,12.7,11.3,13.5,11.6,17.7,11.9,19.6,13.3,
16.1,12.9,16.6,18,13.1,18.1,10.9,12.8,18.6,12.7]
```

Agora

- Descubra a amplitude térmica no quadrimestre
- Divida tal amplitude em 4 parcelas lineares
- Descubra quantos dias estão presentes na primeira parcela (a mais baixa dos 4). Considere a pertinência como igual ou maior e igual ou menor.

Responda no local correto.

4. A seguir, uma coleção de 195 inteiros, na forma de um vetor. Escreva um programa que ache a média daqueles números que são primos. Um número primo é aquele que só tem 2 divisores: ele próprio e a unidade. Pode haver números duplicados, e eles serão tratados normalmente, como se não fossem duplicados.

```
DD=[
148,161,60,2,88,97,124,59,46,129,47,46,45,
39,25,43,169,4,27,14,129,176,119,172,85,111,
106,46,51,164,115,12,31,97,107,126,119,167,102,
148,65,136,159,119,124,82,60,112,42,171,20,137,
144,151,103,101,47,64,69,80,21,133,20,91,24,
166,18,9,89,166,28,49,1,178,75,60,100,164,
22,68,18,41,59,13,178,147,24,126,113,159,81,
81,169,72,47,101,12,75,154,90,47,102,142,2,
161,115,170,35,99,16,24,111,118,63,10,66,32,
118,52,111,177,80,114,113,151,117,20,15,16,72,
151,91,90,43,132,145,178,23,48,58,31,168,162,
143,156,75,102,48,147,109,92,133,98,161,180,120,
43,112,150,120,61,68,179,78,56,11,170,78,162,
15,175,25,171,139,101,35,72,132,51,42,88,11,
129,57,165,60,61,105,142,66,57,20,121,69,55]
```

5. A seguir uma matriz de 30 × 13 números inteiros menores do que 150. Descubra quantos dos números que aparecem na matriz são únicos (isto é, só ocorrem 1 vez).

```
import numpy as np
EE=np.array([
[12,51,44,90,124,20,96,128,134,103,128,83,7],
[112,21,92,124,103,52,104,134,85,117,92,59,91],
[32,138,147,45,134,119,81,80,118,53,145,120,59],
[64,105,41,15,14,101,1,85,132,1,91,28,68],
[134,103,31,118,56,95,120,13,49,125,122,88,73],
[121,106,39,34,85,103,12,50,75,118,119,93,37],
[40,69,46,122,120,129,103,108,85,95,47,110,135],
[133,75,143,98,139,124,57,77,88,123,107,21,65],
[137,113,118,100,133,65,88,32,15,35,128,116,124],
[130,22,1,108,32,30,101,39,43,5,11,87,133],
[148,146,20,20,118,14,67,105,12,115,50,74,65],
[47,26,9,97,83,7,117,137,71,16,150,127,29],
[41,35,91,21,68,106,73,5,140,124,3,28,95],
[63,76,18,122,62,76,23,41,129,134,139,39,70],
[73,44,39,82,97,149,99,40,140,48,28,65,85],
[70,57,20,34,150,11,89,12,70,66,23,86,68],
[103,4,60,74,51,13,102,94,119,116,9,139,121],
[72,93,136,94,92,100,12,56,26,100,121,76,35],
[80,44,3,12,92,14,14,131,119,105,122,120,25],
[138,126,103,146,14,48,83,96,16,17,66,116,19],
[43,146,79,61,116,85,109,124,87,91,10,71,10],
[58,87,11,61,6,16,113,118,54,142,74,148,37],
[132,32,89,73,78,112,105,76,64,57,31,16,125],
[73,79,121,47,56,117,69,110,119,46,117,112,17],
[32,148,81,120,38,2,27,30,93,45,91,61,22],
[108,3,141,112,43,89,113,52,54,143,148,44,39],
[117,115,6,128,122,4,19,44,74,95,20,60,63],
[78,65,103,143,35,135,77,72,5,75,56,78,136],
[36,53,84,8,49,116,73,27,128,86,37,65,65],
[100,111,58,12,79,75,59,132,139,12,128,12,22]]
)
```

Exemplo No mesmo local está o arquivo exemp_1c.myd cujas respostas são: 69 -15 42 75.17 30.

🔗 Para você fazer

Se não quiser transcrever os dados acima, procure no lugar de costume o arquivo cujo nome é

C051001.myd

e utilize os dados lá gravados para resolver os 5 exercícios aqui propostos. Não se esqueça de transcrever no verso os programas utilizados (inéditos!). Se preferir, anexe **listagem** impressa dos mesmos. Transcreva as respostas:

1	2	3	4	5



Prática de programação II

Nesta folha de exercícios vamos começar a desenvolver a habilidade de escrever código a partir de uma especificação escrita em linguagem natural (português). Veja-se o seguinte caso

ESCREVA UM PROGRAMA QUE LEIA 3 VALORES *a*, *b* e *c*. A SEGUIR, DEVE IMPRIMIR TODOS OS MÚLTIPLOS DE *a* QUE SEJAM MAIORES DO QUE *b* E MENORES DO QUE *c*. Aprenda a estudar a definição a partir dos exemplos apresentados, a saber:

5,11,22 -- deve imprimir: 15,20
8,60,100 -- deve imprimir: 64, 72, 80, 88, 96
1,1,6 -- deve imprimir 2,3,4,5
50,30,40 -- não deve imprimir

Uma possível implementação pode ser, em python:

```
a=int(input("informe a: "))
b=int(input("informe b: "))
c=int(input("informe c: "))
b=b+1
while b%a!=0:
    b=b+1
while b<c:
    print(b)
    b=b+a
```

Depois de uma tentativa de conclusão do código ele deve ser posto à prova com profunda desconfiança por parte do programador. Note que para nosso azar profissional, o acerto de um código jamais pode ser garantidamente afirmado (ao contrário do erro, que sempre pode ser certificado).

Para você fazer

1. Um micróbio (o *comequietus pythoniano*), tem um comportamento digno de nota. A cada *n* horas, ele joga uma moeda: se der cara ele morre. Se der coroa, ele se divide em *k* filhotes que crescem e depois de *n* horas voltam a jogar a moeda. Supondo uma infestação de *z* micróbios, ESCREVA UM PROGRAMA CAPAZ DE SIMULAR A SITUAÇÃO E INFORME EM QUANTOS DIAS SE CHEGARÁ À UMA POPULAÇÃO DE 1.000 INDIVÍDUOS , o que segundo as últimas pesquisas passa a ser mortal para o hospedeiro. Note que há alimentação abundante e nenhum outro risco de morte para o miasma, exceto aquele aqui descrito.

O programa deve ler os valores de *z*, *n* e *k*. Como existe uma variável aleatória (a cara e coroa), um único resultado não pode ser aceito como definitivo. Faça uma simulação de pelo menos 4 execuções e responda com a média das populações.

Para gerar um (pseudo-) aleatório em Python, pode fazer:

```
from random import randint
a=randint(0,1) # a=0:carra, a=1:coroa
```

Para verificar o “acerto” (relativo) do seu código eis algumas execuções: (todas com 1000 execuções)

z	n	k	média em dias
35	8	9	0.92
23	13	5	2.17
50	8	4	1.30
12	26	7	4.10

Se a contagem da população chegar a zero na avaliação, esta rodada deve ser desprezada e não deve entrar na média. Para sua execução, use os valores de *z*, *n* e *k* iguais a

10 19 6

e responda no local correto, a média de duração em dias do processo em pelo menos 4 execuções.

2. Você foi contratado como programador para gerar o código de uma máquina distribuidora de troco em moedas. Supondo que no país em tela existem moedas de 1 unidade, 50, 25, 10, 5 e 1 centavo, e imaginando que você deve privilegiar sempre as moedas de maior valor, ESCREVA UM PROGRAMA QUE RECEBA UM CERTO VALOR DE COMPRA E O VALOR DA NOTA QUE O FREGUÊS ENTREGOU PARA PAGAR A COMPRA. O SEU PROGRAMA DEVE INFORMAR QUANTAS E QUAIS MOEDAS DEVEM SER DEVOLVIDAS DE TROCO, SEMPRE TENTANDO OFERECER O MAIOR NÚMERO DAS MOEDAS DE MAIOR VALOR.

Esta estratégia de solução é chamada *gulosa* e muitas vezes ela é ótima (deixa de sê-lo, por exemplo em uma distribuição de moedas não convencional: por exemplo havendo moedas de 100, 50, 16, 5 e 1 centavos. Neste caso um troco de 20 centavos é calculado com 0,0,1,0,4 no total de 5 moedas enquanto um resultado melhor seria 4 moedas de 5 centavos).

Use a seguinte tabela de exemplos, para testar seu código:

compra	pag	troco	moedas
15.96	50	34.04	34 0 0 0 0 4 (t=38)
17.11	20	2.89	2 1 1 1 0 4 (t=9)
8.88	10	1.12	1 0 0 1 0 2 (t=4)

Para sua execução use os seguintes dados (compra e pagamento)

10 10

e informe a QUANTIDADE TOTAL DE MOEDAS DE TROCO no lugar correspondente.

3. Uma receita de salada primavera de frutas usa algumas frutas em uma determinada proporção. Para fazer uma certa quantidade fixa de salada, a receita sempre usa 100 gramas de maçã, 150 gramas de banana, 100 gramas de abacaxi, 100 gramas de mamão, 80 gramas de morango e 50 gramas de açúcar. Você deve ESCREVER UM PROGRAMA DE COMPUTADOR QUE RESPEITANDO INTEGRALMENTE AS PROPORÇÕES DE FRUTAS, LEIA AS QUANTIDADES EM GRAMAS, DISPONÍVEIS DE CADA INGREDIENTE (MAÇÃ, BANANA, ABACAXI, MAMÃO, MORANGO E AÇUCAR) E IMPRIMA A QUANTIDADE MÁXIMA EM QUILOGRAMAS DE SALADA QUE PODE SER PRODUZIDA. Por exemplo examine um caso particular. Suponha que as disponibilidades são: 1100 g de maçã, 1950 g de banana, 450 g de abacaxi, 200 g de mamão, 200 g de morango e 1500 g de açúcar. Chamando um M DULO de salada aquele ideal formado por 100 gramas de maçã, 150 de banana, 100 de abacaxi e mamão cada além de 80 de morango e 50 de açúcar (e pesando 580 gramas) nota-se que com os ingredientes do caso poderiam ser construídos 11 13 4.5 2 2.5 30 módulos se não houvesse restrição dos demais ingredientes. O menor destes valores é 2, já que sim, existem restrições. Então são 2 os módulos que podem ser produzidos gerando 2 × 580 = 1160 gramas ou 1.16 kilos, que é a resposta deste caso. Veja outros casos:

maçã	banana	abacaxi	mamão	morango	açucar	salada
1200	1750	300	350	250	1100	1.74
1300	1050	450	200	500	1500	1.16
1000	950	500	500	250	1100	1.8125

Para sua execução use as seguintes disponibilidades de maçã, banana, abacaxi, mamão, morango e açúcar, medidas em gramas:

1500 1450 200 350 450 1200

e indique no local adequado a quantidade máxima de salada de frutas primavera que pode ser feita em quilogramas.

4. Os egípcios antigos (há mais de 5.000 anos) já conheciam o conceito de π – não com esse nome, claro, mas como a constante que iguala o comprimento de uma circunferência ao seu raio. Como eles não dominavam a aritmética real que temos hoje, π era representado por uma fração ordinária, nomeadamente 22/7.

SEU PROBLEMA AQUI É RECEBER UM NÚMERO REAL QUALQUER COM BASTANTES CASAS DECIMAIS E LOCALIZAR QUAL A FRAÇÃO DE NÚMEROS INTEIROS QUE CHEGA MAIS PERTO DO VALOR DO NÚMERO REAL ORIGINAL.

Numa primeira abordagem ambos – numerador e denominador – devem ser menores do que 100 (ou seja, com 2 dígitos) e no segundo caso devem ser menores do que 1000 (ou seja, com 3 dígitos).

Veja-se alguns exemplos

número	n	d
3.141592654 c/ 2 dígitos	22	7
3.141592654 c/ 3 dígitos	355	113
2.718281828 c/ 2 dígitos	87	32
2.718281828 c/ 3 dígitos	878	323
5.67891234 c/ 2 dígitos	91	16
5.67891234 c/ 3 dígitos	619	109

Para o seu caso, considere o número real

7.32732458

Você deve fazer 2 processamentos: primeiro com números de 2 dígitos e depois com números de 3 dígitos. Sua resposta, portanto, é composta de 2 pares de números (numerador e denominador com até 2 dígitos e numerador e denominador com até 3 dígitos). Responda no local apropriado.

Informe as respostas obtidas aqui:

1	2	3	4



- 1 - /

Prática de programação III

Nesta folha de exercícios vamos começar a desenvolver a habilidade de escrever código a partir de uma especificação escrita em linguagem natural (português).

Para você fazer

1. Suponha que você deve escrever um programa para ajudar a um gerente de estacionamento no centro da cidade. O estabelecimento usa a seguinte regra de preços:

- até 30 minutos: 6 reais.
- De 31 minutos até 2 horas 8 reais.
- De 2h01min até 6 horas 14 reais e
- acima de 6 horas 40 reais.

O estacionamento fecha todo dias às 24h00, pelo que não é possível deixar um carro num dia e apanhá-lo no outro. Por essa razão, na entrada de dados não se anota o dia da entrada (e da saída). Seu programa deve ler 4 valores numéricos, a saber:

- hora da entrada 1
- minuto da entrada 52
- hora da saída 23
- minuto da saída 57

calcular e imprimir o valor a cobrar pelo estacionamento desse carro. Note que o segundo horário sempre será maior do que o primeiro, isso não precisa ser testado.

2. Você deve construir agora um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outro programa cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
(a+b).	certo
((3+5).	errado
((((1*6)))	certo
)33-4(.	errado
2*7.	certo

Note que a expressão lida sempre termina por um ponto (que é o carater indicador de final de expressão). Fora esse uso, o ponto nunca vai aparecer dentro da expressão em análise. Seu programa deverá ler a expressão, analisá-la e depois imprimir a mensagem certo ou a mensagem errado conforme for. Resolva os seguintes casos:

((2-7)*(5+2))/(2/1).
(((1*9)*(3+2)-(5/4))*(3+3).
(((3-9)*(2*8)))-(1-5))/(1-9).
(4*2)-(1-2).

3. Busca Este algoritmo é conhecido como casamento de cadeias e ele é muito importante em qualquer editor de texto (chamado geralmente via CTRL-F nos editores). É um algoritmo que pode ser bastante melhorado, pois na modalidade força bruta é ineficiente. Sua tarefa não chega a tanto, a eficiência deve vir depois da eficácia. Sua tarefa agora é escrever um programa de computador que leia 2 cadeias de caracteres: a primeira, denominada TEXTO contém uma cadeias ASCII normal, terminada por um ponto final (que não faz parte da cadeia em análise). A segunda cadeia, denominada PADRÃO também é terminada por um ponto e também este caracter não faz parte do texto para ser processado. O objetivo do seu programa é informar em que posições do TEXTO aparecem

ocorrências do PADRÃO. Acompanhe no exemplo

TEXTO	PADRÃO	ocorre em
ABRACADABRA.	BR.	2, 9
O GATO E O RATO: EIS UM FATO.	ATO.	4, 13, 26
OUVIRAM DO IPIRANGA.	RAM.	5
IVO VIU A UVA	OUVIU.	nada
abababacabab acanaabacaxi	aba	1 3 5 9 11 18

No exercício a seguir localize o padrão

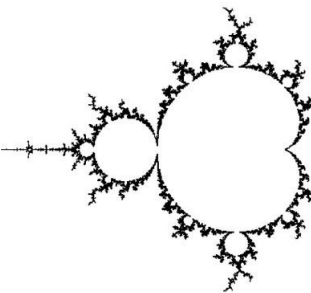
DD

no texto

HDHBCBDHHBDABBEGBEBCCGAGHDEH
HAADGDEEEHBAGCEFEFDBFHAGEBBCD

e informe até 3 ocorrências possíveis do padrão no texto.

4. Fractal de Mandelbroot O fractal de Mandelbroot é um objeto matemático fascinante. É um dos poucos que pode ser visto e ele tem o seguinte aspecto



Para sua construção, estuda-se o comportamento de um número complexo (a + bi). Relembrando um pouco da aritmética complexa, recorde-se que podem-se operar dois complexos usando a adição, subtração, multiplicação e divisão.

Todas elas são realizadas como se os números x e y (complexos) fossem polinômios na variável i. Assim temos: (3 + 4i) + (2 + i) = 5 + 5i
i + (2 + i) = 2 + 2i
2 + 3 = 5
(3 + I) × (2 – I) = 7 – I

Isto posto, o conjunto de mandelbroot (CM), é obtido através da análise de todos os pontos de um conjunto que está no plano de Argand Gauss. O conjunto de Mandelbrot é o conjunto de todos os números complexos c tais que após um certo número de iterações z = z² + c, z não tende para infinito. O valor inicial de z é zero.

Mediante uma análise matemática simples, determina-se se para cada ponto, se ele pertence ou não ao CM. Pintando-se de uma cor a uns e de outra a outros, obtém-se uma visualização do CM, que tem o aspecto visto acima.

Para cada ponto do plano complexo (ou melhor dizendo para cada ponto de uma grade sobre o plano complexo), far-se-á uma sequência de iterações. Se ao final dessa sequência, o resultado final estiver fora do círculo de raio = 2, o ponto está fora do CM. Se o resultado for menor que 2, o ponto está dentro do CM. Embora o cálculo correto fosse testar o ponto contra infinito, pode-se fazer o teste com 2. (Se o ponto ultrapassa 2, ele tenderá ao infinito).

A fórmula da iteração é:

zn+1 = zn² + c

, onde z0 = 0 + 0i e c é o ponto que está sendo testado.

É claro que para decidir se o ponto está ou não dentro do círculo de raio = 2, há que se ter um ponto de parada. Não se pode testar um ponto ao infinito. Se, nas primeiras 1000 iterações o ponto não saiu do círculo, não há garantia de que na vez 1001 ele não vá sair.

Portanto, a definição do CM é: O conjunto de pontos do plano complexo, que após iterados k vezes, segundo a fórmula zn+1 = zn² + c não caem fora do círculo de raio = 2.

Esta definição estaria ótima para pintar o fractal em 2 cores (pertence ou não pertence), mas há um truque que torna os fractais muito mais bonitos: a cor.

É hora de fazer uma distinção entre Fractal de Mandelbrot e Conjunto de Mandelbrot. O conjunto é apenas a reunião dos pontos que satisfazem a iteração acima. O fractal é um desenho formado por esse conjunto e pelos pontos externos, todos eles pintados de alguma maneira especificada.

Uso da Cor Para o exercício a seguir, você deve usar a seguinte tabela de cores fuga do raio = 2

em menos de 3 iterações	cor preto
em 4 iterações	cinza
em 5 iterações	marrom
em 6	vermelho
em 7	amarelo
em 8	rosa
em 9	verde claro
de 10 a 14 (inclusive)	roxo
de 15 a 19	laranja
de 20 a 29	verde escuro
de 30 a 49	azul claro
de 50 a 74	creme
de 75 a 99	azul escuro
não foge antes de 100	branco

Exemplos A seguir, alguns exemplos de pontos e suas cores

0.1655 0.268 -> 23 (verde escuro)
0.8733 0.4979 -> 4 (cinza)
0.5505 0.3246 -> 5 (marrom)
0.1091 0.1443 -> -1 (não sai do raio=2)

No exercício a seguir você deve descobrir de que cor deve ser pintado o pixel correspondente às seguintes coordenadas iniciais (obviamente seguindo a lista de cores acima)

0.3977 0.9936
0.8624 0.5437
0.6158 0.3899

Informe as respostas obtidas aqui:

1	2
preço	C/E
3	4
3 pos	cores

Observação importante: Neste exercício, além dos valores numéricos acima escritos, deve ser encaminhada ao professor, para correção, a listagem do código usado para chegar aos resultados. Informe também qual a linguagem e versão usada para tanto.



Treinamento básico em Algoritmos

Nos trechos de algoritmos a seguir, você deve seguir a lógica de cada trecho descobrindo qual o valor que é impresso ao final.

👉 Para você fazer

1. 1: funcao A11
2: $X \leftarrow 10$
3: enquanto $X \leq 17$
4: $X \leftarrow X + 2$
5: fim{enquanto}
6: escreva X
7: fimfuncao

2. 1: funcao A21
2: $X \leftarrow 10$
3: repita
4: $X \leftarrow X + 3$
5: até $X \geq 16$
6: escreva X
7: fimfuncao

3. 1: funcao A32
2: $X \leftarrow 0$
3: para Y de 4a19passo3
4: $X \leftarrow X + Y$
5: fim{para}
6: escreva X
7: fimfuncao

4. 1: funcao A12
2: $X \leftarrow 10$
3: $Y \leftarrow 7$
4: enquanto $X \leq 15$
5: $Y \leftarrow Y + 4$
6: $X \leftarrow X + 3$
7: fim{enquanto}
8: escreva Y
9: fimfuncao

5. 1: funcao A33
2: $X \leftarrow 2$
3: para Y de 5 a16passo3
4: $X \leftarrow X + Y + 5$
5: fim{para}
6: escreva X
7: fimfuncao

6. 1: funcao A13
2: $X \leftarrow 10$
3: $Y \leftarrow 5$
4: enquanto $X < 17$
5: $Y \leftarrow Y + X + 4$
6: $X \leftarrow X + 4$
7: fim{enquanto}
8: escreva Y
9: fimfuncao

7. 1: funcao A41
2: $X \leftarrow 6$
3: $Y \leftarrow 7$
4: se $X \geq 6$
5: $Y \leftarrow Y + X + 4$
6: senão
7: $Y \leftarrow Y - (X + 11)$
8: fim{se}
9: escreva Y
10: fimfuncao

8. 1: funcao A22
2: $X \leftarrow 5$
3: $Y \leftarrow 10$
4: repita
5: $Y \leftarrow Y + 3$
6: $X \leftarrow X + 4$
7: até $X > 17$
8: escreva Y
9: fimfuncao

9. 1: funcao A34
2: $X \leftarrow 4$
3: para Y de 17a6passo-3
4: se $(Y \bmod 2) = 0$
5: $X \leftarrow X + Y + 4$
6: senão
7: $X \leftarrow X + Y + 2$
8: fim{se}
9: fim{para}
10: escreva X
11: fimfuncao

10. 1: funcao A41
2: $X \leftarrow 4$
3: $Y \leftarrow 1$
4: se $X = 4 \wedge Y \geq 1$
5: $Y \leftarrow Y + X + 7$
6: senão
7: $Y \leftarrow Y - (X + 11)$
8: fim{se}
9: escreva Y
10: fimfuncao

11. 1: funcao A23
2: $X \leftarrow 7$
3: $Y \leftarrow 9$
4: repita
5: $Y \leftarrow Y + X + 4$
6: $X \leftarrow X + 4$
7: até $X \geq 16$
8: escreva Y
9: fimfuncao

12. 1: funcao A11
2: $X \leftarrow 8$
3: enquanto $X \leq 14$
4: $X \leftarrow X + 2$
5: fim{enquanto}
6: escreva X
7: fimfuncao

13. 1: funcao A32
2: $X \leftarrow 1$
3: para Y de 17a7passo-2
4: $X \leftarrow X + Y$
5: fim{para}
6: escreva X
7: fimfuncao

14. 1: funcao A21
2: $X \leftarrow 6$
3: repita
4: $X \leftarrow X + 3$
5: até $X > 17$
6: escreva X
7: fimfuncao

15. 1: funcao A41
2: $X \leftarrow 5$
3: $Y \leftarrow 2$
4: se $X < 4$
5: $Y \leftarrow Y + X + 7$
6: senão
7: $Y \leftarrow Y - (X + 10)$
8: fim{se}
9: escreva Y
10: fimfuncao

16. 1: funcao A13
2: $X \leftarrow 9$
3: $Y \leftarrow 9$
4: enquanto $X \leq 17$
5: $Y \leftarrow Y + X + 3$
6: $X \leftarrow X + 3$
7: fim{enquanto}
8: escreva Y
9: fimfuncao

17. 1: funcao A33
2: $X \leftarrow 2$
3: para Y de 4 a20passo2
4: $X \leftarrow X + Y + 8$
5: fim{para}
6: escreva X
7: fimfuncao

18. 1: funcao A11
2: $X \leftarrow 4$
3: enquanto $X \leq 15$
4: $X \leftarrow X + 3$
5: fim{enquanto}
6: escreva X
7: fimfuncao

19. 1: funcao A21
2: $X \leftarrow 20$
3: repita
4: $X \leftarrow X - 2$
5: até $X \leq 4$
6: escreva X
7: fimfuncao

20. 1: funcao A32
2: $X \leftarrow 0$
3: para Y de 6a13passo2
4: $X \leftarrow X + Y$
5: fim{para}
6: escreva X
7: fimfuncao

21. 1: funcao A12
2: $X \leftarrow 17$
3: $Y \leftarrow 10$
4: enquanto $X > 4$
5: $Y \leftarrow Y + 4$
6: $X \leftarrow X - 2$
7: fim{enquanto}
8: escreva Y
9: fimfuncao

22. 1: funcao A33
2: $X \leftarrow 3$
3: para Y de 5 a19passo3
4: $X \leftarrow X + Y + 5$
5: fim{para}
6: escreva X
7: fimfuncao

23. 1: funcao A13
2: $X \leftarrow 7$
3: $Y \leftarrow 8$
4: enquanto $X < 16$
5: $Y \leftarrow Y + X + 2$
6: $X \leftarrow X + 2$
7: fim{enquanto}
8: escreva Y
9: fimfuncao

24. 1: funcao A23
2: $X \leftarrow 19$
3: $Y \leftarrow 15$
4: repita
5: $Y \leftarrow Y + X + 3$
6: $X \leftarrow X - 2$
7: até $X \leq 10$
8: escreva Y
9: fimfuncao

25. 1: funcao A32
2: $X \leftarrow 0$
3: para Y de 4a17passo3
4: $X \leftarrow X + Y$
5: fim{para}
6: escreva X
7: fimfuncao

Respostas

1	2	3	4	5	soma →
6	7	8	9	10	soma →
11	12	13	14	15	soma →
16	17	18	19	20	soma →
21	22	23	24	25	soma →



1. Posições no xadrez

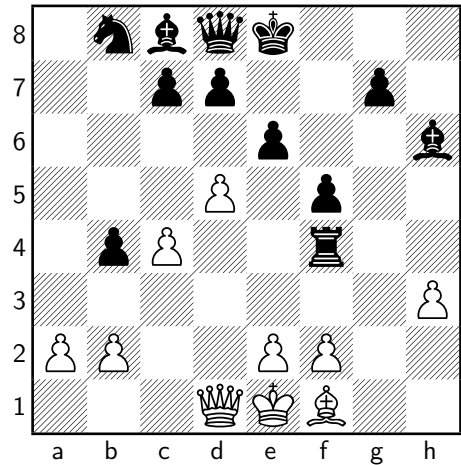
Imagine que você precisa examinar um tabuleiro de xadrez. Trata-se de uma matriz numérica de 8 × 8, com casa brancas e negras alternadas sendo que que a casa embaixo, mais à direita, é branca. Por ser numérica, a matriz usa números para identificar as peças. Primeiro, a cor. Peças brancas são positivas e negras, negativas. Os valores: 1=peão, 2=torre, 3=cavalo, 4=bispo, 5=dama e 6=rei. Você deve escrever um programa que vai receber um tabuleiro preenchido com uma situação de meio-jogo e também uma linha (1-8) e coluna (1-8). Note que a célula linha 1, coluna 1, é a superior esquerda da matriz. Você deve olhar o tabuleiro e se a posição informada estiver zerada, deve responder zero. Se ela contiver uma peça, deve examinar:

- quantas casas essa peça pode ocupar na próxima jogada
- quais peças contrárias podem ser tomadas na próxima jogada. Para este cálculo, some o valor da peça a ser tomada em valor absoluto.

Finalmente, não se assuste se aparecer um tabuleiro com alguma situação muito infrequente do ponto de vista enxadístico: o gerador usa e abusa de números aleatórios. Por exemplo: No tabuleiro

	1	2	3	4	5	6	7	8
1	0	-3	-4	-5	-6	0	0	0
2	0	0	-1	-1	0	0	-1	0
3	0	0	0	0	-1	0	0	-4
4	0	0	0	1	0	-1	0	0
5	0	-1	1	0	0	-2	0	0
6	0	0	0	0	0	0	0	1
7	1	1	0	0	1	1	0	0
8	0	0	0	5	6	4	0	0

que corresponde a esta posição



Se o endereço a pesquisar fosse 5,6 a resposta deveria ser 7,2
Se fosse 2,3 a resposta deveria ser 2,0
Se fosse 1,2 a resposta deveria ser 2,0

Seu programa deverá ler 50 tabuleiros no arquivo e somar os 2 valores acima para os 50 tabuleiros. A distribuição de dados no arquivo é a seguinte: A primeira linha tem L, C seguido de 6 números 999. Tanto L quanto C tem origem dos índices = 1. Na sequência, vem o tabuleiro (8 linhas de 8 colunas), positivos são as brancas, negativos, as negras e 1=peão, 2=torre, 3=cavalo, 4=bispo, 5=dama e 6=rei. Há um total de 450 linhas aqui.

2. Operações mandrake

A seguir, uma lista de operações matemáticas simples (identificadas por letras maiúsculas). Você precisa descobrir o que cada função faz e depois deve ler 50 expressões no arquivo e achar a soma das respostas a elas. Acompanhe os exemplos:

- 2 A 4 = 10
- 3 A 8 = 27
- 4 A 27 = 112
- 5 A 1 = 10

Neste exemplo, a função x A y multiplica x por y e soma x ao resultado. A seguir exemplos das 4 funções que você vai encontrar no seu arquivo

3 A 9 = 30	1 C 9 = 10	6 B 5 = 35
4 D 7 = 77	3 A 7 = 24	2 D 7 = 63
2 A 8 = 18	2 D 2 = 8	4 A 5 = 24
6 A 3 = 24	7 D 3 = 30	4 D 8 = 96
9 B 4 = 40	6 A 9 = 60	2 B 1 = 3
8 A 7 = 64	6 D 9 = 135	2 B 2 = 6
6 B 8 = 56	7 D 8 = 120	9 A 3 = 36

6 C 8 = 84	1 D 1 = 2	7 C 8 = 105
4 D 4 = 32	6 D 7 = 91	2 A 5 = 12
2 B 2 = 6	9 B 8 = 80	4 A 5 = 24
7 D 5 = 60	2 B 2 = 6	9 B 5 = 50
6 D 6 = 72	1 A 3 = 4	1 B 8 = 16
5 C 3 = 40	7 C 1 = 56	6 C 6 = 72
2 A 6 = 14	7 C 5 = 84	5 A 1 = 10
3 A 5 = 18	2 B 9 = 27	4 C 9 = 52

Leia no arquivo uma série de 50 operações e calcule a soma de todos os resultados.

3. Maior valor

Você deve terminar de preencher uma tabela 5 × 5 que já tem algumas casas preenchidas. Duas casas são consideradas vizinhas se têm um vértice ou um lado em comum. As regras que você precisa respeitar são:

- um casa só pode ser preenchida se alguma de suas casas vizinhas já contiver algum número.
- ao preencher uma casa, deve-se colocar a soma de todos os números que já constam em suas casas vizinhas.

A pergunta a responder é: qual é o maior número que é possível escrever na tabela ? Veja a seguir um exemplo completo:

0 0 0 0 0	214	211	207	167	333
0 3 1 0 0	428	3	1	36	130
0 0 0 0 0	885	454	8	27	67
0 0 0 4 0	2702	1363	14	4	98
0 0 2 0 0	8146	4081	2	118	220

cuja resposta é 8146.

Leia no arquivo um conjunto de 20 matrizes 5 × 5 e some os 20 maiores números (um para cada matriz). Note que há uma linha contendo 999 que deve ser ignorada (é apenas uma separação visual).

4. Números de Fibonacci

É conhecida a sequência de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, ... na qual $a_n = a_{n-1} + a_{n-2}$. No exercício a seguir, você deve ler no arquivo 20 quádruplas de números, a saber, a, b, lim_i e lim_s. Com os dois primeiros, deve construir uma sequência de fibonacci e a seguir deve contar quantos números dessa sequência estão incluídos entre os limites dados a seguir. Todos os números são inferiores a 110.000.000. Eis alguns exemplos

8	50	5619158	14834173	-->	2
9	39	1279693	80991085	-->	8
11	43	4235753	105964014	-->	6
43	46	9458750	35163265	-->	2
21	41	4920461	84942215	-->	6

👉 Para você fazer

Leia o seu arquivo publicado no lugar de sempre com o nome de

F0550001.myd

Estão no mesmo lugar os arquivos F055EXE1.MYD, F055EXE2.MYD e F055EXE3.MYD cujas respostas são:

142	56	2027	1076816	105
122	25	2072	1367135	119
90	42	1953	1896456	96

1.1 casas	1.2 peças	2. 50 operações	3. 20 maiores	4.fibo



- 1 - /

Introdução à Programação

OBI 2018: fase 1 - Album da Copa

Em ano de Copa do Mundo de Futebol, o álbum de figurinhas oficial é sempre um grande sucesso entre crianças e também entre adultos. Para quem não conhece, o álbum contém espaços numerados de 1 a N para colar as figurinhas; cada figurinha, também numerada de 1 a N, é uma pequena foto de um jogador de uma das seleções que jogará a Copa do Mundo. O objetivo é colar todas as figurinhas nos respectivos espaços no álbum, de modo a completar o álbum (ou seja, não deixar nenhum espaço sem a correspondente figurinha).

As figurinhas são vendidas em envelopes fechados, de forma que o comprador não sabe quais figurinhas está comprando, e pode ocorrer de comprar uma figurinha que ele já tenha colado no álbum.

Para ajudar os usuários, a empresa responsável pela venda do álbum e das figurinhas quer criar um aplicativo que permita gerenciar facilmente as figurinhas que faltam para completar o álbum e está solicitando a sua ajuda.

Dados o número total de espaços e figurinhas do álbum, e uma lista das figurinhas já compradas (que pode conter figurinhas repetidas), sua tarefa é determinar quantas figurinhas faltam para completar o álbum.

Entrada

A primeira linha contém um inteiro N indicando o número total de figurinhas e espaços no álbum. A segunda linha contém um inteiro M indicando o número de figurinhas já compradas. Cada uma das M linhas seguintes contém um número inteiro X indicando uma figurinha já comprada.

Saída

Seu programa deve produzir uma única linha contendo um inteiro, o número de figurinhas que falta para completar o álbum.

Restrições

1 ≤ N ≤ 100
1 ≤ M ≤ 300
1 ≤ X ≤ N

Exemplos

10 3 5 8 3
7

5 6 3 3 2 3 3 3
3

3 4 2 1 3 3
0

Para Fazer

Considere 3 execuções a saber:

20 23 13 14 16 10 20 2 17 1 18 14 14 10
14 7 3 17 8 12 5 16 5 3 11

26 30 15 6 14 21 12 20 1 1 1 26 7 15 16
2 5 9 3 3 8 18 8 15 21 6 10 23 14 26 1 23

15 13 2 2 3 4 4 2 11 7 5 6 9 15 3

OBI 2018: Fase 2 - Cápsulas O discípulo Fan Chi'ih retornou recentemente da China com algumas cápsulas mágicas, que são capazes de produzir moedas de ouro! Uma cápsula possui um certo ciclo de produção, que é um número C de dias. A cada C dias a cápsula produz uma nova moeda; a moeda é sempre produzida no último dia do ciclo. Fan Chi'ih vai ativar todas as cápsulas ao mesmo tempo e quer acumular uma fortuna de pelo menos F moedas. Ele precisa da sua ajuda para computar o número mínimo de dias para que as cápsulas produzam, no total, pelo menos F moedas. Na tabela abaixo, por exemplo, existem três cápsulas com ciclos de 3, 7 e 2 dias. Se Fan Chi'ih quiser acumular pelo menos 12 moedas, ele

vai ter que esperar pelo menos 14 dias.

cápsula	ciclo	dia													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3			1			1			1			1		
2	7							1							1
3	2		1		1		1		1		1		1		1

Entrada

A primeira linha da entrada contém dois inteiros N e F, indicando o número de cápsulas e o número de moedas que Fan Chi'ih quer produzir, respectivamente. A segunda linha contém N inteiros Ci, para 1 ≤ i ≤ N, representando os ciclos de cada cápsula.

Saída

Imprima um inteiro, representando o número mínimo de dias para que as cápsulas produzam, no total, pelo menos F moedas.

Restrições

1 ≤ N ≤ 10⁵
1 ≤ F ≤ 10⁹ e
1 ≤ Ci ≤ 10⁶

Em todos os casos de teste, a resposta é sempre menor ou igual a 10⁸ dias; Em todos os casos de teste, o número de moedas produzido, no total, após 10⁸ dias, é sempre menor ou igual a 10⁹.

Exemplos

3 12 3 7 2
14

10 100 17 13 20 10 12 16 10 13 13 10
130

Para fazer

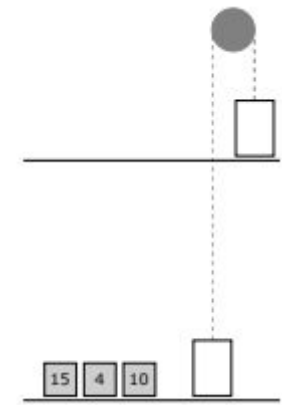
Considere 3 instâncias do problema, a saber:

7 12 5 2 1 4 5 4 4

5 13 5 3 2 6 6

8 8 3 2 3 1 6 1 6 3

OBI 2018: Fase J (júnior) - Pesos Uma fábrica instalou um elevador composto de duas cabines ligadas por uma roldana, como na figura.



Quando uma cabine sobe, a outra desce. No primeiro andar da fábrica existem algumas caixas de pesos diversos e precisamos levar todas as caixas para o segundo andar, usando o elevador. Apenas uma caixa pode ser colocada por vez dentro de uma cabine. Além disso, existe uma restrição de segurança importante: durante uma viagem do elevador, a diferença de peso entre as cabines pode ser no máximo de 8 unidades. De forma mais rigorosa, P − Q ≤ 8, onde P é o peso da cabine mais pesada e Q, o peso da cabine mais leve. O gerente da fábrica não está preocupado com o número de viagens que o elevador vai fazer. Ele apenas precisa saber se é possível ou não levar todas as caixas para o segundo andar. No exemplo da figura, podemos levar todas as três caixas usando a seguinte sequência de seis viagens do elevador:

1. Sobe a caixa de peso 4, desce a outra cabine vazia; (diferença de 4)
2. Sobe a caixa de peso 10, desce a caixa de peso 4; (diferença de 6)

3. Sobe a caixa de peso 15, desce a caixa de peso 10; (diferença de 5)
4. Sobe a caixa de peso 4, desce a outra cabine vazia; (diferença de 4)
5. Sobe a caixa de peso 10, desce a caixa de peso 4; (diferença de 6)
6. Sobe a caixa de peso 4, desce a outra cabine vazia. (diferença de 4)

Dados os pesos de N caixas no primeiro andar, em ordem crescente, seu programa deve determinar se é possível ou não levar todas as N caixas para o segundo andar.

Entrada

A primeira linha da entrada contém um inteiro N indicando o número de caixas. A segunda linha da entrada contém N inteiros representando os pesos das caixas, em ordem crescente.

Saída

Imprima uma linha na saída. A linha deve conter o caracter S caso seja possível, ou N caso não seja possível levar todas as caixas até o segundo andar da fábrica.

Restrições

1 ≤ N ≤ 10⁴
O peso das caixas está entre 1 e 10⁵ inclusive.

Exemplos

3
4 10 15
S

8
2 6 15 20 25 35 35 40
N

4
10 14 20 23
N

1
8
S

Para fazer

Resolva os 3 problemas a seguir

8 8 13 18 22 27 30 36 43

11 7 10 17 22 26 33 38 39 44 52 55

7 5 12 14 20 27 33 35

Para você fazer

1	2	3

Observação:
1. 3 valores em cada caixa acima.
2. Além das respostas textuais acima, o aluno deve entregar cópia do código usado para encontrar as respostas.



método D'Hondt O método D'Hondt, também conhecido como método dos quocientes ou método da média mais alta D'Hondt, é um método para alocar a distribuição de deputados e outros representantes eleitos na composição de órgãos de natureza colegial. O método tem o nome do jurista belga que o inventou, Victor D'Hondt.

O método é usado em países como Cabo Verde, Portugal, Timor-Leste, Argentina, Áustria, Bélgica, Dinamarca, Espanha, Finlândia, Islândia, Países Baixos, Paraguai, Uruguai e outros países.

O método consiste numa fórmula matemática, ou algoritmo, destinada a calcular a distribuição dos mandatos pelas listas concorrentes, em que cada mandato é sucessivamente alocado à lista cujo número total de votos dividido pelos números inteiros sucessivos, começando na unidade (isto é no número 1) seja maior. O processo de divisão prossegue até se esgotarem todos os mandatos e todas as possibilidades de aparecerem quocientes iguais aos quais ainda caiba um mandato. Em caso de igualdade em qualquer quociente, o mandato é atribuído à lista menos votada.

Utilizando representação matemática, o método pode ser representado pela fórmula $\frac{V}{s+1}$, onde V é o número total de votos apurado para a lista e s o número de lugares já colocados na lista em cada iteração do cálculo. O processo repete-se até todos os lugares estarem atribuídos.

O processo d'Hondt, baseia-se na atribuição dos mandatos por forma a que a proporcionalidade entre os votos recebidos pelas listas seja reproduzida, tanto quanto possível, na composição do órgão eleito, sem descurar a introdução de um fator de discriminação positiva em relação às minorias, permitindo-lhe uma representação que a simples divisão aritmética dos votos lhes negaria. Ao contrário do que acontece em órgãos colegiais compostos por simples maioria, nos compostos utilizando estes métodos, as minorias em geral conseguem representação razoável.

O método pode ser utilizado com o estabelecimento de limiares mínimos de eleição, sendo nesse caso eliminados de consideração os votos que recaiam nas listas cuja percentagem no total seja inferior ao mínimo estabelecido. Outra variante permite que o eleitor determine a sequência de atribuição dos mandatos dentro de cada lista, sendo os mandatos atribuídos à lista ocupados por ordem decrescente dos votos no candidato.

Exemplo prático (conversão dos votos em mandatos). O círculo eleitoral "X"tem direito a eleger 7 deputados e concorrem 4 partidos: A, B, C e D. Apurados os votos, a distribuição foi a seguinte: A - 12.000 votos; B - 7.500 votos; C - 4.500 votos; e D - 3.000 votos. Da aplicação do método d'Hondt resulta a seguinte série de quocientes:

Partido	A	B	C	D
Votos	12000	7500	4500	3000
	6000	3750	2250	1500
	4000	2500	1500	1000
	3000	1875	1125	750

No exemplo constante da tabela, os quocientes correspondentes a mandatos, assinaladas a negrito, levam à seguinte distribuição: Partido A - 3 deputados, correspondentes aos quocientes 12000 (1º eleito), 6000 (3º eleito) e 4000 (5º eleito). Note-se que apesar do quociente resultante da divisão por 4 ser 3000, igual aos votos obtidos pelo partido D, o mandato é atribuído ao menos votado, isto é ao Partido D, que assim elege o seu deputado.Partido B - 2 deputados, correspondentes aos quocientes 7500 (2º eleito) e 3750 (6º eleito). Partido C - 1 deputado, correspondente ao quociente 4500 (4º eleito). Partido D - 1 deputado, correspondente ao quociente 3000 (7º e último eleito), beneficiando da regra que em igualdade atribui o lugar à lista menos votada, arrebatando, por um só voto, o lugar ao partido A.

Sainte-Lague Na teoria do voto, o método de Sainte-Laguë é um procedimento para calcular coeficientes eleitorais e distribuir cadeiras em um câmara com voto proporcional, foi inventado pelo matemático francês André Sainte-Laguë. É um método similar ao método D'Hondt, a diferença são os divisores usados, o método de Sainte-Laguë favorece mais os partidos menores.

Este método é usado na Nova Zelândia, Noruega, Suécia, Dinamarca, Bósnia e Herzegovina, Letônia, Cosovo e Alemanha.

O método consiste em sucessivas divisões: a cada cadeira alocada, é calculado um coeficiente eleitoral dado pela fórmula: $\frac{V}{2s+1}$, onde: V é o número total de votos recebido pelo partido e s é número de cadeiras obtidas até o momento. Todos os partidos começam com $s = 0$ na primeira fase. A cada fase, é atribuída uma cadeira ao partido com maior coeficiente eleitoral. Exemplo:

Partido	A	B	C	D	E
Votos	340.000	280.000	160.000	60.000	15.000
Cadeira 1	340.000	280.000	160.000	60.000	15.000
Cadeira 2	113.333	280.000	160.000	60.000	15.000
Cadeira 3	113.333	93.333	160.000	60.000	15.000
Cadeira 4	113.333	93.333	53.333	60.000	15.000
Cadeira 5	68.000	93.333	53.333	60.000	15.000
Cadeira 6	68.000	56.000	53.333	60.000	15.000
Cadeira 7	48.571	56.000	53.333	60.000	15.000
cadeiras	3	2	1	1	0

No Brasil O método brasileiro é uma mistura dos métodos de Jeferson e Dhondt. No Brasil deve-se calcular o **Quociente eleitoral** que é a divisão do número de votos válidos pelo número de cadeiras, devidamente arredondado. (art. 106 do C.E.). Depois calcula-se o **Quociente Partidário** dividindo-se o total de votos do partido (ou coligação) pelo quociente eleitoral, desprezada a fração. (art. 107). Neste caso cada partido recebe o número de cadeiras dado pela parte inteira do Quociente Partidário. Se a soma de cadeiras dadas a partidos equivale ao total de cadeiras em disputa, está encerrada a apuração. Senão, essa diferença constitui as chamadas **sobras**. As vagas das

sobras serão alocadas ao partido que obtiver maior média resultado da divisão do Quociente Partidário (QP) pelo número de cadeiras + 1. (Partidos que não obtiveram nenhuma cadeira até aqui, estão fora do cálculo das médias e portanto não recebem mais nada). A cada alocação de nova cadeira as médias são recalculadas até não haver mais cadeiras a distribuir. Exemplo: Seja uma eleição brasileira onde estão em disputa 9 cadeiras e há 4 partidos (ou coligações). O total de votos é 6050 que dividido por 9 dá um *QE* (Quociente eleitoral) de $672,22 \approx 672$.

partido	A	B	C	D
votos	2250	1900	1350	550
QP	3,34	2,82	2,00	0,81
cadeiras	3	2	2	0
sobra 1	0,83	0,94	0,66	0,81≈0
cadeiras	3	3	2	0
sobra 2	0,83	0,70	0,66	0,81≈0
cadeiras	4	3	2	0

Note que se o partido D tivesse obtido 600 votos (ao invés de 550) ele teria um quociente de sobra de 0,89, maior que o do partido A na última sobra, mas ainda assim, não faria jus à vaga, já que sua votação foi inferior ao quociente eleitoral. Note também que $3.34 \div 4 = 0.83$ e que $2.82 \div 3 = 0.94$.

Alguns exemplos Para seu governo, a seguir alguns exemplos: Em cada instância, os números são: total de cadeiras em disputa, votação de corte (se igual a 0 não há corte), seguido pelas votações dos partidos, já em ordem decrescente de votação:

- a) 17 0 98527 91114 75296 46803 31337 10395 6242 4509
- b) 17 0 36521 6123 3005 2898 122
- c) 11 0 62124 61065 51864 36601 20912 11040 6606 1319
- d) 13 0 96402 93335 65820 21706 17499
- e) 11 17291 90063 87316 77511 76348 66365 64710 15165 11483

A seguir o resultado dos 3 métodos em cada caso

caso	dhondt	slague	Brasil
a)	5 5 4 2 1 0 0 0	5 4 4 2 2 0 0 0	5 5 4 2 1 0 0 0
b)	13 2 1 1 0	13 2 1 1 0	13 2 1 1 0
c)	3 3 2 2 1 0 0 0	3 3 2 2 1 0 0 0	3 3 3 2 0 0 0 0
d)	5 4 3 1 0	4 4 3 1 1	5 5 3 0 0
e)	2 2 2 2 2 1	2 2 2 2 2 1	***

*** No Brasil não há este tipo de corte.

🗳 Para você fazer

Usando a mesma especificação dos exemplos acima, calcule a distribuição de cadeiras em 3 eleições, usando um método em cada uma.

- Seja o país de GUARANIA cuja eleição usa o método de D'Hondt. A votação foi 12 4122 72535 68635 50106 35141 7474 5288 4122
Calcule a distribuição de cadeiras e indique quantas cadeiras obteve o SEGUNDO partido mais votado.
- Seja o país de FRIURA cuja eleição usa o método de Saint Lague. A votação foi 14 0 77395 58536 58135 20239 14525
Calcule a distribuição de cadeiras e indique quantas cadeiras obteve o TERCEIRO partido mais votado.
- Seja o país Brasil, na câmara municipal de DIZQUESIM cuja eleição usa o método do Código Eleitoral Brasileiro. A votação foi 15 0 99247 86866 49825 47914 32620 28848
Calcule a distribuição de cadeiras e indique quantas cadeiras obteve o QUARTO partido mais votado.

Responda aqui:

local	
GUARANIA	segundo=_____
FRIURA	terceiro=_____
DIZQUESIM	quarto=_____



Introdução à programação - Euler

Esta folha pede que o aluno desenvolva 10 programas baseados na coleção do projeto Euler devidamente adaptados. Para seu governo o projeto Euler mora em <http://projecteuler.net> e vale uma visita.

Project Euler
.net

AboutArchivesRecentNewsRegister

About Project Euler
What is Project Euler?

1. Múltiplos - baseado no Euler1 Se listarmos os números naturais abaixo de 30 que são múltiplos de 2 e 3 (considere 2 valores), obteremos como soma o valor 285 . Ache a soma de todos os números múltiplos de 2 e 3 abaixo de 2000 . Responda abaixo, no local indicado.

2. Números de Fibonacci pares - Euler2 Cada novo termo na sequência de Fibonacci é gerado pela adição dos dois números anteriores na sequência. Iniciando com 1 e 2, os primeiros 10 termos são

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Se considerarmos a sequência dos valores menores do que 2800000 , ache a soma dos termos que são divisíveis por 8 . Responda abaixo, no local indicado.

3. Maior fator primo - Euler3 Os fatores primos de 175376 são 2 97 113 Qual o maior fator primo de 12965835 ? Responda abaixo, no local indicado.

4. N-ésimo primo - Euler7 Se listarmos os 13 primeiros números primos podemos ver que o décimo terceiro número primo é 41 Qual o 3643 -ésimo número primo ? Responda abaixo, no local indicado.

5. Maior produto em uma série - Euler8 Ache o maior produto de 6 dígitos consecutivos no número de 600 dígitos:

811503503770867740434273415405316572597470666744245114449821
488852349595946004557904686038807383587968286628761776479392
355177529860999028566271924987687081969043283559281010627099
049204693105681711476572441913918151629471922538408839122061
306879344249793518295403712926819298836474886661709356517323
588955709174497977444559430436851897284800695575675388763726
690980369953971524182046687518532124886406071639089441286346
608655271052085324844458377969660224201365979484036155394126
920028655195411212469537748462032548113496414933929041835941
607621441543742339536414676591001186493913720641507835404724

? Responda abaixo, no local indicado.

6. Soma dos primos - Euler 10 A soma dos primeiros primos abaixo de 26 é 100 . Ache a soma dos primos abaixo de 35993 . Responda abaixo, no local indicado.

7. Número triangular com muitos divisores - Euler 12 A sequência de números triangulares é gerada somando números naturais. Assim o sétimo número triangular é 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. Os dez primeiros números triangulares são

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

A lista dos fatores (divisores) dos primeiros 7 números triangulares são:
1: 1
3: 1,3
6: 1,2,3,6
10: 1,2,5,10
15: 1,3,5,15
21: 1,3,7,21
28: 1,2,4,7,14,28

Pode-se ver que o 28 é o primeiro número triangular a ter mais do que 5 divisores. Qual é o primeiro número triangular que tem mais do que 11 divisores ? Responda abaixo, no local indicado.

8. Maior sequência interativa - Euler 14 A seguinte sequência interativa é definida para o conjunto dos inteiros positivos

$$n \rightarrow n/2 \text{ (se } n \text{ é par)}$$
$$n \rightarrow 3n + 1 \text{ (se } n \text{ é ímpar)}$$

Usando a regra acima e começando com 13, gera-se a seguinte sequência:

13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1

Pode-se ver que esta sequência começando em 13 e terminando em 1 tem 10 termos. Ainda que não tenha sido provado (Problema Collatz) é consenso que todos os números terminam em 1. Qual o primeiro número inicial, entre 1671 e 2036 que gera a sequência mais longa ? Responda abaixo, no local indicado.

9. Números amigos - Euler21 Definindo $d(n)$ como sendo a soma dos divisores próprios de n (números menores do que n que dividem n sem deixar resto). Se $d(a) = b$ e $d(b) = a$ onde $a \neq b$ então a e b são um par de amigos e cada um deles a e b são chamados números amigos. Por exemplo os divisores de 220 são 1,2,4,5,10,11,20,22,44,55 e 110. Então $d(220) = 284$ e os divisores próprios de 284 são 1,2,4,71 e 142 então $d(284) = 220$. Ache a soma de todos os números amigos abaixo de 50000 . Responda abaixo, no local indicado.

10. Maior produto em grade - Euler11 Na grade 10 x 10 abaixo

89	87	85	4	77	70	56	10	35	92
58	11	83	25	5	50	73	44	7	74
94	54	77	81	47	80	65	38	89	48
14	60	39	89	15	(99=89=85)	96	74		
1	3	46	67	66	27	20	42	52	90
78	75	25	28	4	60	23	80	24	86
28	92	91	82	20	46	91	32	58	33
46	8	55	6	55	30	2	70	35	62
89	6	94	57	78	12	54	98	91	88
52	74	33	3	29	74	62	27	55	44

A maior soma de 3 vizinhos nas 8 direções possíveis está marcada com (==) e vale 273. Atente que em casos extremos apenas 2 vizinhos (ou no limite um só) é que podem ser o maior valor. Ache a maior soma de até 3 vizinhos nas 8 direções na matriz 15 × 15 a seguir:

54	44	81	34	95	97	83	96	72	5	56	69	14	59	82
23	41	40	69	7	15	62	26	98	72	64	36	93	42	74
93	56	13	91	5	81	60	44	84	17	71	68	40	68	39
31	43	76	91	42	66	11	9	60	93	93	95	35	72	81
82	19	96	86	70	38	61	46	24	38	55	75	24	53	40
54	74	9	34	91	76	28	46	60	15	28	79	28	46	85
40	7	63	76	40	12	49	37	82	77	20	2	68	38	78
21	50	7	95	26	25	2	16	67	74	45	24	46	51	40
64	35	13	71	64	94	81	7	43	88	91	22	77	41	7
22	71	91	36	53	48	66	61	31	82	94	99	86	36	90
33	53	41	53	80	61	40	3	15	5	89	38	45	20	27
78	84	91	67	98	79	10	40	38	21	23	48	4	1	29
74	72	11	38	84	14	70	54	50	26	20	9	33	83	13
14	30	58	7	96	2	37	71	34	89	98	81	6	40	32
42	86	54	2	17	80	50	56	58	69	56	73	23	54	13

E responda no local adequado.

👉 Para você fazer

1	2	3	4	5
6	7	8	9	10

Grupos algébricos

Define-se um sistema matemático como um conjunto de 3 coisas:

1.

um conjunto de elementos
2.

uma ou mais operações que combinam esses elementos
3.

uma ou mais relações entre esses elementos

Um exemplo familiar é o conjunto $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ junto com a operação de adição e a relação de igualdade. Vai-se ver agora um sistema matemático chamado aritmética do relógio de 12 horas que lida com um conjunto finito de elementos. Imagine um relógio sem ponteiro de minutos e onde o 12 foi trocado por zero. Nosso conjunto agora é finito e igual a $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$. Vai-se definir a adição movendo o ponteiro das horas na direção horária. Por exemplo, para adicionar 3+8, primeiro vai-se mover o ponteiro para o 3, depois ele vai andar 8 posições. Onde ele parar, estará a resposta que é 3+8=11. Entretanto se a soma ultrapassar o limite, o resultado poderá ser inesperado, por exemplo: 7+7=2 ou 11+4=3. Não é possível construir uma tabela de adição para $\mathbb{Z}$ (ele é infinito), mas é possível uma tabela para o sistema do relógio-12.

+	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11	0
2	2	3	4	5	6	7	8	9	10	11	0	1
3	3	4	5	6	7	8	9	10	11	0	1	2
4	4	5	6	7	8	9	10	11	0	1	2	3
5	5	6	7	8	9	10	11	0	1	2	3	4
6	6	7	8	9	10	11	0	1	2	3	4	5
7	7	8	9	10	11	0	1	2	3	4	5	6
8	8	9	10	11	0	1	2	3	4	5	6	7
9	9	10	11	0	1	2	3	4	5	6	7	8
10	10	11	0	1	2	3	4	5	6	7	8	9
11	11	0	1	2	3	4	5	6	7	8	9	10

Algumas propriedades podem ser estudadas aqui. Por exemplo, a adição no relógio-12 é associativa? Para ser, precisa ser verdade que $a + (b + c) = (a + b) + c$. Não há como provar que isto é sempre verdade sem fazer todas as combinações possíveis, mas alguns exemplos devem nos convencer de que a propriedade é verdadeira. A subtração é facilmente implementada invertendo o sentido (de horário para anti-horário) na adição. Por exemplo $2 - 5 = 9$. Para conferir esta conta basta fazer $9 + 5 = 2$ e *voilà*. O elemento inverso $-a$ também existe e obedece à regra geral $a + (-a) = 0$ e $(-a) + a = 0$. Tente achar a tabela de inversos

a	0	1	2	3	4	5	6	7	8	9	10	11
-a												

A partir da tabela de inversos, pode-se definir formalmente a subtração como $a - b = a + (-b)$, como é a definição usual na matemática convencional. Números de relógio também podem ser multiplicados, usando a definição usual da multiplicação. Assim, 5×4 é igual a $4 + 4 + 4 + 4 + 4 = 8$, e daqui $5 \times 4 = 8$. Experimente, a seguir, preencher a tabela de multiplicação

×	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												

Agora a pergunta: Existe o inverso multiplicativo no relógio-12? A definição pede, por exemplo, para achar o inverso multiplicativo de 5 que se tenha $5 \times y = 1$ e olhando a tabela acima descobre-se que achar o i.m. de 2, tem-se $2 \times y = 1$. Mas, olhando-se a tabela da multiplicação, na linha do 2, só há números pares, donde se conclui que não existe y , ou seja o 2 não tem i.m. Como há pelo menos

um número diferente de zero que não tem i.m., a propriedade da multiplicação não tem inverso em relógio-12.

Sistemas modulares

Aqui, se generalizam as ideias do relógio. Por exemplo, no relógio-12, $0 + 7 = 7$. Entretanto, se somarmos 31 horas também vamos obter 7. Daqui, 7 e 31 tem o mesmo ponto de chegada. Então 7 e 31 são ditos *congruentes* módulo 12 e escreve-se $7 \equiv 31(mod\ 12)$. O sinal $\equiv$ indica *congruência*.

A razão pela qual 7 e 31 são congruentes é que sua diferença $31 - 7 = 24$ que é múltiplo de 12. Verifique agora se $21 \equiv 69(mod\ 12)$? Daqui a regra: Dois inteiros a e b são ditos congruentes módulo k e escreve-se $a \equiv b(mod\ k)$ se a diferença $a - b$ é divisível por k onde k é inteiro maior que 1. A ideia da congruência foi desenvolvida por Gauss em 1801, quando ele tinha 24 anos. Outra expressão do mesmo fenômeno é: $a \equiv b(mod\ k)$ sse obtem-se o mesmo resto quando a e b são divididos (divisão inteira) por k . A matemática modular pode ser desenvolvida do mesmo jeito que se fez no relógio-12. Por exemplo, a soma $(9 + 14)(mod\ 3)$, deve ser assim: Primeiro soma-se 9 + 14 e se obtém 23. Agora deve-se achar o menor não negativo congruente a 23 mod3. Faz-se isso dividindo 23 por 3. O resto é 2 e portanto $23 \equiv 2(mod\ 3)$ e então $(9 + 14) \equiv 2(mod\ 3)$. Ache $(18 + 27)(mod\ 6) \rightarrow 3$
 $(50 + 34)(mod\ 7) \rightarrow 0$
 $(8 \times 9)(mod\ 10) \rightarrow 2$
 $(12 \times 10)(mod\ 5) \rightarrow 0$

Tal como na álgebra, equações podem ser resolvidas em sistemas modulares. Uma equação é uma sentença tal como $(3 + x) \equiv 5(mod\ 7)$ que pode ou não ser verdade a depender do valor de x . Veja como resolver esta equação: O sistema $mod\ 7$ só admite os valores $\{0, 1, 2, 3, 4, 5, 6\}$. Substituindo estes valores em x verifica-se que a equação só é verdadeira para $x = 2$. Como 2 é uma solução, as demais obtem-se somando repetidamente 7. Assim, o conjunto de soluções para $(3 + x) \equiv 5(mod\ 7)$ é $\{2, 9, 16, 23, 30, 37, \dots\}$. Um outro exemplo: Resolver $6x \equiv 6(mod\ 6)$. Tentando os inteiros $\{0, 1, 2, 3, 4, 5\}$ verifica-se que qualquer inteiro pode ser usado como solução. Como a expressão é verdadeira para qualquer valor da variável, a variável é chamada identidade.

Sistemas finitos

Vai-se introduzir um conjunto finito de elementos $\{a, b, c, d\}$ e uma operação de símbolo $\star$ usada para operar com dois elementos do conjunto. A solução de $c \star d$ deve ser procurada em uma tabela, olhando a intersecção da linha c com a coluna d . Estamos interessados nas propriedades associativa, fechamento, comutativa, identidade e inversa, que já estudamos.

propriedade	definição
fechamento	$a \star b$ está no conjunto
comutativa	$a \star b = b \star a$
associativa	$(a \star b) \star c = a \star (b \star c)$
identidade	$a \star ei = a$ e $ei \star a = a$
inverso	$a \star x = ei$ e $x \star a = ei$

Outra propriedade é a distributiva que, ao contrário das que já vimos (que envolvem uma única operação), agora envolve uma segunda operação, digamos $\diamond$. Agora, para os elementos a, b e c , diz-se que $\star$ é distributiva em relação a $\diamond$ se $a \star (b \diamond c) = (a \star b) \diamond (a \star c)$.

Grupos

Um sistema matemático que satisfaz fechamento, associativa, identidade e inversa é um grupo. Por exemplo, os inteiros com a adição é um grupo. Os grupos foram estudados por Evariste Galois (1811-1832) que se concentrou em suas propriedades na solução de equações e formam importante aspecto da álgebra abstrata moderna.

Galois morreu antes de completar 21 anos, num duelo contra um capitão do exército francês. Na véspera, pressentindo que ele não ia sobreviver escreveu uma carta a um amigo, resumindo todos os resultados que ele já havia encontrado. Na carta le-se *...morri vítima de uma infame namorada e de dois idiotas que ela envolveu. Minha vida termina em consequência de miserável calúnia.*

Porque tenho que morrer por uma coisa tão insignificante e desprezível?... não tenho tempo, não tenho tempo...

Voltando à matemática, veja-se que o conjunto dos inteiros ímpares $\{\dots, -3, -1, 1, 3, 5, \dots\}$ com a multiplicação não formam grupo, pois não tem a propriedade inversa. Já o conjunto $\{-1, 1\}$ com a multiplicação, formam um grupo. Grupos não se aplicam só a números. As simetrias de um quadrado com a operação de rotação formam um grupo ([Mil90], pág, 305).

Onde se usa isso ?

Existem muitas aplicações de aritmética modular. No cálculo de dígitos verificadores, que é o básico em qualquer código computacional, na criptografia RSA, no código QR no qual todo o desenvolvimento se faz sobre um grupo de Galois, na geração de números pseudo-aleatórios, etc etc. Sem contar que 100% da aritmética inteira dentro de um computador é *mod* alguma coisa. Nossa sorte é que essa *alguma coisa* é sempre bem grande. Também é a base de quaisquer operações envolvendo datas no nosso calendário.

🔗 Para você fazer

1. Ache o valor das expressões a seguir

$(17 + 13)(mod\ 7)$

$(12 + 31)(mod\ 3)$

2. Ache os 3 primeiros menores valores positivos da equação em x

$x + 6 \equiv 19(mod\ 12)$

$x + 3 \equiv 23(mod\ 7)$

3. A fórmula seguinte é usada para achar o dia da semana do dia 1º de janeiro de um determinado ano. y é o ano (que deve ser maior que 1582, quando nosso calendário Gregoriano começou)

$a = y + [(y - 1)/4] - [(y - 1)/100] + [(y - 1)/400]$

onde $[x]$ é o chão de x (o inteiro menor ou igual a x).

Depois de achar a deve-se achar b tal que

$a \equiv b(mod\ 7)$

Agora b representa o dia da semana de 1º de janeiro. (b=0 é domingo, b=1 é segunda e b=6 é sábado).

Ache o dia da semana do primeiro dia do ano dos dois anos seguintes

2053

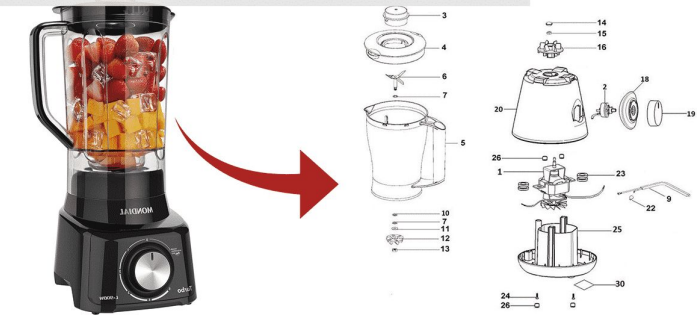
2097

1.1	1.2	2.1	2.2	3.1	3.2



Engenharia Reversa

ENGENHARIA REVERSA



Engenharia Reversa de algoritmos

Se a programação fosse mais engenharia e menos arte, talvez não precisássemos tanto usar os dons de Sherlock Holmes para corrigir programas. Infelizmente (ou será felizmente ??) tal não ocorre, e um programador profissional gasta mais de 50% do seu tempo procurando e corrigindo erros em programas.

É tarefa difícil, sutil, complexa e delicada. s vezes um ponto fora do lugar põe a perder um imenso projeto. É tão importante a tarefa que se diz que bom programador não é aquele de programa bem, mas aquele que depura bem. A propósito, depurar é **identificar, localizar e corrigir erros, de maneira continuada e integrada para produzir software de qualidade.**

O exercício de hoje, visa estimular esta habilidade. Você vai receber alguns códigos e alguns resultados, e deverá inferir quais as inicializações originais das variáveis em cada código.

Por exemplo, no algoritmo

```
1: função EXEMPLO
2: A ← _____
3: B ← 2
4: C ← 3
5: enquanto A < 10
6:   se (B mod 3) = 0
7:     A ← A + 1
8:   senão
9:     A ← A + 2
10:  fim{se}
11:  se (C mod 4) = 0
12:    B ← B + 2
13:  senão
14:    A ← A + 1
15:  fim{se}
16:  imprima A, B, C
17: fim{enquanto}
```

Analisando as saídas, que são:

4 2 3
7 2 3
10 2 3

Percebe-se que a inicialização original do A foi 1. A resposta deste exemplo portanto, é 1.



Para você fazer

A seguir, diversos algoritmos. Em cada um deles, você deve dizer qual o valor inicial da variável A.

Exercício 1

```
1: função A1
2: A ← _____
3: B ← 9
4: C ← 7
5: enquanto (A + B) > 4
6:   se A > C
7:     A ← A - 1
8:   senão
9:     A ← A + 1
```

```
10:  fim{se}
11:  B ← B - 2
12:  imprima A, B, C
13: fim{enquanto}

Esta execução deu como resultado
6 7 7
7 5 7
8 3 7
7 1 7
8 -1 7
7 -3 7
```

Exercício 2

```
1: função A2
2: A ← _____
3: B ← 11
4: C ← 11
5: enquanto (C ≠ 0) ∧ (C ≠ 1)
6:   se (B mod 5) ≠ 0
7:     B ← B - 1
8:   fim{se}
9:   se (A mod 3) = 0
10:    A ← A + 1
11:  fim{se}
12:  se (A > B)
13:    A ← A + 2
14:  fim{se}
15:  C ← C - 1
16:  imprima A, B, C
17: fim{enquanto}
```

Esta execução deu como resultado

13 10 10
15 10 9
18 10 8
21 10 7
24 10 6
27 10 5
30 10 4
33 10 3
36 10 2
39 10 1

Exercício 3

```
1: função A3
2: A ← _____
3: B ← 3
4: C ← 5
5: repita
6:   A ← A + 1
7:   B ← B + 1
8:   C ← C + 1
9: enquanto C > 10
10:  C ← C - 2
11: fim{enquanto}
12: se (C ≠ 3) ∨ (B ≠ 2)
13:   A ← A + 2
14: fim{se}
15: imprima A, B, C
16: até (A + B + C) > 30
```

Esta execução deu como resultado

7 4 6
10 5 7
13 6 8

16 7 9

Exercício 4

```
1: função A4
2: A ← _____
3: B ← 15
4: C ← 13
5: enquanto (A>10) ∨ (B>10) ∨ (C>10)
6:   se A > B
7:     C ← C - 1
8:   fim{se}
9:   se (B mod 3) = 1
10:    A ← A - 1
11:   fim{se}
12:   se C > B
13:     B ← B - 3
14:   fim{se}
15:   imprima A, B, C
16:   A ← A - 1
17:   B ← B - 1
18:   C ← C - 2
19: fim{enquanto}
```

Esta execução deu como resultado

13 15 13
12 14 11
10 13 9
9 12 7
8 11 5

Respostas

Escreva aqui os valores presumidos para as 4 variáveis “A” de cada um dos 4 algoritmos

1	2	3	4

1

Equações e inequações

Álgebra é uma generalização da aritmética. Ambas trabalham com números, mas a álgebra usa também *variáveis*, letras que representam números. Isto permite que problemas mais complexos sejam resolvidos. A origem da álgebra remonta aos babilônios, por volta de 2000 AEC. Eles já resolviam problemas envolvendo x^2 . Multiplicação de positivo e negativo já aparecia por volta de 300 AEC.

Equações lineares

Uma expressão algébrica é feita de 3 coisas: números, variáveis e sinais de operação. Uma variável é uma letra que representa um número. O produto de um número e uma variável (como $8x$) é chamado **termo**. O número que multiplica a variável é chamado **coeficiente**. Uma frase que afirma que duas expressões são iguais é uma **equação**. Por exemplo, $y + 8 = 15$, $5m = 30$, $8p - 2 = 3p + 8$ e $t = 3$. Uma equação é resolvida quando se descobrem os valores das variáveis que tornam a equação verdadeira. Estes valores são as **soluções** da equação. Por exemplo, o número 3 é a solução da equação $8x + 7 = 31$, já que ao substituir a variável x por 3, a equação acima se torna verdadeira. A equação acima é **linear**. Isto se refere ao fato de que o maior expoente da variável é 1. (embora não seja escrito: $x = x^1$). Formalmente, uma equação na variável x é linear quando pode ser escrita como $ax + b = c$, onde $a, b, c \in \mathbb{R}$ e $a \neq 0$. Equações lineares podem ser resolvidas mediante uma sequência de etapas que terminam quando em um lado da igualdade houver apenas a variável e do outro apenas um número. Estas etapas abraçam a idéia de que se fizermos A MESMA operação dos dois lados da igualdade, a mesma não se altera (segue sendo uma igualdade). Por exemplo, suponhamos resolver a equação $x + 5 = 12$. Pretende-se mudar o lado $x + 5$ para apenas x . Para isso, deve-se somar -5 a $x + 5$ já que $x + 5 + (-5)$ se converte em x . Mas, ao fazer isso e para manter a igualdade, há que se somar -5 no outro lado da igualdade também. Este, fica $12 + (-5)$ que é igual a 7. O resultado final é $x = 7$ e está terminada a resolução da equação. Isto pode ser verificado substituindo x por 7 na equação original e questionando a veracidade do resultado $7 = 7$ é verdadeiro! As duas regras a seguir podem ser usadas na solução

- O mesmo termo pode ser somado ou subtraído em ambos os lados da equação.
- O mesmo termo (**não zero**) pode multiplicar ou dividir ambos lados da equação.

A frase termo não zero é necessária já que multiplicar qualquer equação por 0, leva a $0 = 0$ que é verdade, mas não ajuda a achar a solução. Mais ainda, a divisão por 0 é sempre proibida.

Durante a solução, apenas termos que tenham a mesma variável podem ser combinados. Assim, por exemplo, $6m - 2m$ pode ser resolvido dando $4m$, mas $5k - 4t$ não pode ser resolvido. Deve-se agora, resolver alguns exemplos

$$9k + 1 = 12k + 5$$

$$4r + 5r - 3 + 8 - 3r - 5r = 12 + 8$$

$$3y + 4 - 2y - 7 = 4y + 4$$

$$4(k - 3) - k = k - 6$$

Uma pegadinha

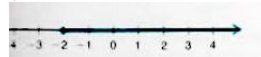
Vamos calcular $1 + 2 + 4 + 8 + 16 + 32 + \dots$. Fazendo-a igual a N fica $N = 1 + 2 + 4 + 8 + 16 + 32 + \dots$, reagrupando $N = 1 + (2 + 4 + 8 + 16 + 32 + \dots)$ ou $N = 1 + 2(1 + 2 + 4 + 8 + 16 + \dots)$ ou $N = 1 + 2N$ ou $N = -1$ e daí $-1 = 1 + 2 + 4 + 8 + 16 + 32 + \dots$ que obviamente é uma besteira. Qual a pegadinha? É simples: $1 + 2 + 4 + 8 + 16 + 32 + \dots$ NÃO é um número real, e portanto fazer $N = \dots$ é um argumento inválido. Desta pegadinha fica a lição de que as substituições não podem ser feitas de maneira automática e impensada. Vale o mesmo para dividir por 0. (por exemplo em $a = b$ multiplicado por a fica $a^2 = ab$ e subtraindo b^2 fica $a^2 - b^2 = ab - b^2$ e fatorando $b(a - b) = (a + b)(a - b)$ e simplificando $b = a + b$ e como $a = b$ recebe-se $1 = 2 \dots$).

Equações quadráticas

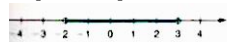
É uma equação que tem 2 como maior expoente da variável. Ela pode ser escrita como $ax^2 + bx + c = 0$, onde $a \neq 0$. Esta última restrição é necessária senão a equação reverte para uma equação linear. Muitas vezes é possível reverter uma equação quadrática para o produto $(x - x_1)(x - x_2) = 0$. Se isto for possível (x_1 e x_2 são as raízes), então pode-se usar a propriedade de fator zero que diz "Se a e b são números reais, com $ab = 0$ então a ou b ou ambos são iguais a zero. Por exemplo, para resolver $(x - 4)(3x + 7) = 0$, pode-se fazer $x - 4 = 0$ e $3x + 7 = 0$ e resolver cada equação separadamente e as raízes serão 4 e $-7/3$. Cada expressão como $x - 4$ é chamada fator. E o processo de passar da equação $ax^2 + bx + c = 0$ para $(x - x_1)(x - x_2) = 0$ é chamado fatoração. Ajuda aqui saber que em $x^2 - Sx + P = 0$, S é a soma das raízes e P o seu produto. Por exemplo, em $p^2 + 7p + 12 = 0$ as duas raízes somadas devem dar -7 e multiplicadas devem dar 12. Logo, as raízes são -3 e -4 . A solução geral de uma equação de segundo grau $ax^2 + bx + c = 0$ é dada por $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. A duplicidade de raízes (uma equação de grau 2 tem 2 raízes) é dada pela presença do sinal $\pm$. Se dentro da raiz quadrada se obtiver 0, ambas raízes serão iguais. E se se obtiver um número menor que zero, as raízes serão complexas.

Inequações lineares

Agora as duas expressões em estudo são desiguais. Se antes a solução era um número (eq. linear) ou dois (eq. segundo grau), agora a solução pode ser um conjunto infinito de números. Usualmente aqui a resposta é dada sobre a reta numerada. Por exemplo ao resolver a inequação $x \geq -2$ a resposta é



Note que a resposta começa em -2 , com um ponto cheio. Isto significa que o valor -2 faz parte da solução. Na inequação $-2 < x < 3$ deve ser respondida por



Veja que agora tanto -2 como 3 são círculos abertos sobre os números. O significado é que esses pontos NÃO fazem parte da solução.

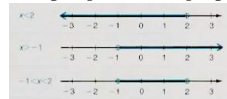
Para resolver inequações, supondo a, b e c , números reais, usam-se 3 propriedades

- se $a < b$ então $a + c < b + c$
- se $a < b$ e se $c > 0$ então $ac < bc$
- se $a < b$ e se $c < 0$ então $ac > bc$

Esta última propriedade demanda algum cuidado. Se iniciarmos com $-24 < 12$ e multiplicarmos a inequação por -1 precisamos inverter o sinal da inequação para ficar com $24 > -12$. Em resumo, se ambos os lados de uma inequação são multiplicados ou divididos por um negativo, o sinal da inequação deve ser revertido.

Inequações compostas e valor absoluto

Uma inequação composta é feita por 2 ou mais inequações conectadas por $\wedge$ (e). Por exemplo, seja a inequação composta formada por $x < 2$ e por $x > -1$. Desenhando separadamente, a resposta final é a sobreposição das duas respostas. Note que examinando o gráfico pode-se concluir que outra redação para a inequação composta é $-1 < x < 2$.



Resolva agora: $x < 4$ e $x < 2$. Depois: $x \leq -4$ e $x \geq -1$.

Para resolver inequações envolvendo módulo, use as seguintes regras:

- Para $b > 0$, ao resolver $|a| = b$, ache $a = b$ ou $a = -b$
- Para resolver $|a| = |b|$ ache $a = b$ ou $a = -b$

Em resumo, assumindo $b > 0$, para resolver $|a| < b$ faça $-b < a < b$ e para resolver $|a| > b$ faça $a < -b$ ou $a > b$.

Exemplos resolvidos

A seguir, alguns exemplos de inequações a resolver:

- dado $3 < |x + 1| \leq 4$
 Como se sabe, há que transformar essa inequação em duas e resolvê-las separadamente. Depois, os resultados tem que ser integrados usando o conector E ($\wedge$).
 1.1 dado $3 < |x + 1|$
 1.1.1 - ao eliminar o módulo, faz-se $3 < x + 1$ e daí tem-se $2 < x$
 1.1.2 - o lado oposto do módulo, faz-se $3 < -(x + 1)$ e multiplicando-se por -1 fica: $-3 > x + 1$ e daí $-4 > x$. A resposta à primeira parte da inequação é $(-\infty, -4)$, $(2, \infty)$
 1.2 dado $|x + 1| \leq 4$
 1.2.1 eliminando o módulo, fica $x + 1 \geq 4$ e daí $x \geq 3$
 1.2.2 lado oposto: $-(x + 1) \geq 4$, multiplicando por -1 , dá $x + 1 \leq -4$ ou $x \leq -5$. A resposta à segunda parte da inequação é $(-\infty, -5]$, $[3, \infty)$. Juntando as 2 respostas usando $\wedge$ fica $(-\infty, -5] \cup [3, \infty)$ que é a resposta desejada.

- Seja o sistema de inequações formado por $|x + 11| \geq 1$ e $|x + 2| > 7$. Para resolvê-lo, como sempre há que resolver as duas inequações separadamente e depois juntar o resultado. Resolvendo $|x + 11| \geq 1$, tem-se como resultado $x \geq -10$ ou $x \leq -12$. Resolvendo a inequação $|x + 2| > 7$, tem-se $x > 5$ ou $x < 9$. Mesclando as duas respostas, o intervalo de solução é $(-\infty, -12] \cup (5, \infty)$, que é a resposta. 3. seja $3 > |x - 1| \geq -1$
 3.1 tem-se $|x - 1| < 3$ (note que inverte-se a inequação, para deixar o x à esquerda, não há nenhum problema (veja que $>$ virou $<$)).
 3.1.1 agora $x - 1 < 3$ ou $x < 4$
 3.1.2 lado oposto: $-(x - 1) < 3$ vezes -1 fica $x - 1 > -3$ ou $x > -2$. Juntando ambas, fica $(-2, 4)$.
 3.2 agora $|x - 1| \geq -1$
 3.2.1 faz-se $x - 1 \geq -1$ e daí $x \geq 0$
 3.2.2 lado oposto $-(x - 1) \geq -1$ e multiplicando por -1 fica $x - 1 \leq 1$ e daí $x \leq 2$. Juntando ambas, fica $(-\infty, 2]$. E finalmente a resposta final é $(-\infty, 2] \cap (-2, 4) = (-2, 2]$.
 4.1 dado $-1 \geq |x - 1|$
 4.1.1 eliminando o módulo, fica $-1 \geq x - 1$ ou $0 \geq x$
 4.1.2 lado oposto $-1 \geq -(x - 1)$ e multiplicando por -1 fica $1 \leq x - 1$ e $2 \leq x$. Daqui, percebe-se que a resposta a esta primeira parte é $(-\infty, \infty)$.
 4.2.1 agora calcula-se $|x - 1| > 4$. Tirando o módulo, fica $x - 1 > 4$ e obtem-se $x > 5$.
 4.2.2 fazendo-se $-(x - 1) > 4$ e multiplicando por -1 fica $x - 1 < -4$ e daí $x < -3$. A resposta à segunda parte é $(-\infty, -3)$, $(5, \infty)$ que é a resposta final, uma vez que a primeira parte, por valer $(-\infty, \infty)$ não interfere no resultado final.

Para você fazer

Resolva a seguir inequações ou sistemas de inequações e ache os intervalos de solução:

- $2 \geq |x + 5| > -2$
- $|x - 6| \geq 13$ e $|x + 10| \geq 6$
- $-2 \geq |x + 6| > 2$
- $|x + 7| > 10$ e $|x - 6| \leq 9$
- $3 < |x + 6| > 1$

Responda aqui

1
2
3
4
5



Em cada um dos sete algoritmos a seguir, descubra qual número é devolvido/impresso. Antes de prosseguir nos exercícios, relembre as regras de cada um:

enquanto
enquanto A faça
| ...
fimenquanto

A é testado. Se verdadeiro, a sequência ... é executada. Ao se encontrar o comando fimenquanto, existe um retorno ao comando enquanto e A é novamente avaliada. Se A é falso, há um desvio para o comando imediatamente posterior ao fimenquanto.

repita
repita
| ...
até A

A sequência ... é executada ao menos uma vez. Ao chegar ao comando até, A é testada. Se verdadeira, a repetição é abandonada e executa-se o próximo comando após o até. Se A é falso, existe um desvio para o comando repita.

para
para A de B até C passo D faça
| ...
fimpara

Existem 2 casos: Se D é positivo, A é inicializado com o valor B e ocorre o teste $A \leq C$. Se esta condição é verdadeira, a sequência ... é executada. Ao chegar no fimpara, A é incrementado com o valor D e desvia-se até o para, quando o teste é refeito. Se a condição é falsa, ocorre um desvio para o próximo comando após o fimpara.

Se D é negativo, A é inicializado com o valor B e ocorre o teste $A \geq C$. Se esta condição é verdadeira, a sequência ... é executada. Ao chegar no fimpara, A é decrementado com o valor $abs(D)$ e desvia-se até o para, quando o teste é refeito. Se a condição é falsa, ocorre um desvio para o próximo comando após o fimpara. **Aqui, uma sugestão: escreva o fluxograma dos 3 comandos estudados.**

Exercício 1

```
1: funcao ex1
2: A, B, C, D: inteiro
3: B ← 3
4: A ← 2
5: D ← 5
6: C ← 5
7: B ← 3
8: enquanto B ≤ 13
9:   D ← B+2
10:  A ← 13
11:  repita
12:    se ((B+D) mod 2) = 0
13:      C ← C - 1
14:    senão
15:      C ← C + 2
16:    fim{se}
17:    A ← A-4
18:  até A < 1
19:  B ← B+4
20: fim{enquanto}
21: se 15 > B+C
22:   devolva B+D
23: senão
24:   devolva A-C
25: fim{se}
26: fim funcao
```

Exercício 2

```
1: funcao ex2
2: A, B, C, D: inteiro
3: A ← 3
4: C ← 3
5: B ← 4
6: D ← 5
```

```
7: D ← A+1
8: repita
9:   B ← A+2
10:  C ← 11
11:  enquanto C ≥ 2
12:    se ((A+B) mod 2) = 0
13:      D ← D - 1
14:    senão
15:      D ← D + 2
16:    fim{se}
17:    C ← C-3
18:  fim{enquanto}
19:  A ← A+4
20: até A ≥ 10
21: se 11 > A+D
22:   devolva A+B
23: senão
24:   devolva C-D
25: fim{se}
26: fim funcao
```

Exercício 3

```
1: funcao ex3
2: A, B, C, D: inteiro
3: C ← 6
4: D ← 2
5: A ← 5
6: B ← 4
7: C ← 12
8: enquanto C ≥ 3
9:   A ← C+2
10:  D ← 2
11:  enquanto D < 10
12:    se ((C+A) mod 2) = 0
13:      B ← B - 1
14:    senão
15:      B ← B + 2
16:    fim{se}
17:    D ← D+3
18:  fim{enquanto}
19:  C ← C-3
20: fim{enquanto}
21: se 12 > C+B
22:   devolva C+A
23: senão
24:   devolva D-B
25: fim{se}
26: fim funcao
```

Exercício 4

```
1: funcao ex4
2: A, B, C, D: inteiro
3: C ← 3
4: B ← 4
5: A ← 2
6: D ← 4
7: C ← 3
8: repita
9:   A ← C+1
10:  D ← C+1
11:  para B de 3 a 10 passo 3
12:    se ((C+A) mod 2) = 0
13:      D ← D - 1
14:    senão
15:      D ← D + 2
16:    fim{se}
17:    A ← B+1
18:  fim{para}
19:  C ← C+3
20: até C ≥ 12
21: se 11 > C+D
22:   devolva C+A
23: senão
24:   devolva B-D
25: fim{se}
26: fim funcao
```

Exercício 5

```
1: funcao ex5
2: A, B, C, D: inteiro
```

```
3: B ← 6
4: C ← 6
5: D ← 3
6: A ← 3
7: A ← C+2
8: para B de 14 a 3 passo -3
9:   D ← B+2
10:  C ← 13
11:  enquanto C > 2
12:    se ((B+D) mod 2) = 0
13:      A ← A - 1
14:    senão
15:      A ← A + 2
16:    fim{se}
17:    C ← C-3
18:  fim{enquanto}
19:  D ← C+1
20: fim{para}
21: se 11 > B+A
22:   devolva B+D
23: senão
24:   devolva C-A
25: fim{se}
26: fim funcao
```

Exercício 6

```
1: funcao ex6
2: A, B, C, D: inteiro
3: A ← 6
4: D ← 5
5: C ← 4
6: B ← 1
7: A ← 11
8: repita
9:   C ← A+2
10:  D ← 2
11:  repita
12:    se ((A+C) mod 2) = 0
13:      B ← B - 1
14:    senão
15:      B ← B + 2
16:    fim{se}
17:    D ← D+4
18:  até D ≥ 11
19:  A ← A-3
20: até A ≤ 2
21: se 15 > A+B
22:   devolva A+C
23: senão
24:   devolva D-B
25: fim{se}
26: fim funcao
```

Exercício 7

```
1: funcao ex7
2: A, B, C, D: inteiro
3: C ← 6
4: B ← 2
5: D ← 2
6: A ← 4
7: D ← C+1
8: para C de 10 a 3 passo -3
9:   D ← C+2
10:  A ← C+2
11:  para B de 1 a 14 passo 2
12:    se ((C+D) mod 2) = 0
13:      A ← A - 1
14:    senão
15:      A ← A + 2
16:    fim{se}
17:    A ← B+1
18:  fim{para}
19:  D ← B+2
20: fim{para}
21: se 15 > C+A
22:   devolva C+D
23: senão
24:   devolva B-A
25: fim{se}
26: fim funcao
```

Respostas

ex1	ex2	ex3	ex4	ex5	ex6	ex7



Algoritmos de função

Exercício 1

```
1: funcao programa11
2: inteiro A,B,C,D,Q,X,N,E,U,L,Y
3: inteiro R,K,H
4: A←1; B←2; C←1;D←2;Q←3
5: X←3; N←2; E←1;U←3;L←2
6: Y←2; R←3; K←2;H←2;Q←2
7: repita
8:   X←3
9:   enquanto X<1
10:    D←4
11:    se D=4
12:     N←3
13:     enquanto N≤9
14:      E←E+B+3
15:      U←U+3
16:      L←L+1
17:      N←N+2
18:    fim{enquanto}
19:  senão
20:   Y←3
21:   repita
22:    R←R+3
23:    K←K+3
24:    H←H+D+3
25:    Y←Y+2
26:   até Y>9
27:   fim{se}
28:   X←X+1
29:   fim{enquanto}
30:   Q←Q+2
31: até Q≥7
32: imprima R+C+Q+K
33: fim{função}
```

Exercício 2

```
1: funcao programa12
2: inteiro A,B,C,D,F,U,E,L,J,N,K
3: inteiro T,H,Q,S,X,R,O
4: A←1; B←2; C←2;D←2;F←1
5: U←2; E←1; L←2;J←2;N←1
6: K←1; T←3; H←2;Q←1;S←2
7: X←2
8: R←3
9: O←3
10: A←8
11: se A<5
12:   F←1
13:   repita
14:     U←2
15:     enquanto U≤1
16:     C←2
17:     se C≠4
18:      E←E+1
19:      L←L+2
20:   senão
21:    J←J+B+2
22:    N←N+D+2
23:    K←K+1
24:    T←T+1
25:    H←H+C+2
26:   fim{se}
27:   U←U+2
28:   fim{enquanto}
29:   F←F+1
30:   até F>6
31: senão
32:   Q←1
33:   enquanto Q<7
34:    S←2
35:    enquanto S≤8
36:     X←X+1
37:     R←R+C+1
38:     O←O+1
39:     S←S+2
40:   fim{enquanto}
```

```
41:   Q←Q+2
42:   fim{enquanto}
43: fim{se}
44: imprima F+U+A+T
45: fim{função}
```

Exercício 3

```
1: funcao programa13
2: inteiro A,B,C,D,W,J,Y,E,T,U,Z
3: inteiro F,V,L
4: A←2; B←3; C←2;D←1;W←1
5: J←2; Y←3; E←3;T←3;U←3
6: Z←3; F←2; V←1;L←3;B←7
7: se B<4
8:   W←1
9:   enquanto W<4
10:    J←3
11:    enquanto J<4
12:     Y←3
13:     enquanto Y≤2
14:      E←E+B+1
15:      T←T+3
16:      U←U+2
17:      Y←Y+2
18:    fim{enquanto}
19:    J←J+2
20:    fim{enquanto}
21:    W←W+2
22:  fim{enquanto}
23: senão
24:   Z←2
25:   repita
26:     F←3
27:     repita
28:       V←2
29:       repita
30:         L←3
31:         repita
32:           L←L+2
33:           até L≥1
34:           V←V+1
35:         até V>2
36:         F←F+2
37:       até F>6
38:       Z←Z+2
39:     até Z>5
40:   fim{se}
41: imprima Y+C+V+T
42: fim{função}
```

Exercício 4

```
1: funcao programa14
2: inteiro A,B,C,D,O,P,U,Q,S,I,N
3: inteiro F,Z,W,G,E,L,V
4: A←3; B←1; C←3;D←2;O←3
5: P←1; U←3; Q←1;S←3;I←1
6: N←2; F←2; Z←2;W←3;G←2
7: E←2
8: L←2
9: V←1
10: D←4
11: se D>9
12:   A←1
13:   se A≥3
14:    O←2
15:    repita
16:     P←1
17:     enquanto P≤1
18:     A←2
19:     se A≤1
20:      U←U+2
21:      Q←Q+2
22:      S←S+C+2
23:      I←I+C+2
24:      N←N+D+2
25:   senão
```

```
26:   F←F+B+1
27:   Z←Z+D+3
28:   W←W+A+3
29:   G←G+2
30:   E←E+C+3
31:   fim{se}
32:   P←P+2
33:   fim{enquanto}
34:   O←O+2
35:   até O≥8
36:   fim{se}
37: senão
38:   C←8
39:   se C<5
40:    L←1
41:    repita
42:     V←2
43:     repita
44:       V←V+2
45:     até V>8
46:     L←L+2
47:     até L≥9
48:   fim{se}
49: fim{se}
50: imprima L+U+S+E
51: fim{função}
```

Exercício 5

```
1: funcao programa11
2: inteiro A,B,C,D,Q,N,G,Z,O,V,L
3: inteiro S,P,I
4: A←2; B←2; C←1;D←1;Q←2
5: N←1; G←3; Z←2;O←3;V←3
6: L←1; S←1; P←2;I←2;Q←3
7: repita
8:   N←3
9:   enquanto N<4
10:    C←6
11:    se C<3
12:     G←3
13:     enquanto G≤9
14:      Z←Z+D+1
15:      O←O+C+3
16:      V←V+C+2
17:      G←G+1
18:    fim{enquanto}
19:   senão
20:    L←3
21:    repita
22:     S←S+1
23:     P←P+A+2
24:     I←I+2
25:     L←L+2
26:     até L>3
27:   fim{se}
28:   N←N+2
29:   fim{enquanto}
30:   Q←Q+1
31: até Q≥5
32: imprima D+Z+I+C
33: fim{função}
```

Exercício 6

```
1: funcao programa12
2: inteiro A,B,C,D,S,O,L,J,H,I,N
3: inteiro V,Y,Z,X,G,Q,F
4: A←1; B←3; C←2;D←1;S←1
5: O←2; L←2; J←3;H←2;I←2
6: N←2; V←2; Y←3;Z←1;X←1
7: G←3
8: Q←3
9: F←1
10: C←2
11: se C≠9
12:   S←2
13:   repita
```

```
14:   O←3
15:   enquanto O<9
16:    B←3
17:    se B≤4
18:     L←L+A+2
19:     J←J+D+1
20:   senão
21:    H←H+2
22:    I←I+1
23:    N←N+1
24:    V←V+3
25:    Y←Y+B+2
26:   fim{se}
27:   O←O+1
28:   fim{enquanto}
29:   S←S+1
30:   até S≥2
31: senão
32:   Z←3
33:   enquanto Z≤2
34:    X←3
35:    enquanto X≤9
36:     G←G+A+3
37:     Q←Q+2
38:     F←F+D+1
39:     X←X+1
40:   fim{enquanto}
41:   Z←Z+1
42:   fim{enquanto}
43: fim{se}
44: imprima F+V+I+H
45: fim{função}
```

Exercício 7

```
1: funcao programa13
2: inteiro A,B,C,D,W,R,E,L,K,M,X
3: inteiro N,H,O
4: A←1; B←3; C←2;D←1;W←2
5: R←1; E←3; L←3;K←3;M←3
6: X←1; N←3; H←1;O←1;C←2
7: se C<9
8:   W←1
9:   enquanto W<3
10:    R←2
11:    enquanto R<2
12:     E←3
13:     enquanto E<3
14:      L←L+3
15:      K←K+1
16:      M←M+1
17:      E←E+1
18:    fim{enquanto}
19:    R←R+2
20:    fim{enquanto}
21:    W←W+1
22:    fim{enquanto}
23: senão
24:   X←2
25:   repita
26:     N←3
27:     repita
28:       H←1
29:       repita
30:         O←1
31:         repita
32:           O←O+1
33:         até O≥4
34:       H←H+2
35:       até H≥8
36:       N←N+2
37:     até N≥8
38:     X←X+2
39:   até X>9
40: fim{se}
41: imprima R+C+B+H
42: fim{função}
```

Respostas:

1	2	3	4	5	6	7



Geometrias não euclidianas

Para os antigos gregos, matemática era geometria (geo=terra, metria=medir). E, uma das principais e importantes ideias era a da prova rigorosa. Cerca de 300 AEC, Euclides escreveu o livro "Elementos". A potência deste livro cerca o extraordinário: ele tem sido estudado quase sem mudança até os nossos dias, como um livro texto de Geometria pelo seu modelo de lógica dedutiva. O livro começa com definições básicas tais como ponto, reta e plano. Euclides dá então 5 postulados (postular=pedir para você acreditar), para construir o resto do texto. São estes os postulados:

- 1. Dois pontos determinam uma e só uma reta.
- 2. Uma reta se estende em qualquer direção ao infinito.
- 3. Um círculo pode ser desenhado em qualquer centro e com qualquer raio.
- 4. Todos os ângulos retos são iguais.
- 5. Dada uma reta *k* e um ponto *P* fora dela, existe apenas uma reta *m* que passa por *P* e é paralela a *k*.

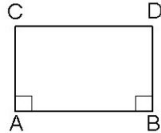
Depois ele cita mais 5 axiomas (coisas iguais a uma terceira são iguais entre si; se iguais são somados a iguais, a soma permanece igual; idem para subtraídos; figuras que coincidem são iguais e o todo é maior do que qualquer parte) e com estas 10 afirmações, mais as regras da lógica, ele prova um grande número de "proposições" sobre figuras geométricas.

Tudo parece ter corrido mais ou menos bem, mas o quinto postulado (o da paralela) nunca deixou de causar algum desconforto.

Outra maneira de afirmá-lo é que se duas retas *k* e *m* são cortadas por uma terceira reta *n*, então se a soma dos dois ângulos internos é menor que dois ângulos retos, então as retas *k* e *m* se encontrarão em algum ponto no lado de *n*. Todas as outras 9 afirmações são simples e facilmente aceitas, mas o postulado das paralelas é mais complexo de entender sem pelo menos um rascunho gráfico ou muito raciocínio.

Pesquisadores desconfiaram que o postulado era um teorema: havia que prová-lo. Eis uma lista de matemáticos que andaram escrevendo sobre o quinto postulado: Frederico Comandino (1509-1575, Itália); Chistopher S. Clavio (1537-1612, Alemanha); Pietro A. Cataldi (1548-1626, Itália); G.A. Boreli (1608-1679, Itália); Giordano Vitale (1633-1711, Itália); J. Wallis (1616-1703, Inglaterra); Girolamo Saccheri (1667-1733, Itália); Johann Heinrich Lambert (1728-1777, Suíça); Adrien Marie Legendre (1752-1833, França); Nicolai Lobachevsky (1792-1856, Rússia); Farkas Bolyai (1775-1856, Hungria); Janos Bolyai (1802-1860, Hungria); Beltrami (1835-1900, Itália); Arthur Cayley (1821-1895, Inglaterra); Felix Klein(1849-1925, Alemanha); Henri Poincaré (1854-1912, França) e Georg Bernhard Riemann (1826-1866, Alemanha).

O jesuíta italiano Girolamo Saccheri tentou a prova usando a redução ao absurdo. Começou com um retângulo *ABCD*.



Os ângulos *A* e *B* são retos e os lados *AC* e *BD* são iguais. Ele pretendia assumir que *∠C* e *∠D* são obtusos e mostrar que isto leva a uma contradição. Depois assumir que eles são agudos e também obter uma contradição. Daí, se *∠C* e *∠D* não podem ser obtusos e nem agudos só podem ser retos. Se *∠C* e *∠D* são retos, estava provado o postulado das paralelas. O problema é que a segunda afirmação (*∠C* e *∠D* são agudos) não gera uma contradição, ainda que ele tenha feito força para se convencer: seu livro tinha como título *Euclides livre de todas as falhas*.

Entretanto, os 10 postulados-axiomas foram e são muito úteis para descrever o mundo. Hoje se sabe que o quinto postulado é verdadeiro em um mundo plano. Mas, ao mudar o quinto postulado

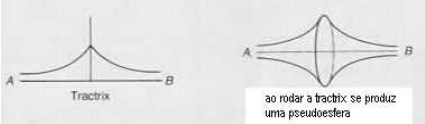
pode-se descrever outros universos (dispostos em outras superfícies). Um sistema geométrico onde o quinto postulado não é verdadeiro descreve uma *geometria não euclidiana*.

Uma das primeiras geometrias deste tipo foi desenvolvida por 3 pessoas diferentes mais ou menos ao mesmo tempo: No começo do século 19, Gauss (na minha opinião, o maior matemático) trabalhou em uma geometria consistente na qual ele trocou o quinto postulado. Ele nunca publicou seu trabalho, ficou com medo de parecer ridículo e ser atacado por quem não conseguia pensar "fora da casinha". É dele o termo "não euclidiano". Lobachevski (1793-1856) publicou um sistema similar em 1830 em língua russa. Janos Bolyai, um militar húngaro, fez o mesmo publicando o material em 1832, ainda sem saber do trabalho de Lobachevski. Bolyai nunca se recuperou do desapontamento de não ter sido o primeiro: nunca mais produziu nenhuma matemática. Lobachevski trocou o quinto postulado por *os ângulos C e D são agudos* e o reescreveu como *Dado o ponto P e a linha k, ao menos duas linhas diferentes passando por P podem ser paralelas a k*. Deve-se notar que este fenômeno (ter dificuldade em provar alguma coisa e descobrir depois de algum tempo que a negativa é que era uma baita descoberta) é algo comum na ciência. Por exemplo, o erro em calcular a variação da velocidade da luz implicou em ter a mecânica newtoniana substituída pela mecânica relativística de Einstein, ou a dificuldade de eliminar a radiação de fundo na radioastronomia levou à descoberta de sua importância.

Geometria Lobachevskiana

Muitos dos teoremas de Euclides são válidos na geometria de Lobachevski, mas muitos não são. Por exemplo, aqui, a soma dos ângulos internos de um triângulo é menor do 180°.

A geometria Lobachevskiana pode ser representada em uma pseudoesfera formada pela revolução de uma curva chamada tractrix (Em palavras: suponha um carrinho percorrendo uma linha reta sobre uma mesa. Amarrado ao carrinho um cordão finito, inicialmente perpendicular ao trajeto do carrinho. Na ponta do cordão, uma caneta. Veja a figura. Ao rodar a tractrix produz-se a pseudoesfera. (Ela também é conhecida como geometria hiperbólica, mas o nome é ruim: não tem muito a ver com a hipérbole).



Geometria Riemanniana

Uma outra geometria plenamente desenvolvida por Riemann, começou distinguindo o fato de uma reta ser infinita e ter comprimento infinito: por exemplo uma circunferência sobre a superfície de uma esfera continua infinitamente, mas não tem comprimento infinito (na verdade mede 2πr). Outro nome para esta geométrica é elipsoidal. Lembre que a esfera é um caso particular de elipsóide. Agora é hora de fazer a experiência da laranja. Corte um oitavo de laranja: um corte no equador e dois outros cortes na direção dos gomos e a 90° entre si. Você ficará com um triângulo esférico e com o detalhe de que seus três ângulos têm 90° cada. Agora 90° + 90° + 90° = 270°. Acaba-se de descrever a geometria que importa na navegação em nosso planeta. Veja o trajeto de um avião entre Fortaleza e Lisboa (reta vermelha) que se contrapõe à menor distância euclidiana no mapa (linha azul).



Geometria do motorista de taxi

Deixando os contextos históricos vistos acima, pode-se criar mais geometrias. Um exemplo clássico é o do motorista de taxi em Manhattan. A ilha em Nova Iorque é conhecida por suas ruas

(sentido leste-oeste) e avenidas (norte-sul). Não se pode ir de um ponto a outro de taxi, atravessando prédios e casas, passando pelos seus telhados (eles podem ser muito altos e você não é Clark Kent). Há que obedecer a malha de ruas e avenidas. A maioria das ruas são numeradas, começando com a rua 1 no sul e terminando com a rua 191 no norte. As ruas de número ímpar vão no sentido oeste e as ruas de número par vão no sentido leste. Existem algumas exceções a esta regra, como no bairro de Greenwich Village, onde as ruas têm nomes. As avenidas também têm exceção, como a avenida Broadway. O que nos interessa aqui é que ruas e avenidas se cruzam em 90°. Então esta geometria redefine por exemplo a menor distância entre os pontos *A* e *B* que no modelo euclidiano é $d_{AB} = \sqrt{(b_x - a_x)^2 + (b_y - a_y)^2}$ como $d_{AB} = |b_x - a_x| + |b_y - a_y|$. Note que a distância no modelo euclidiano é única enquanto que no modelo do motorista de taxi é múltipla.

Um resumo

A seguir uma imagem que resume as características das geometrias, tirada de *Mathematical Ideas* de Charles Milers and others, pág 475.

Euclides (300AEC)	Lobachevski (1830)	Riemann (1850)
Ângulos <i>C</i> e <i>D</i> do quadrilátero de Saccheri são ângulos retos	Ângulos <i>C</i> e <i>D</i> são agudos	Ângulos <i>C</i> e <i>D</i> são obtusos
Dado um ponto <i>p</i> e uma reta <i>k</i> só há uma linha paralela a <i>k</i> em <i>p</i>	Há mais de uma reta paralela	Não há paralela
$A + B + C = 180^\circ$	$A + B + C < 180^\circ$	$A + B + C > 180^\circ$
Dois triângulos podem ter ângulos iguais (similaridade) e lados diferentes (congruência)	dois triângulos com mesmos ângulos têm os mesmos lados	dois triângulos com mesmos ângulos têm os mesmos lados
Retas têm comprimento infinito	Retas têm comprimento infinito	Retas tem comprimento finito
Geometria no plano	Geometria em superfície como a pseudoesfera	Geometria na esfera

Para você fazer

1. Refaça o exemplo da laranja ou da maçã em sua casa. Ao mesmo tempo que terá uma alimentação saudável e gostosa, reflita sobre o triângulo obtido e como a humanidade levou 2000 anos para descobrir uma obviedade que estava na nossa frente o tempo todo. Há precedentes: Aristóteles acreditava que os humanos tinham 44 dentes, sendo 20 na mandíbula superior e 24 na mandíbula inferior. Esta crença foi baseada em suas observações de animais, como cães e gatos, que têm um número semelhante de dentes. Na realidade os humanos adultos têm apenas 32 dentes, sendo 16 na mandíbula superior e 16 na mandíbula inferior. Bastava abrir a boca na frente do espelho e contar. A despeito disso, as escolas de medicina ensinaram errado até o século XVI, quando o anatomista italiano Gabriele Falloppio estudou a anatomia humana de forma mais cuidadosa e descobriu que os humanos adultos têm apenas 32 dentes.

2. Usando a geometria do motorista de taxi, ache a menor distância entre os pontos *A* = (1, -11), *B* = (-6, 11).



Seguir o código: repetições

Nos exercícios a seguir, faça três coisas: a. Siga o código e ache o resultado impresso. b. Depois, traduz este código para C++ ou para Python (como queira) e mande executar. c. Só depois que obtiver uma coincidência de resultado, responda esta folha e entregue-a. Lembre que para x de y ate z passo k, executa se o valor achado for $\leq z$.

```
-----1-----
1:funcao AAAA
2:A = 7
3:B = 1
4:C = 9
5:D = 2
6:D = C - 6
7:A = D - 5
8:G = 18
9:enquanto G < 30
10:A = D + 4
11:para M de 30 ate 2 passo -7
12:C = D - 7
13:Q = 16
14:enquanto Q > -14
15:C = B - 3
16:D = C - 9
17:se B = 8
18:C = D - 6
19:senao
20:B = B - 7
21:fim{se}
22:Q = Q+-10
23:fim{enquanto}
24:fim{para}
25:G = G+4
26:fim{enquanto}
27:imprima (A+B+C+D)
28:fim{funcao}
-----2-----
1:funcao AAAA
2:A = 5
3:B = 5
4:C = 8
5:D = 9
6:C = B + 6
7:B = D + 6
8:P = 29
9:enquanto P > 11
10:C = B - 2
11:se B > 2
12:A = A + 2
13:A = D + 2
14:fim{se}
15:B = B + 7
16:para Q de 21 ate 5 passo -8
17:B = A + 5
18:R = 9
19:enquanto R < 17
20:D = B + 3
21:D = A - 7
22:se C >= 5
23:C = B - 4
24:C = B + 1
25:fim{se}
26:R = R+2
27:fim{enquanto}
28:fim{para}
29:P = P+-9
30:fim{enquanto}
31:imprima (A+B+C+D)
32:fim{funcao}
-----3-----
1:funcao AAAA
2:A = 2
3:B = 6
4:C = 7
5:D = 9
6:A = B + 3
7:A = B + 4
8:para Q de 12 ate -6 passo -6
9:C = B + 9
10:F = 7
11:enquanto F > -5
12:A = A - 9
13:F = F+-3
14:fim{enquanto}
15:fim{para}
16:imprima (A+B+C+D)
17:fim{funcao}

-----4-----
1:funcao AAAA
2:A = 6
3:B = 1
4:C = 9
5:D = 6
6:D = A + 9
7:para K de 23 ate 2 passo -7
8:C = C - 1
9:Q = 19
10:enquanto Q < 43
11:B = C - 4
12:se D > 9
13:A = D + 6
14:senao
15:B = C + 3
16:fim{se}
17:B = B + 4
18:A = B + 4
19:J = 30
20:enquanto J > 15
21:C = A + 1
22:C = C - 2
23:J = J+-3
24:fim{enquanto}
25:Q = Q+8
26:fim{enquanto}
27:fim{para}
28:imprima (A+B+C+D)
29:fim{funcao}
-----5-----
1:funcao AAAA
2:A = 8
3:B = 4
4:C = 9
5:D = 8
6:C = C - 5
7:B = C - 6
8:para P de 22 ate 42 passo 5
9:C = A - 9
10:K = 23
11:enquanto K > 11
12:B = C + 5
13:C = C + 5
14:K = K+-6
15:fim{enquanto}
16:fim{para}
17:imprima (A+B+C+D)
18:fim{funcao}
-----6-----
1:funcao AAAA
2:A = 8
3:B = 9
4:C = 5
5:D = 5
6:para G de 19 ate -31 passo -1
7:D = A + 1
8:C = A + 1
9:se (C > 1)|| (D = 6)
10:C = B + 4
11:senao
12:D = A + 4
13:fim{se}
14:para M de 10 ate 25 passo 3
15:C = C - 2
16:F = 31
17:enquanto F < 58
18:B = D - 4
19:se (D <= 7)|| (C <= 3)
20:D = D + 1
21:senao
22:D = B + 3
23:fim{se}
24:F = F+9
25:fim{enquanto}
26:fim{para}
27:fim{para}
28:imprima (A+B+C+D)
29:fim{funcao}
-----7-----

1:funcao AAAA
2:A = 4
3:B = 9
4:C = 8
5:D = 4
6:para P de 25 ate 21 passo -1
7:A = D - 1
8:L = 10
9:enquanto L > -2
10:C = B + 4
11:se (C != 1)&&(A >= 7)
12:B = C - 2
13:senao
14:D = B - 9
15:fim{se}
16:C = B - 9
17:B = C - 4
18:L = L+-3
19:fim{enquanto}
20:fim{para}
21:imprima (A+B+C+D)
22:fim{funcao}
-----8-----
1:funcao AAAA
2:A = 1
3:B = 9
4:C = 6
5:D = 1
6:A = A - 3
7:para N de 17 ate 38 passo 7
8:C = C + 1
9:J = 30
10:enquanto J < 42
11:A = A - 1
12:B = B + 5
13:se D < 2
14:B = A + 3
15:senao
16:A = A - 9
17:fim{se}
18:para Q de 9 ate 12 passo 1
19:A = B + 4
20:M = 13
21:enquanto M < 53
22:B = A + 4
23:M = M+8
24:fim{enquanto}
25:fim{para}
26:J = J+3
27:fim{enquanto}
28:fim{para}
29:imprima (A+B+C+D)
30:fim{funcao}
-----9-----
1:funcao AAAA
2:A = 8
3:B = 4
4:C = 4
5:D = 5
6:para R de 4 ate 19 passo 5
7:B = A - 8
8:P = 7
9:enquanto P > 2
10:B = B + 2
11:D = C + 3
12:B = B - 8
13:A = B + 4
14:H = 7
15:enquanto H < 25
16:C = B - 7
17:se B = 1
18:D = D - 5
19:senao
20:D = C + 8
21:fim{se}
22:H = H+9
23:fim{enquanto}
24:P = P+-1
25:fim{enquanto}
26:fim{para}

27:imprima (A+B+C+D)
28:fim{funcao}
-----10-----
1:funcao AAAA
2:A = 9
3:B = 8
4:C = 9
5:D = 5
6:A = A + 8
7:B = D + 2
8:B = D - 6
9:para G de 18 ate 12 passo -3
10:B = D + 2
11:F = 6
12:enquanto F < 18
13:D = B - 3
14:B = C - 3
15:F = F+3
16:fim{enquanto}
17:fim{para}
18:imprima (A+B+C+D)
19:fim{funcao}
-----11-----
1:funcao AAAA
2:A = 4
3:B = 8
4:C = 2
5:D = 3
6:B = B + 2
7:L = 18
8:enquanto L > -12
9:C = C + 3
10:se (B < 1)|| (C = 8)
11:A = D - 7
12:senao
13:C = C + 4
14:fim{se}
15:para R de 21 ate 24 passo 1
16:A = C + 4
17:N = 11
18:enquanto N > -34
19:A = B + 4
20:B = D - 2
21:se A <= 4
22:B = C + 9
23:senao
24:A = A + 2
25:fim{se}
26:N = N+-9
27:fim{enquanto}
28:fim{para}
29:L = L+-6
30:fim{enquanto}
31:imprima (A+B+C+D)
32:fim{funcao}
```



Algoritmos II

A seguir existem 4 algoritmos que foram gerados de maneira aleatória. Todos estão corretos, não entram em loop, geram valores perfeitamente determinados e todos têm complexidade aceitável. Execute os testes de mesa pedidos e ao final calcule as 4 somas pedidas.

Algoritmo 1

```

1: Algoritmo 1
2: inteiro  $A, B, C, D, E$ 
3:  $A \leftarrow 16$ 
4:  $B \leftarrow -5$ 
5:  $C \leftarrow 17$ 
6:  $D \leftarrow 7$ 
7:  $E \leftarrow 12$ 
8: enquanto ( $D > 0$ ) faça
9:     se ( $A < -3$ ) então
10:          $B \leftarrow B + 1$ 
11:     fim se
12:     enquanto ( $A > (-1 - 6)$ ) faça
13:          $B \leftarrow A + B + 2$ 
14:          $A \leftarrow A - 2$ 
15:          $E \leftarrow E + 1$ 
16:     se ( $D > (E - 14)$ ) então
17:          $C \leftarrow C + 2$ 
18:     fim se
19:     fim enquanto
20:      $D \leftarrow D - 4$ 
21: fim enquanto
22: imprima ( $A + B + C + D + E$ )
    
```

SOMA = _____ ($A+B+C+D+E$)

Algoritmo 2

```

1: Algoritmo 2
2: inteiro  $F, G, H, I, J$ 
3:  $F \leftarrow 14$ 
4:  $G \leftarrow 1$ 
5:  $H \leftarrow 0$ 
6:  $I \leftarrow 19$ 
7:  $J \leftarrow 10$ 
8: enquanto ( $(H + J) \leq (3 \times J)$ ) faça
9:     se ( $F < -3$ ) então
10:          $G \leftarrow G + 1$ 
11:     fim se
12:     enquanto ( $(F > -2) \vee (G \leq -1)$ ) faça
13:          $G \leftarrow G + 2$ 
14:          $F \leftarrow F - 2$ 
15:     se ( $I > 5$ ) então
16:          $J \leftarrow J + 3 - F$ 
17:     fim se
18:     fim enquanto
19:      $H \leftarrow H + 3$ 
20: fim enquanto
21: imprima ( $F + G + H + I + J$ )
    
```

SOMA = _____ ($F+G+H+I+J$)

Algoritmo 3

```

1: Algoritmo 3
2: inteiro  $K, L, M, N, P$ 
3:  $K \leftarrow 9$ 
4:  $L \leftarrow -9$ 
5:  $M \leftarrow 11$ 
6:  $N \leftarrow 10$ 
7:  $P \leftarrow 1$ 
8: enquanto ( $K \geq -2$ ) faça
9:      $L \leftarrow L + 1$ 
10:    se ( $N \leq 20$ ) então
11:        enquanto ( $M \leq 8$ ) faça
12:             $M \leftarrow M + 6$ 
13:        fim enquanto
14:         $M \leftarrow M + 1$ 
15:    fim se
16:     $K \leftarrow K - 3$ 
17:     $N \leftarrow N + 1$ 
18:     $P \leftarrow P + 2 - K$ 
19:    enquanto ( $N \bmod 3 \neq 0$ ) faça
20:         $N \leftarrow N + 1$ 
21:    fim enquanto
22: fim enquanto
23: imprima ( $K + L + M + N + P$ )
    
```

SOMA = _____ ($K+L+M+N+P$)

Algoritmo 4

```

1: Algoritmo 4
2: inteiro  $Q, R, S, T, U$ 
3:  $Q \leftarrow 1$ 
4:  $R \leftarrow 11$ 
5:  $S \leftarrow -6$ 
6:  $T \leftarrow 15$ 
7:  $U \leftarrow -4$ 
8: repita
9:     se ( $Q < 6$ ) então
10:          $R \leftarrow R + 1$ 
11:     fim se
12:     se ( $T > -7$ ) então
13:          $S \leftarrow S + 2$ 
14:     fim se
15:     enquanto ( $U \leq 20$ ) faça
16:          $U \leftarrow U + 4$ 
17:         se ( $S > 19$ ) então
18:              $T \leftarrow T + 1$ 
19:         fim se
20:     fim enquanto
21:      $Q \leftarrow Q + 2$ 
22: até ( $((Q > 0) \vee (R \neq 0))$ )
23: imprima ( $Q + R + S + T + U$ )
    
```

SOMA = _____ ($Q+R+S+T+U$)

||||| ||||| ||||| - 1 - /

Programas C/C++ com parâmetros

Um programa C/C++ em linha de comando precisa ter uma função `main()` definida dentro dele. Essa exigência é para ter a quem passar o controle da execução quando este programa for chamado a partir da linha de comando de uma janela do sistema operacional.

Tipicamente, um programa pode iniciar perguntando coisas ao seu operador, recebendo dele as respostas e a partir daí definindo o que fazer.

Entretanto, existe uma maneira mais rápida de fazer isso. É combinar com o usuário que ele vai fornecer as informações de que o programa precisa junto com o nome do programa. Em outros termos, ao chamar o programa o operador já vai passar a ele os dados que ele precisa. Com isso, há pelo menos duas vantagens:

- evita-se um ciclo de pergunta e resposta
- mas, principalmente o programa deixa de ser interativo, podendo fazer parte de uma sequência de programas postos a rodar em conjunto sem interferência do operador

A esta última característica dá-se o nome de processamento em lote (ou em inglês, *batch processing*).

Para construir programas com este comportamento, a função `main` do seu programa que era escrita como `main()`, passará a ser escrita como `main(int ac, char *av[])`. Você deve escrever a primeira linha de `main` exatamente como acima descrito, sem modificar nada.

O parâmetro `ac` é uma variável inteira que contém o número de parâmetros passados ao seu programa quando ele foi chamado. O menor valor para `ac` é sempre 1, já que o nome do seu programa é considerado neste caso como um parâmetro.

O parâmetro `av[]` é um vetor de caracteres e ele contém o valor dos parâmetros. O primeiro elemento `av[0]` como já se viu contém o nome do seu programa. Os elementos seguintes, desde `av[1]` até `av[n]` contém os valores dos demais parâmetros.

Uma coisa que pode ser feita neste tipo de programa que espera receber (por exemplo) um parâmetro é o seguinte teste:

```
if (ac != 2) {  
    printf("Chamada inválida ao programa\n");  
    printf("0 correto é ... \n");  
}
```

Se o parâmetro esperado é um número, o compilador C entrega-o como uma cadeia de caracteres (já que a linha de comando é uma cadeia de caracteres). Antes de usar o parâmetro como uma variável numérica é necessário transformá-lo nisso. Veja como

```
#include<math.h>  
int x;  
float y;  
...  
x = atoi(av[1]);  
y = atof(av[2]);  
...
```

Lembrando que `av[1]` é o primeiro parâmetro (`av[0]` é o nome do programa) na forma de uma cadeia de caracteres e `x` é a variável numérica inteira que contém o valor do parâmetro. `y` é similar, mas para variáveis de ponto flutuante. Note a diferença entre variáveis inteiras e reais.

Em C++

Vale mais ou menos a mesma coisa, veja

```
#include<iostream>  
using namespace std;  
int main(int ac, char* av[]){  
    if (ac!=2){  
        cout<<"Erro, parametros invalidos";  
    }  
    else {  
        cout<<"Parametro "<<av[1];  
    }  
}
```

Para você fazer

A seguir a definição de alguns programas C/C++ que você deve escrever, codificar, compilar, testar, depurar e certificar como corretos. Imprima os códigos fonte C/C++ em anexo a esta folha e devolva tudo para o professor. Por favor, identifique claramente onde começa cada programa. Ajude na correção. Obrigado. Se alguma coisa que é pedida ainda lhe é desconhecida, não se avexe, o professor está à sua disposição.

1. Escreva um programa que receba um parâmetro N inteiro e positivo e imprima todos os múltiplos de 17 que são menores do que 200 e maiores do que N . Assim, por exemplo, se $N=49$, o programa deve imprimir 51 68 85 102 119 136 153 170 187. Se $N=102$, o programa deve imprimir 119 136 153 170 187 e se $N=7$, deve imprimir 17 34 51 68 85 102 119 136 153 170 187.
2. Escreva um programa que receba um parâmetro N , inteiro e positivo e menor que 14. O programa deve calcular e imprimir o fatorial de N . Define-se o fatorial como $n! = n \times (n-1) \times (n-2) \times \dots \times 2$. Assim, por exemplo, se o parâmetro for 13, deve ser impresso 6227020800.
3. Escreva um programa que receba um parâmetro R real e positivo e imprima o volume da esfera de raio R . Use a fórmula

$$V = \frac{4}{3} \times \pi \times R^3$$

Assim, por exemplo se o parâmetro for 11, deve ser impresso 5575.27.

4. Escreva um programa que receba 6 parâmetros, a saber: N_1, N_2, N_3, N_4, N_5 e N_6 . O programa deve imprimir "CRESCENTE" se eles estiverem em ordem crescente e deve imprimir "ESTRITAMENTE CRESCENTE" se eles estiverem em ordem estritamente crescente. Uma sequência de números está em ordem crescente se e somente se $N_i \leq N_j \forall i < j$. Uma sequência de números está em ordem estritamente crescente se e somente se $N_i < N_j \forall i < j$. Assim, por exemplo, se a sequência for 6 7 7 7 7, o programa deve imprimir CRESCENTE. Se for 1 2 3 4 5 6 deve imprimir CRESCENTE e ESTRITAMENTE CRESCENTE. Se for 1 2 3 4 5 4 não deve imprimir nada.
5. Um número primo é chamado Primo Pitagoreano se e somente se ele pode ser escrito como a soma de 2 quadrados. Por exemplo, 13 é um primo pitagoreano porque $13 = 2^2 + 3^2 = 4 + 9 = 13$. Escreva um programa que receba um parâmetro N e imprima "SIM" se N for um primo pitagoreano e "NAO" senão. Note que se N nem primo é, a resposta é "NAO".
6. Escreva um programa que receba um parâmetro N inteiro e positivo e imprima "PERFEITO" ou "NAO PERFEITO" conforme o valor de N . Um número k é perfeito quando ele é igual à soma dos seus divisores próprios. Note que o único divisor impróprio de um número é o próprio número. Assim, por exemplo, 6 é perfeito pois seus divisores próprios são $3+2+1=6$.
7. Escreva um programa que receba dois parâmetros D e M inteiros e positivos, o primeiro representando um dia e o segundo um mês do ano de 2016. A data é válida, isso não precisa ser testado. O programa deve imprimir quantos dias se passaram desde 1/jan/2016. Por exemplo, se os números forem 17 5, o programa deve imprimir 108.
8. Escreva um programa que receba 1 parâmetro N inteiro e positivo. O programa deve calcular e imprimir a quantidade de divisores próprios de N . Os divisores próprios de N são todos os inteiros MENORES que N que o dividem sem deixar resto. Assim, por exemplo, para $N=20$, o programa deve responder 22, que é a soma $1+2+4+5+10=22$.

9. Escreva um programa que receba 3 parâmetros inteiros e positivos. O programa deve imprimir o maior deles. Assim, por exemplo, se os parâmetros forem 10, 12 e 13, o programa deve imprimir 13. Se forem 5, 4 e 5, deve imprimir 5 e se forem 3, 3 e 3 deve imprimir 3.

10. Escreva um programa que receba um parâmetro A indicando o ano de nascimento de uma pessoa. O programa deverá indicar qual a idade dessa pessoa no último dia do ano presente. Deve também imprimir "PR" se ela puder ser presidente da república do Brasil e imprimir "DEP" se ela puder ser deputada federal nesse dia. As regras para isso são: *no Brasil, o presidente deve ter 35 anos ou mais, e o deputado federal deve ter 21 anos ou mais*. Por exemplo, se o sujeito nasceu em 1999, em 31/12/2014 ela terá 14 anos e o programa deve responder 14. Se $A = 1966$ o programa deve responder 49 PR DEP, e se $A = 1985$ deve responder 30 DEP.

11. Escreva um programa que receba um parâmetro N inteiro e positivo e calcule e imprima o valor S real, assim calculado

$$S = 1\frac{1}{2} + 2\frac{1}{3} + 3\frac{1}{4} + \dots + n\frac{1}{n+1}$$

Note que S tem N parcelas. Por exemplo, se $N = 3$ a resposta do programa deve ser 1.916666667. Se $N = 6$ o programa deve imprimir 4.407142857.

12. Escreva um programa que receba 6 parâmetros, a saber: N_1, N_2, N_3, N_4, N_5 e N_6 . O programa deve imprimir "DECRESCENTE" se eles estiverem em ordem decrescente e deve imprimir "ESTRITAMENTE DECRESCENTE" se eles estiverem em ordem estritamente decrescente. Uma sequência de números está em ordem crescente se e somente se $N_i \geq N_j \forall i < j$. Uma sequência de números está em ordem estritamente crescente se e somente se $N_i > N_j \forall i < j$. Assim, por exemplo, se a sequência for 7 7 7 7 7 6, o programa deve imprimir DECRESCENTE. Se for 6 5 4 3 2 1 deve imprimir DECRESCENTE e ESTRITAMENTE DECRESCENTE. Se for 5 4 3 2 1 não deve imprimir nada.

13. Escreva um programa que receba seis parâmetros $A_1, A_2, \dots, A_6$ reais calcule e imprima a média e o desvio padrão desses seis números. A média é $\sum n \div 6$ e o desvio padrão é calculado usando-se a seguinte regra:

- ache a média
- subtraia cada número da média
- eleve ao quadrado o número anterior
- some os quadrados
- divida o número anterior por 5
- ache a raiz quadrada do número anterior.

Por exemplo, se os números forem 6 7 5.5 4 10 8, a média é 6.75 e o desvio padrão é 2.09.

Observação importante. Não use o cabeçalho `main()` seguido por `printf()` e `scanf()`. Agindo assim você não estará usando parâmetros, que é uma exigência desta folha. Leia a teoria logo acima, se não entendeu a razão desta observação.



- 1 - /

Problemas da folha 071c

- Escreva um programa que receba um parâmetro N inteiro e positivo e imprima todos os múltiplos de N que são menores do que 100. Assim, por exemplo, se $N=49$, o programa deve imprimir 49 e 98. Se $N=102$, o programa não deve imprimir nada e se $N=7$, deve imprimir 7 14 21 28 35 42 49 56 63 70 77 84 91 98.
- Escreva um programa que receba um parâmetro N inteiro e positivo e imprima todos os múltiplos de 17 que são menores do que 200 e maiores do que N . Assim, por exemplo, se $N=49$, o programa deve imprimir 51 68 85 102 119 136 153 170 187. Se $N=102$, o programa deve imprimir 119 136 153 170 187 e se $N=7$, deve imprimir 17 34 51 68 85 102 119 136 153 170 187.
- Escreva um programa que receba 2 parâmetros inteiros e positivos A e B e imprima todos os números múltiplos de 13 que são maiores ou iguais a A e menores ou iguais a B . Assim, por exemplo, se $A=10$ e $B=50$, o programa deve imprimir 13 26 39. Se $A=50$ e $B=200$, deve imprimir 52 65 78 91 104 117 130 143 156 169 182 195. Se $A=40$ e $B=30$ não deve imprimir nada.
- Escreva um programa que receba 2 parâmetros: o primeiro é um número inteiro P e o segundo é um número real H . O primeiro indica o peso de uma pessoa (em kg) e o segundo sua altura (em m). O programa deve calcular e imprimir o índice de Massa Corporal (IMC) que mede a obesidade de uma pessoa. O cálculo é $IMC = P \div H^2$. Por exemplo, se $P=112$ e $H=1.79$ o IMC deve ser 34.95
- Escreva um programa que receba 3 parâmetros inteiros e positivos. O programa deve imprimir o maior deles. Assim, por exemplo, se os parâmetros forem 10, 12 e 13, o programa deve imprimir 13. Se forem 5, 4 e 5, deve imprimir 5 e se forem 3, 3 e 3 deve imprimir 3.
- Escreva um programa que receba 1 parâmetro N inteiro e positivo. O programa deve calcular e imprimir a quantidade de divisores próprios de N . Os divisores próprios de N são todos os inteiros MENORES que N que o dividem sem deixar resto. Assim, por exemplo, para $N=20$, o programa deve responder 22, que é a soma $1 + 2 + 4 + 5 + 10 = 22$.
- Escreva um programa que receba 2 parâmetros: o primeiro é um número real V e o segundo é um número inteiro J . O primeiro indica o valor em reais e centavos de uma determinada compra. O segundo indica a modalidade de pagamento dessa mesma compra, podendo ser $J=1$ indicando compra em dinheiro, $J=2 \rightarrow$ cartão de débito, $J=3 \rightarrow$ cartão de crédito, $J=4 \rightarrow$ cheques pré-datados para compras em 5 vezes. O programa deve calcular o valor real a cobrar, a depender da modalidade de pagamento, sujeito às seguintes regras. Para pagamento em dinheiro, dar um desconto de 15%. Para cartão de débito o desconto é 5%. Para cartão de crédito, o valor é o preço da etiqueta e para compras parceladas em 5X, o valor é acrescido de 10%. Por exemplo, a chamada ao programa com $V=110.00$ e $J=3$ deve imprimir 110.00. A chamada com 200.00 4 deve imprimir 220.00. A chamada com 100.00 1 deve imprimir 85.00.
- Escreva um programa que receba um parâmetro A indicando o ano de nascimento de uma pessoa. O programa deverá indicar qual a idade dessa pessoa no último dia do ano presente. Deve também imprimir "PR" se ela puder ser presidente da república do Brasil e imprimir "DEP" se ela puder ser deputada federal nesse dia. As regras para isso são: *no Brasil, o presidente deve ter 35 anos ou mais, e o deputado federal deve ter 21 anos ou mais*. Por exemplo, se o sujeito nasceu em 1999, em 31/12/2014 ela terá 14 anos e o programa deve responder 14. Se $A = 1966$ o programa deve responder 49 PR DEP, e se $A = 1985$ deve responder 30 DEP.
- Escreva um programa que receba um parâmetro N inteiro e positivo e calcule e imprima o valor S real, assim calculado

$$S = 1\frac{1}{2} + 2\frac{1}{3} + 3\frac{1}{4} + \dots + n\frac{1}{n+1}$$

Note que S tem N parcelas. Por exemplo, se $N = 3$ a resposta do programa deve ser 1.916666667. Se $N = 6$ o programa deve imprimir 4.407142857.

- Escreva um programa que receba um parâmetro N inteiro e positivo e calcule e imprima o valor S real, assim calculado

$$S = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \dots + \frac{n}{n+1}$$

Note que S tem N parcelas. Por exemplo, se $N = 4$ a impressão deve ser 2.716666667 e se $N = 6$ deve ser 4.407142857

- Escreva um programa que receba um parâmetro N inteiro e positivo e calcule e imprima o valor S real, assim calculado

$$S = \frac{2}{1} + \frac{3}{4} + \frac{4}{9} + \dots + \frac{n}{n-1^2}$$

Note que S tem N parcelas. Por exemplo, se $N = 4$ a impressão deve ser 3.506944444 e se $N = 6$ deve ser 3.941388889

- Escreva um programa que receba um parâmetro N inteiro e positivo e imprima "PERFEITO" ou "NAO PERFEITO" conforme o valor de N . Um número k é perfeito quando ele é igual à soma dos seus divisores próprios. Note que o único divisor impróprio de um número é o próprio número. Assim, por exemplo, 6 é perfeito pois seus divisores próprios são $3+2+1=6$.
- Escreva um programa que receba um parâmetro N inteiro e positivo e imprima "PAR" ou "IMPAR" conforme o valor de N . Assim, por exemplo, 6 é par e 7 é ímpar.
- Escreva um programa que receba três parâmetros A , B e C , inteiros e imprima todos os termos de uma Progressão Aritmética (PA) na qual o termo inicial é A , a razão é B e o número de termos é C . Assim, por exemplo, se os parâmetros forem 12 3 4 o programa deve imprimir 12 15 18 21.
- Escreva um programa que receba três parâmetros A , B e C , inteiros e imprima todos os termos de uma Progressão Geométrica (PG) na qual o termo inicial é A , a razão é B e o número de termos é C . Assim, por exemplo, se os parâmetros forem 12 3 4 o programa deve imprimir 12 36 108 324.
- Escreva um programa que receba oito parâmetros $A_1, A_2 \dots A_8$, reais e calcule a média bimestral de uma suposta disciplina na qual o valor da prova (primeiro parâmetro ou A_1) tem peso 40% e a média dos outros 7 trabalhos (de A_2 até A_8) tem peso 60%. O programa deve imprimir a média bimestral com pelo menos uma casa decimal. Assim, por exemplo, se os parâmetros forem 8.5 10 10 5 0 3 10 7 o programa deve imprimir 7.25.
- Escreva um programa que receba um número indeterminado de parâmetros $A_1, A_2 \dots A_n$, com $n \leq 8$, reais e calcule a média bimestral de uma suposta disciplina na qual o valor da prova (primeiro parâmetro ou A_1) tem peso 40% e a média dos outros trabalhos (de A_2 até A_n) tem peso 60%. O programa deve imprimir a média bimestral com pelo menos uma casa decimal. Note que a quantidade de notas pode ser obtida via recepção de parâmetros da função *main*. Assim, por exemplo, se os parâmetros forem 8.5 10 10 5 o programa deve imprimir 8.40.
- Escreva um programa que receba um parâmetro que pode ser 1, 2 ou 3. Nos 3 casos o programa deve emitir uma tabela de funções trigonométricas (1=seno, 2=cosseno e 3=tangente). Nos 3 casos a tabela deve prever ângulos variando de 1° começando em 0° e terminando em 90° . Eis como ficaria o começo da tabela se o programa fosse chamado com parâmetro igual a 1.

0	0.0000000000
1	0.0174524064
2	0.0348994967
3	0.0523359562

- Escreva um programa que receba dois parâmetros M e N , inteiros e positivos, ache e imprima a soma dos múltiplos de M que são menores do que N . Então, por exemplo, se $M = 7$ e $N = 70$, o programa deve imprimir 14 21 28 35 42 49 56 63. Note que o próprio M não integra a saída.

20. Escreva um programa que receba seis parâmetros $A_1, A_2, \dots, A_6$ reais calcule e imprima a média e o desvio padrão desses seis números. A média é $\sum n \div 6$ e o desvio padrão é calculado usando-se a seguinte regra:
- ache a média
 - subtraia cada número da média
 - eleve ao quadrado o número anterior
 - some os quadrados
 - divida o número anterior por 5
 - ache a raiz quadrada do número anterior.
- Por exemplo, se os números forem 6 7 5.5 4 10 8, a média é 6.75 e o desvio padrão é 2.09.
21. Escreva um programa que receba dois parâmetros D e M inteiros e positivos, o primeiro representando um dia e o segundo um mês do ano de 2014. A data é válida, isso não precisa ser testado. O programa deve imprimir quantos dias se passaram desde 1/jan/2014. Por exemplo, se os números forem 17 5, o programa deve imprimir 107.
22. Escreva um programa que receba dois parâmetros D e M inteiros e positivos, o primeiro representando um dia e o segundo um mês do ano de 2014. A data é válida, isso não precisa ser testado. O programa deve imprimir quantos dias faltam até 31/dez/2014. Por exemplo, se os números forem 17 5, o programa deve imprimir 258.
23. Escreva um programa que receba dois parâmetros D e M inteiros e positivos, o primeiro representando um dia e o segundo um mês do ano de 2016. A data é válida, isso não precisa ser testado. O programa deve imprimir quantos dias se passaram desde 1/jan/2016. Por exemplo, se os números forem 17 5, o programa deve imprimir 108.
24. Suponha que existe um processo industrial que tem uma duração de tempo determinado e conhecem-se as horas de início e fim do mesmo. Você deve escrever um programa que receba 4 parâmetros, a saber: H_i, M_i, H_f e M_f , respectivamente hora e minuto de início e hora e minuto de fim. Seu programa deve calcular e imprimir a duração do processo em minutos. Por exemplo, se os parâmetros forem 10 50 11 16, o programa deve imprimir 26.
25. Escreva um programa que receba um único parâmetro N , inteiro e positivo e imprima "SIM" se N for um número primo e "NAO" se ele não for primo.
26. Escreva um programa que receba um único parâmetro N , inteiro e positivo e imprima "SIM" se N for um número par E múltiplo de 3 E múltiplo de 5. Deve imprimir "NAO" se ele não atender a uma ou mais dessas condições.
27. Escreva um programa que receba 6 parâmetros, a saber: N_1, N_2, N_3, N_4, N_5 e N_6 . O programa deve imprimir "CRESCENTE" se eles estiverem em ordem crescente e deve imprimir "ESTRITAMENTE CRESCENTE" se eles estiverem em ordem estritamente crescente. Uma sequência de números está em ordem crescente se e somente se $N_i \leq N_j \forall i < j$. Uma sequência de números está em ordem estritamente crescente se e somente se $N_i < N_j \forall i < j$. Assim, por exemplo, se a sequência for 6 7 7 7 7 7, o programa deve imprimir CRESCENTE. Se for 1 2 3 4 5 6 deve imprimir CRESCENTE e ESTRITAMENTE CRESCENTE. Se for 1 2 3 4 5 4 não deve imprimir nada.
28. Escreva um programa que receba 6 parâmetros, a saber: N_1, N_2, N_3, N_4, N_5 e N_6 . O programa deve imprimir "DECRESCENTE" se eles estiverem em ordem decrescente e deve imprimir "ESTRITAMENTE DECRESCENTE" se eles estiverem em ordem estritamente decrescente. Uma sequência de números está em ordem decrescente se e somente se $N_i \geq N_j \forall i < j$. Uma sequência de números está em ordem estritamente decrescente se e somente se $N_i > N_j \forall i < j$. Assim, por exemplo, se a sequência for 7 7 7 7 6, o programa deve imprimir DECRESCENTE. Se for 6 5 4 3 2 1 deve imprimir DECRESCENTE e ESTRITAMENTE DECRESCENTE. Se for 4 5 4 3 2 1 não deve imprimir nada.
29. Escreva um programa que receba um parâmetro N , inteiro e positivo, calcule e imprima o N -ésimo número na sequência de Fibonacci. Define-se a sequência de Fibonacci como $F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, \dots$. Assim, por exemplo, se o parâmetro for 13, deve ser impresso 233.
30. Escreva um programa que receba um parâmetro N , inteiro e positivo e menor que 14. O programa deve calcular e imprimir o fatorial de N . Define-se o fatorial como $n! = n \times (n-1) \times (n-2) \times \dots \times 2$. Assim, por exemplo, se o parâmetro for 13, deve ser impresso 6227020800.
31. Escreva um programa que receba um parâmetro R real e positivo e imprima o volume da esfera de raio R . Use a fórmula

$$V = \frac{4}{3} \times \pi \times R^3$$

Assim, por exemplo se o parâmetro for 11, deve ser impresso 5575.27.

32. Escreva um programa que receba um parâmetro R real e positivo e imprima a área da superfície esférica da esfera de raio R . Use a fórmula

$$S = 4 \times \pi \times R^2$$

Assim, por exemplo se o parâmetro for 11, deve ser impresso 1520.53.

33. Escreva um programa que receba um parâmetro R real e positivo e imprima a área da superfície do círculo de raio R . Use a fórmula

$$S = \pi \times R^2$$

Assim, por exemplo se o parâmetro for 11, deve ser impresso 380.13.

34. Escreva um programa que receba um parâmetro R real e positivo e imprima o comprimento da circunferência de raio R . Use a fórmula

$$C = 2 \times \pi \times R$$

Assim, por exemplo se o parâmetro for 11, deve ser impresso 69.11.

35. Imagine uma casquinha de sorvete deliciosa e geometricamente perfeita. A parte de baixo é um cone de baquinha. A parte de cima é um hemisfério. Para descrever completamente a casquinha precisa-se saber o raio do cone (que é igual ao raio do hemisfério) e a altura do cone. Note-se que o conjunto está repleto de maravilhoso sorvete de morango. Escreva um programa que receba dois parâmetros R e H , representando o raio e a altura do cone, ambos medidos em cm. O programa deve calcular e imprimir a quantidade de sorvete que existe na casquinha em cm^3 .

Para sua referência eis as relações importantes:

volume do cone = $\frac{1}{3}$ do volume do cilindro de mesma altura

volume do cilindro = altura $\times$ área da base

área da base = $2 \times \pi \times R^2$

volume do hemisfério = $\frac{1}{2}$ do volume da esfera de mesmo raio.

36. Sabendo que qualquer primo maior do que 2 pode ser escrito como a diferença de 2 quadrados de números inteiros vizinhos, escreva um programa que receba N (que é primo maior do que 2, não é preciso testar isto) e imprima os 2 números que quando elevados ao quadrado e subtraídos reproduzem o primo original. Por exemplo, $101 = 51^2 - 50^2$.
DICA: estude a fórmula $(n+1)^2 - n^2 = 2n+1 = n+n+1$

37. Um número primo é chamado Primo Pitagoreano se e somente se ele pode ser escrito como a soma de 2 quadrados. Por exemplo, 13 é um primo pitagoreano porque $13 = 2^2 + 3^2 = 4 + 9 = 13$. Escreva um programa que receba um parâmetro N e imprima "SIM" se N for um primo pitagoreano e "NAO" senão. Note que se N nem primo é, a resposta é "NAO".
38. Imagine um relógio de ponteiros trabalhando na modalidade discreta: Durante 60 segundos inteiros os ponteiros permanecem fixos nas suas posições. Ao virar o minuto, os ponteiros andam ambos para suas novas posições. Como se sabe, os ponteiros têm centro comum, logo eles formam um determinado ângulo entre eles. Escreva um programa que receba 2 parâmetros: H_1 e M_1 , calcule e imprima o menor ângulo em graus formado pelos dois ponteiros. Por exemplo, se os parâmetros forem 12 10, o programa deverá imprimir 55.
39. Escreva um programa que receba um parâmetro real N e imprima-o arredondado para 1 casa. Estude os exemplos: arred(1.99)=2.0; arred(1.49)=1.5; arred(1.75)=1.8; arred(1.749)=1.7.
40. Um criptógrafo bolou um esquema onde cada letra é representada pela inversão de sua posição no alfabeto. Assim, A=1, B=2, C=3, ... Z=26 e invertendo isso fica A=10, B=20, C=30, ... Z=62. Escreva um programa que receba como parâmetros 5 inteiros que representam letras criptografadas como aqui descrito e imprima a mensagem decriptografada. Por exemplo se o programa receber 62 50 20 81 10, deve imprimir "ZEBRA".

Iniciando na programação

Citando Aristóteles, *Tudo o que precisamos aprender, aprendemos fazendo* (no seu livro *Ética*), vamos concordar que para se desenvolver na habilidade de programar computadores, o que é necessário fazer é programar. Na verdade, nada é muito diferente em muitas outras habilidades humanas: cirurgia, conserto de veículos, alta gastro-nomia, tocar piano ou qualquer outro instrumento, escrever bem, desenhar maravilhosamente, erguer uma parede, ... a lista é interminável.

Dito isto, a seguir vem uma lista de programas que você deve produzir. A linguagem escolhida pode ser qualquer uma:

C durante muito tempo a linguagem usada em universidades e escolas para aprender

C++ uma evolução de C, sobretudo para simplificar as entradas e saídas de dados

Python a nova queridinha no processo de aprender a programar. É a que garante maior produtividade sobretudo no início do aprendizado

Javascript um bocadinho mais complicada, por ter que usar um browser durante sua execução, mas tem a vantagem de permitir um visual GUI bem elegante e fácil (por conta da dupla HTML/CSS)

APL meio antiga, mas insuperável para construir programas pequenos embora muito poderosos

Java até pouco tempo a preferida dos desenvolvedores, roda em qualquer coisa aparentada ao computador

Outras Assembler, Octave, Maple, Pascal, Fortran, Lisp, Prolog, ... a lista é quase interminável.

Enfim, a lista acima apareceu para subsidiar o comentário de que **não importa a linguagem escolhida** o algoritmo é sempre o mesmo ou muito parecido. E, o algoritmo é sempre o mais difícil de produzir. Então, de fato, a linguagem utilizada é somente importante.

Em resumo, use o que quiser e dentro da linguagem escolhida use o recurso que quiser. O importante é achar o resultado:

Programa 1: Múltiplos de um número

Se listarmos todos os números naturais abaixo de 10 que são múltiplos de 3 ou 5, teremos:

3, 5, 6 e 9

A soma desses números é 23.

Resolvendo o caso

para uma instância real. Suponhamos $a = 3$ e $b = 5$ e $limite = 1000$. Uma maneira simples de resolver o caso é:

```
limite = 1000 - 1
soma = 0
para i de 1 até limite passo 1
    se (i%3==0) || (i%5==0)
        soma=soma+i
fim{se}
fim{para}
escreva(soma)
```

Este programa deve imprimir 233.168 que é a soma pedida.

O próximo passo é deixar o programa genérico com as variáveis a , b e $limite$ sendo lidas no início do processamento.

Teste o seu programa com as seguintes triplas

a=7, b=11, lim=2600, resposta=747741
a=3, b=17, lim=3000, resposta=1676031

Desafio Se você for questionado sobre um limite de 1.000.000.000 este programa pode demostrar um tanto. Uma estratégia um pouco mais eficiente, pode ser obter a soma dos divisíveis por a menores que $limite$ somado com a soma dos divisíveis por b (menores que $limite$). Só que fazendo isso, você somou os divisíveis por $a \times b$ duas vezes. Vamos olhar o exemplo para $n = 3$ Precisamos somar $3 + 6 + 9 + 12 + \dots + 999$ que é igual a $3 \times (1 + 2 + 3 + 4 + \dots + 333)$.

Para $n = 5$ precisamos somar $5 + 10 + 15 + \dots + 995 = 5 \times (1 + 2 + 3 + \dots + 199)$. Note que $199 = 995/5$, mas também $999/5$ em termos de divisão inteira. Se também notarmos que $1 + 2 + 3 + \dots + p = \frac{1}{2} \times p \times (p + 1)$, temos:

Se você definir uma função chamada

```
função somds(int n, int lim)
    limite = lim-1
    p = limite / n // (divisao inteira)
    retorne n * (p * (p+1)) / 2
fim{função}
```

O cálculo agora é fácil, bastando fazer:

```
somds(3,1000)+somds(5,1000)-somds(15,1000)
```

Programa 2: Números primos

Um número primo é um número natural maior que 1 que possui apenas dois divisores positivos: 1 e ele mesmo.

Em outras palavras, um número primo não pode ser formado pela multiplicação de outros números naturais menores.

Exemplos de números primos: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, ...

Propriedades importantes dos números primos:

O número 1 não é primo: Ele possui apenas um divisor (ele mesmo). Todos os números pares maiores que 2 são compostos: Eles podem ser formados pela multiplicação de dois números menores (por exemplo, $8 = 2 \times 4$). A distribuição dos números primos é infinita: Não existe um limite para a quantidade de números primos que existem. Os números primos desempenham um papel fundamental na teoria dos números e em diversas áreas da matemática e da computação: Eles são utilizados na criptografia, na construção de códigos e na decomposição de números em fatores primos. Especificamente na área da computação, eles são a base de geração de pseudo-aleatórios, na construção de tabelas e códigos *hash*, na maioria dos algoritmos de criptografia (RSA, El Gamal, elíptica, etc).

Métodos para encontrar números primos: Básico: contar os divisores do número. Se forem apenas 2 o número (inteiro e maior que 1) é primo. Se forem mais, ele é composto. Crivo de Eratóstenes: Um método antigo e eficiente para encontrar números primos menores que um determinado valor. Teste de Miller-Rabin: Um algoritmo probabilístico para determinar se um número é primo ou composto. Outros métodos: É moda procurar estes algoritmos. Se alguém achar um que seja determinístico e de complexidade menor que exponencial, seu nome vai para a história.

Curiosidades: O maior número primo conhecido: Em dezembro de 2023, o maior número primo conhecido é $2^{(82,589,933)} - 1$, com 25.097.524 dígitos decimais. Os números primos gêmeos: São dois números primos consecutivos, como (3, 5) e (11, 13). Os números primos palíndromos: São números primos que lêem iguais em ambas as direções, como (2, 5, 7, 11, 131).

O algoritmo que você deve implementar é

```
leia (a,b)
ct = 0
enquanto (a<=b)
    var=2
    enquanto (var<a)
        se a%var == 0
            saia // a é composto
        senão
            var = var + 1
    fim{se}
    fim{enquanto}
    se var == a
        ct=ct+1 // a é primo
    fim{se}
    a=a+1
fim{enquanto}
escreva(ct)
```

Para testar seu programa, use considere que entre 2 e 100 há 25 primos. Outras opções de teste:

Li=2780, Ls=8777, Quantidade=689
Li=6055, Ls=7617, Quantidade=177

Programa 3: Sequência de Fibonacci

A Sequência de Fibonacci, também conhecida como Sequência de Leonardo, é uma série de números inteiros positivos, começando normalmente por 0 e 1, na qual cada termo subsequente é a soma dos dois anteriores.

Fórmula: O termo n da sequência de Fibonacci pode ser calculado pela fórmula: $F_n = F_{n-1} + F_{n-2}$ Onde: F_n é o termo n da sequência, F_{n-1} é o termo anterior ($n - 1$) da sequência e F_{n-2} é o termo anterior ($n - 2$) da sequência.

Os primeiros termos da sequência são: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... O quinto termo da sequência (F_5) é igual a 5, pois $F_5 = F_4 + F_3 = 3 + 2 = 5$. O décimo termo da sequência (F_{10}) é igual a 55, pois $F_{10} = F_9 + F_8 = 34 + 21 = 55$.

A razão entre dois números consecutivos da sequência tende para o Número Áureo, representado pela letra grega ϕ (fi). O Número Áureo é um número irracional aproximadamente igual a 1,6180339887... A sequência de Fibonacci está presente em diversos elementos da natureza, como na disposição das pétalas de algumas flores, nas ramificações de árvores e na estrutura de conchas. A sequência de Fibonacci tem diversas aplicações em matemática, computação, arte e arquitetura.

O programa que você deve construir é para obter a soma dos números da sequência de Fibonacci que são PARES e menores ou igual a um determinado limite dado.

Por exemplo, se o limite for 100, a sequência é 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 e a soma buscada é $2 + 8 + 34 = 44$.

O algoritmo em questão é

```
soma=0
leia(limite)
a=1
b=2
soma=2
c=3
enquanto (c<=limite)
    a=b
    b=c
    c=a+b
    se c%2==0
        soma=soma+c
    fim{se}
fim{enquanto}
escreva(soma)
```

Para testar seu programa use estes casos

Limite=121000, Soma=60696
Limite=1777242, Soma=1089154

👉 Para você fazer

- Encontre a soma de todos os múltiplos de 13 ou 2 abaixo de 3400
- Descubra quantos primos existem entre 7085 e 7097 (incluídos os extremos).
- O limite da sua implementação para os números de Fibonacci (soma dos pares) é 67972

Responda aqui:

1	2	3



Iniciando na programação - II

Mais 3 programas para você tentar melhorar sua habilidade na programação.

Programa 1: Que dia do ano é hoje

? Nosso calendário não é uma obra da racionalidade e da exatidão. Talvez por ter quase 500 anos, sem nunca ter sido alterado e também por depender da duração de um ano, um número para lá de irregular (365,2422 dias).

O programa que você deve construir pergunta o dia e o mês de uma determinada data e responde com o dia do ano dessa data. Considere que o ano em questão não é bissexto. Assim, acompanhe no exemplo:

```
se informar 10/01 - o programa responde 10
se informar 3/02 - 34
se informar 31/12 - 365
se informar 12/10 - 285
```

Uma solução para este problema é criar um vetor de 12 inteiros com os dias de cada mês. No caso

M = 31,28,31,30,31,30,31,31,30,31,30,31

Agora, um segundo vetor onde cada elemento será a soma de todos os valores de M localizados ANTES de cada elemento

MA = 0,31,59,90,120,151,181,212,243,273,304,334

Agora, para achar o dia anual, realiza-se a seguinte operação

$$d_{anual} = MA[mes] + dia$$

Acompanhe os exemplos: 10/01 → 0 + 10 = 10, 3/2 → 31 + 3 = 34, 31/12 → 334 + 31 = 365 e 12/10 → 273 + 12 = 285
O algoritmo pode ser

```
M=31,28,31,30,31,30,31,31,30,31,30,31
MA=0,0,0,0,0,0,0,0,0,0,0,0
para i de 2 ate 12
    MA[i]=M[i-1]+MA[i-1]
fim{para}
leia (dia,mes)
escreva (MA[mes]+dia)
```

Teste o seu programa com as seguintes datas

dia 13 do mês 11 é o dia 317
dia 27 do mês 5 é o dia 147

Programa 2: Sequência de Collatz

Existe uma sequência interativa que é definida para o conjunto dos positivos:

$$n \rightarrow \frac{n}{2} \quad \text{se } n \text{ par}$$

$$n \rightarrow 3n + 1 \quad \text{se } n \text{ impar}$$

Usando a regra acima e começando com o número 13, gera-se a seguinte sequência:

13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1.

Pode-se ver que esta sequência (começando em 13 e terminando em 1) tem 10 termos. Ainda que não se tenha provado (este é a conjectura de Collatz) estima-se que independente do número inicial escolhido, o resultado final sempre será 1.

O programa que você deve construir deve descobrir qual o número inicial (abaixo de um certo limite) produz a cadeia mais longa. Em outras palavras, seu programa deve ler o limite e escrever o número inicial que gera a cadeia mais longa. Eis um possível algoritmo

```
maior = -99999
leia(lim)
para j de 1 ate lim-1
    qt=1
    i=j
    enquanto i != 1
        se i%2 == 0
            i=i/2
        senão
            i = 3*i + 1
        fim{se}
        qt = qt+1
    fim{enquanto}
    se qt > maior
        jsalvo = j
        maior = qt
    fim{se}
fim{para}
escreva (jsalvo)
```

Algumas observações:

- 1. Se programar em C++, as variáveis precisam ser long long, pois pode haver (houve) a quebra do limite de 32 bits.
- 2. Se programar em Python, pode usar o conceito de dicionário, para evitar ficar recalculando o que já foi calculado. Em média esta ideia baixa o processamento em 90%. Veja

```
limi=1000000
def collatz(n):
    value = colla.get(n)
    if value != None:
        return value
    else:
        if n%2==0:
            colla[n]=1+collatz(n/2)
        else:
            colla[n]=collatz(3*n+1)
        return colla[n]
maior=-99999
respo=0
colla={1:1}
i=limi//2 ou chao(limi/2)
while i<=limi:
    if collatz(i)>maior:
        maior=collatz(i)
        respo=i
    i=i+1
print(respo)
```

- 3. Este exemplo é um convite ao uso de recursividade. Fique à vontade
- 4. Este problema é o número 14 do projeto Euler.

Teste seu programa com estes casos

no limite 20800 o laço mais longo tem 17647
no limite 25032 o laço mais longo tem 23529

Programa 3: Achar o n-ésimo primo

A listagem dos 6 primeiros números primos é 2, 3, 5, 7, 11 e 13. Pode-se ver que o 6º número primo é 13. Qual é o j-ésimo primo? Na folha passada (vivx072b) já vimos este código. Ele pode ser implementado seguindo na sequência dos naturais e a cada natural que for primo, incrementa-se um dado contador. Quando este cpontador for igual a j, basta imprimir o natural em exame. Mais ou menos assim

```
i = 3
ct = 1
leia (j)
enquanto j != ct
    var = 2
    enquanto var < i
        se i%var == 0
            saia
        senão
            var = var + 1
    fim{se}
    fim{enquanto}
    se var == i
        ct = ct + 1
    fim{se}
    i = i + 1
fim{enquanto}
escreva (i-1)
```

Mas, há algumas considerações, que permitem melhorar o desempenho deste programa.

- Como está, há 2 laços aninhados (um enquanto dentro de outro enquanto), o que proporciona um desempenho terrível. Conhecido na teoria da complexidade como $O(n^2)$.
- Da teoria de divisibilidade, sabe-se que os divisores vem aos pares, e a multiplicação dos elementos do par entre si, reproduz o valor sob análise. Não me acredite, faça a experiência com qualquer inteiro. O primeiro elemento do par começa em 1 e vai até $\sqrt{n}$ e o segundo elemento do par começa em $\sqrt{n}$ e vai até n . Daqui, a busca por divisor só precisa ir até $\sqrt{rtn}$ e não até n o que representa um ganho importante.
- Qualquer primo maior que 3 pode ser representado pela expressão $p = 6k \pm 1$ com $k = 1, 2, 3, \dots$
- Todos os primos exceto 2 são impares

Juntando todos estes fatos e fazendo uso do importante conceito de função, eis como pode ficar o código

```
função primo(x)
se x=1
    retorne falso
senão
    se x<4
        retorne verdadeiro
    senão
        se x%2 == 0
            retorne falso
        senão
            se x<9
                retorne verdadeiro
            senão
                se x%3 == 0
                    retorne falso
                senão
                    r = chão(sqrt(x))
                    f = 5
                    enquanto f <= r
                        se x%f == 0
                            retorne falso
                    fim{se}
                    se x%(f+2) == 0
                        retorne falso
                    fim{se}
                    f = f+6
                fim{enquanto}
            fim{se}
        fim{se}
    fim{se}
    retorne verdadeiro
fim{função}
```

```
leia (limite)
contador = 1
candidato = 1
repara
    candidato = candidato + 2
    se primo(candidato)
        contador = contador + 1
    fim{se}
até contador != limite
escreva (candidato)
```

Para testar seu programa use estes casos

0 115325-ésimo número primo é 1516217
0 48915-ésimo número primo é 597521

🔗 Para você fazer

- 1. dia 10 do mês 12 é qual dia ?
- 2. no limite 20800 qual o comprimento do laço mais longo ?
- 3. Qual o 146989-ésimo número primo ?

Responda aqui:

1	2	3



Iniciando na programação-III

Qual o maior fator ? o Teorema Fundamental da Aritmética (TFA) afirma que: todo número inteiro positivo maior que 1 pode ser decomposto em um produto único de números primos. Essa decomposição é única, a menos da ordem dos fatores. Em outras palavras, qualquer número inteiro maior que 1 pode ser escrito como um produto de números primos, e essa forma de escrita é única, desconsiderando a ordem dos fatores.

Exemplo:
O número 12 pode ser decomposto em $2 \times 2 \times 3$. Essa é a única forma de escrever 12 como um produto de números primos, desconsiderando a ordem dos fatores ($2 \times 3 \times 2$ também é a mesma decomposição).

Outro exemplo, 474 pode ser decomposto em $2 \times 3 \times 79$.
O TFA é um dos pilares da teoria dos números e tem diversas aplicações em matemática, criptografia e outras áreas.

Dado um número qualquer, o algoritmo que descobre sua decomposição em primos é:

```
long long int n,fator,ultfator
leia (n)
fator=2
enquanto n>1
    se n % fator == 0
        ultfator=fator
        n = n / fator
        enquanto n % fator == 0
            n = n / fator
        fim{enquanto}
        fim{se}
        fator = fator + 1
    fim{enquanto}
escreva (ultfator)
```

O exercício acima está baseado no problema 3 do projeto Euler. <https://projecteuler.net/>.

 Para você fazer

Os fatores primos de 13195 são 5, 7, 13 e 29.
Qual o maior fator primo do número 26126339399 ?

Qual a maior série de multiplicações ? Suponha um número grande de 100 dígitos. Quer-se saber qual é o maior valor obtido multiplicando 5 dígitos vizinhos. A estratégia é percorrer o vetor de dígito multiplicando 5 vizinhos e observando (e guardando) o maior valor. Ao final, este é o resultado esperado.

Uma dificuldade é dada pela conversão de um caractere em seu número. Lembrando que no código ASCII os números ocupam as posições de 48 a 57, acompanhe

número	0123456789

posições no	4455555555
cód. ASCII	8901234567

Para ler a tabela acima, considere que o caracter zero ocupa a posição 48 na tabela ASCII. O caracter '1' ocupa a posição 49 e assim por diante...

Vai daí, que a expressão $x-48$ converte o caracter x em seu deslocamento no código ASCII. Então, por exemplo '6'-48 nos dá como resposta 54 que é a posição do caracter 6 no ASCII. Para converter este valor inteiro no real valor do dígito há que subtrair 48. Então, a expressão que nos interessa é $(x-48)-48$ ou se você quiser ser mais direto $x-48$ também serve.

Eis o algoritmo:

```
string f="..."
maior = -999999
para i de 4 até 100
    n1 = f[i-4]-48
    n2 = f[i-3]-48
    n3 = f[i-2]-48
    n4 = f[i-1]-48
    n5 = f[i]-48
    produto = n1*n2*n3*n4*n5
    se produto > maior
        maior = produto
    fim{se}
fim{para}
escreva (maior)
```

O problema acima está baseado no problema 8 do projeto Euler. <https://projecteuler.net/>.

 Para você fazer

Dado um número enorme, deve-se achar o maior produto de 5 dígitos consecutivos. Considere o número a seguir formado por 100 dígitos:

30358907296290491560440772390713810515859307960866
701742427121883998797908792274921901699720888093776

Números triangulares e seus divisores A sequência de números triangulares é gerada pela adição dos números naturais. Assim, o 7º número triangular é a soma dos 7 primeiros números naturais ou seja $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$. Os 10 primeiros números triangulares são

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Criando uma lista dos fatores dos primeiros 7 números triangulares tem-se:

- 1: 1
- 3: 1,3
- 6: 1,2,3,6
- 10: 1,2,5,10
- 15: 1,3,5,15
- 21: 1,3,7,21
- 28: 1,2,4,7,14,28

Como se pode ver, 28 é o primeiro triangular que tem mais do que 5 divisores.

Um possível algoritmo para achar o primeiro trinagulos com mais divisores do que um dado limite, poderia ser este, ainda que ele seja terrivelmente ineficiente, sobretudo para 400 ou mais divisores...

```
t=1
a=1
cnt=0
enquanto cnt < 0_NUMERO_DE_DIVISORES
    cnt=0
    a=a+1
    t=t+a
    para i igual a 1 ate i menor ou igual a t
        se t % i == 0
            cnt = cnt + 1
        fim{se}
    fim{para}
    fim{enquanto}
    escreva (t)
```

 Para você fazer

Qual o primeiro triangular que tem mais do que 250 fatores ?

Este problema está baseado no problema 12 do projeto Euler (lá são 500 divisores).

Responda aqui

Qual o maior fator ? (problema 1)
Qual o maior produto de 5 dígitos ? (problema 2)
Qual o primeiro triangular ? (problema 3)

Iniciando na programação IV

Euler 500

Seja o seguinte problema, como descrito no Euler (problema 500)

The eight divisors of 24 are 1,2,3,4,6,8,12 and 24. The ten numbers not exceeding 100 having exactly eight divisors are 24, 30, 40, 42, 54, 56, 66, 70, 78 and 88. Let $f(n)$ be the count of numbers not exceeding n with exactly eight divisors. You are given $f(100) = 10$, $f(1000) = 180$ and $f(1000000) = f(10^6) = 224427$. Find $f(10^{12})$.

Força bruta

Uma estratégia bem **força bruta** pode ser

```
função dv(x)
  q=2
  i=2
  enquanto i<x
    se x%i==0
      q=q+1
    fim{se}
    i=i+1
  fimenquanto
  retorne q
fim{função}
```

```
função qv(limite)
  q=0
  i=1
  enquanto i<=x
    se dv(i)==8
      q=q+1
    fim{se}
    i=i+1
  fim{enquanto}
  imprima q
fim{função}
```

A função dv recebe um número x e testa todos os inteiros entre 2 e $x - 1$ para ver quantos divisores dividem x . Dá-se por definido que a quantidade de divisores já começa em 2 (a unidade e o próprio x). Esta função devolve a quantidade de divisores. Na sequência vem a função qv que também recebe um valor ($limite$) e testa todos os inteiros entre 1 e $limite$ contando para cada um a quantidade de inteiros nesse intervalo que tem 8 divisores. Note o funcionamento correto, de acordo com a definição original do Euler

	qv(100)
10	
	qv(1000)
180	

Só que – obviamente – esta função tem um desempenho medonho. É hora de fazer melhorias.

Força bruta 2

Começa-se verificando que os divisores sempre vem aos pares. Cada par, quando multiplicado, reproduz o número original. Por exemplo, os divisores de 10 são 1, 2, 5 e 10. Os pares são 1 e 10 e também 2 e 5. Que multiplicados reproduzem 10.

O primeiro elemento do par surge antes de $\sqrt{x}$, razão pela qual a busca pode começar em 1 e terminar em $\sqrt{x}$. Repare a melhora bruta no desempenho. Em vez de terminar o segundo loop em x , ele termina em $\sqrt{x}$ o que é muito melhor. Se não conseguir *enxergar* faça um teste até 10.000. No primeiro caso são 10000 iterações e no segundo caso são $\sqrt{10000} = 100$ iterações Surge aqui uma dificuldade quando x é um quadrado perfeito. Neste caso, ele forma par com ele mesmo. Agora o segundo elemento do par não deve ser contado (ele já foi contado como o primeiro elemento do par). Eis como fica

```
função dv2(limite)
  parada=sqrt(limite)
  q=2
  i=2
  se parada*parada == limite
    q=q-1
  fim{se}
  enquanto i<parada
    se limite%i == 0
      q=q+2
    fim{se}
    i=i+1
  fim{enquanto}
  return q
fim{função}
```

A função qv é a mesma, mas a chamada a dv deve ser substituída pela chamada a $dv2$. Funcionou, mas demorou longos minutos. Se formos pedir o que o problema original demanda que é $f(10^{12})$ simplesmente vai demorar meses...

Matemática, socorro !

Do Teorema fundamental da aritmética, qualquer inteiro maior que 1 pode ser obtido de maneira única multiplicando primos entre si. Acompanhe

2:	1 × 2
3:	1 × 3
4:	1 × 4 2 × 2
5:	1 × 5
6:	1 × 6 2 × 3
7:	1 × 7
8:	1 × 8 2 × 4
9:	1 × 9 3 × 3

E assim, por diante. Desta regra, queremos números inteiros cujos fatores primos possam ser organizados de exatamente 4 maneiras distintas levando a 4 pares ou 8 divisores. Fuçando na teoria dos primos, descubra-se que os números que têm 8 divisores são aqueles (e só aqueles) que têm uma das seguintes formas: (onde p_j é um primo qualquer).

- $p_j^7 \times 1 \times 1 \times 1$
- $p_j^3 \times p_k \times 1 \times 1$
- $p_j \times p_k \times p_m \times 1$

Para esta análise a tabela de primos (o crivo de eratóstenes) deve ir até $N//6$ (este 6 é obtido multiplicando os dois menores primos do crivo que são 2 e 3).

```
função primo(n)
  se n==1
    retorne Falso
  fim{se}
```

```
se n<4
  retorne Verdadeiro
fim{se}
se n%2 == 0
  retorne Falso
fim{se}
se n<9
  retorne Verdadeiro
fim{se}
se n%3 == 0
  retorne Falso
fim{se}
r=floor(sqrt(n))
f = 5
enquanto f <= r
  se (n%f)==0 || (n%(f+2))==0
    retorne Falso
  f=f+6
fim{enquanto}
retorne Verdadeiro
fim{função}
```

```
função crivo(ate)
  n=floor(ate/6)
  cr[n]=0
  indice=1
  i=2
  enquanto i<=ate
    se primo(i)
      cr[indice]=i
      indice=indice+1
    fim{se}
    i=i+1
  fim{enquanto}
  retorne cr
fim{função}
```

```
função euler500(lim)
  cr = crivo(lim)
  ctd=0
  para i de 1 ate tamanho(cr)
    se cr[i]^7 < lim
      ctd=ctd+1
  senao
    saia
  fim{se}
  fim{para}
  para i de 1 ate tamanho(cr)
    para j de i ate tamanho(cr)
      para k de j ate tamanho(cr)
        se i != j
          se cr[i]*cr[j]^3 < lim
            ctd = ctd+1
          senão
            saia
          fim{se}
          fim{apara}
        fim{para}
      para i de 1 ate tamanho(cr)
        para j de i ate tamanho(cr)
          para k de j ate tamanho(cr)
            se (i != j) && (i != k) \
              && (j != k)
              se cr[i]*cr[j]*cr[k]<lim
                ctd = ctd +1
            senão
              saia
            fim{se}
            fim{se}
          fim{apara}
        fim{para}
      fim{para}
    escreva (ctd)
  fim{função}
```

Euler 4

Este problema esta apenas baseado no Euler 4 (seja curioso, veja lá...). Ele pede para identificar quando um número é ou não é um palíndromo: Segundo o Gemini, *Um número palíndromo, também chamado de capicua, é um número inteiro positivo que se lê da mesma forma da esquerda para a direita e da direita para a esquerda.* O algoritmo base aqui pode ser:

```
função palindromo(x)
  w=x
  c=0
  i=(-1)+ceil(log10(x+1))
  enquanto (x != 0)
    c=c+(x%10)*pow(10,i)
    i=i-1
    x=x/10
  fim{enquanto}
  retorne c=w
fim{função}
```

A função *palindromo* retorna verdadeiro se o número é palíndromo e retorna falso se ele não é.

Feita a identificação, nosso programa deve receber dois números a e b (sempre $a \leq b$) e verificar quantos palíndromos existem entre a e b , com os extremos incluídos. Veja um possível algoritmo

```
função euler4()
  ct=0
  leia(a,b)
  enquanto (a<=b)
    se palindromo(a)
      ct=ct+1
    fim{se}
    a=a+1
  fim{enquanto}
  escreva(ct)
fim{função}
```

Exemplos

1. Caso 1
Quantos números menores que 2060 tem 8 divisores: 395.

Quantos palíndromos há entre 3359 e 8990: 55.
2. Caso 2
Quantos números menores que 9576 tem 8 divisores: 2017.

Quantos palíndromos há entre 11257 e 87650: 763.
3. Caso 3
Quantos números menores que 55480 tem 8 divisores: 12189.

Quantos palíndromos há entre 59000 e 99000: 400.

👉 Para você fazer

1. Informe quantos números menores do que 4219 têm 8 divisores.
2. Dados os limites $a= 3549$ e $b= 5916$. Ache a quantidade de palíndromos entre eles

Responda aqui:

1	2



Iniciando na programação V

Euler 18

Seja o seguinte problema, como descrito no Euler (problema 18)
By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.
3
7 4
2 4 6
8 5 9 3

That is, 3 + 7 + 4 + 9 = 23.
Find the maximum total from top to bottom of the triangle below:
75
95 64
17 47 82
20 04 82 47 65
19 01 23 75 03 34
88 02 77 73 07 63 67
99 65 04 28 06 16 70 92
41 41 26 56 83 40 80 70 33
41 48 72 33 47 32 37 16 94 29
53 71 44 65 25 43 91 52 97 51 14
70 11 33 28 77 73 17 78 39 68 17 57
91 71 52 38 17 14 91 43 58 50 27 29 48
63 66 04 68 89 53 67 30 73 16 69 87 40 31
04 62 98 27 23 09 70 98 73 93 38 53 60 04 23

NOTE: As there are only 16384 routes, it is possible to solve this problem by trying every route. However, Problem 67, is the same challenge with a triangle containing one-hundred rows; it cannot be solved by brute force, and requires a clever method!

Uma explicação O truque aqui é algo chamado programação dinâmica. Você tem esse triângulo:

3
7 4
2 4 6
8 5 9 3

Quando você se move de baixo para cima, você pode calcular as melhores escolhas na última iteração. Neste caso, você pega a última linha 8 5 9 3 e maximiza a soma na linha anterior.
Na linha anterior, você tem a linha 2 4 6, vamos iterar nela.

- De 2, você pode ir para 8 ou 5, então 8 é melhor (maximize seu resultado de 2) então você calcula a primeira soma 8 + 2 = 10.
- De 4, você pode ir para 5 ou 9, então 9 é melhor (maximize seu resultado de 4) para calcular a segunda soma 9 + 4 = 13.
- De 6, você pode ir para 9 ou 3, então 9 é melhor novamente (maximize seu resultado de 6) para calcular a terceira soma 9 + 6 = 15.

Este é o fim da primeira iteração e você obteve a linha de somas 10 13 15.
O truque final aqui, é que agora, você tem um triângulo menor que é

3
7 4
10 13 15

Agora continue até obter um valor, que é exatamente 23.

O algoritmo

```
função euler18
mat[1][c]={...},{...}...
i = len(mat)
enquanto i > 1
  j = i-1
  aux[1,2,3...,i] = 0
  enquanto j > 0
    aux[j]=max(mat[i;j]+mat[i-1;j], \
               mat[i;j+1]+mat[i-1;j])
  j = j-1
  fim(enquanto)
  x[i-1;1,2,...,i-1] = aux[1,2,...,i-1]
  i = i-1
fim(enquanto)
retorne aux[1]
fim(função)
```

Para você fazer

14
48 3
38 11 33
94 9 7 1
97 12 15 80 58
27 44 59 25 79 86
93 67 70 61 69 3 30
32 68 55 38 91 76 85 2
77 69 28 67 66 72 33 36 10
96 5 59 61 25 60 51 10 4 58
46 37 50 52 54 79 96 60 79 78 28
8 7 40 8 43 21 91 49 38 62 67 81
2 96 59 54 74 83 25 29 42 97 23 73 73
60 82 83 32 56 66 18 12 20 39 29 46 77 66
65 6 97 30 62 85 13 83 6 93 30 15 57 86 94

Ache o valor máximo saindo do elemento 1,1 e chegando até algum elemento na linha 15. Responda aqui:

Euler 21 Como esta lá (devidamente traduzido):

Seja $d(n)$ a soma dos divisores próprios de n (números menores do que n que dividem n sem deixar resto. Se $d(a) = b$ e $d(b) = a$ e $a \neq b$, então a e b são um par amigável e cada um deles a ou b são chamados números amigos. Por exemplo, os divisores próprios de 220 são 1,2,4,5,10,11,20,22,44,55 e 110. Daqui $d(220) = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$. Os divisores próprios de 284 são 1,2,4,71 e 142, então $d(284) = 1 + 2 + 4 + 71 + 142 = 220$ e $d(284) = 220$. Logo 220 e 284 são amigos um do outro.

O algoritmo Primeiro você precisa definir a função que acha a soma dos divisores próprios de um número

```
função sdivs(x)
z = chão(raizquadrada(x))
s = 1
para i de 2 ate z
  se x%i==0
    s=s+i
    s=s+x//i
fim{se}
fimpara}
se z*z == x
  s = s - z
fim{se}
retorne s
fim(função)
```

Agora, A conta é fácil. Dado um número x qualquer, vai-se obter a sua função d , por hipótese $y = d(x)$. Agora faz-se o caminho inverso, buscando $d(y)$ e comparando este resultado com x .

Para você fazer

Dado o número
12285
descubra se ele é amigo de alguém ou não. Se for, escreva aqui o amigo dele e se não for escreva -1.

Euler 28 Como está lá, devidamente traduzido:

Começando no número 1 e movendo-se para a direita no sentido dos ponteiros do relógio, uma matriz 5×5 é formada como segue:

21	22	23	24	25
20	7	8	9	10
19	6	1	2	11
18	5	4	3	12
17	16	15	14	13

Pode-se ver que a soma dos números vermelhos, nas diagonais é 101.

O algoritmo Primeiro, repare que a ordem da matriz precisa ser um número impar. Se não for, a solução não está definida. A solução, como quase sempre, não está em construir a matriz antes e somar depois, afinal isso dará um trabalho enorme, sem contar que pode faltar memória caso a matriz seja muito grande. É muito melhor buscar leis de formação. Vamos dividir as diagonais em 4 semi-diagonais saindo do centro e indo para cada uma das extremidades da matriz. Começando com a diagonal das 1h30, veja-se que a soma é de $1+9+25+49+...$ que vem a ser $1^2+3^2+5^2+7^2+...$ então esta semi-diagonal é $\sum_{n=1,3,5,7,...} = n^2$. A semi-diagonal seguinte, (a das 10h30), tem uma soma parecida: $1+7+21+43+...$, que vem a ser $1^2-0+3^2-2+5^2-4+7^2-6+...$ e esta soma é $\sum_{n=1,3,5,...} = n^2 - (n-1)$. Em frente: a semi-diagonal das 7h30, é $1+5+17+37+...$ que é $1^2-0+3^2-4+5^2-8+7^2-12+...$ ou $\sum_{n=1,3,5,...} = n^2 - 2 \times (n-1)$. Finalmente a diagonal das 4h30 tem fórmula $\sum_{n=1,3,5,...} = n^2 - 3 \times (n-1)$.

Aqui vale excluir o um, já que ele é somado uma única vez, ficando fora do laço. Somando as 4 semi-diagonais, a fórmula final é $4n^2 - 6(n-1)$.

Agora acompanhe:	
diagonal das 1h30:	n^2
diagonal da 10h30:	$n^2 - (n-1)$
diagonal das 7h30:	$n^2 - 2(n-1)$
diagonal das 4h30:	$n^2 - 3(n-1)$

A soma dos 4: $4n^2 - 6(n-1)$
o código

```
função achaespiral (ordem)
s = 1 // porque o 1 é excluída da soma
n = 3 // daí, a ordem começa em 3
enquanto n <= ordem
  s = s + (4*n^2) - 6*(n-1)
  n = n + 2
fim(enquanto)
retorne s
fim(função)
```

Para você fazer

Qual a soma dos números nas diagonais numa espiral de 843×843 formada da mesma maneira ?



- 1 -

Iniciando na programação VI

Euler 80

Como está lá (devidamente traduzido)
É bem sabido que se a raiz quadrada de um número natural não for um inteiro, então ele é irracional. A expansão decimal de tais raízes quadradas é infinita sem qualquer padrão de repetição.
A raiz quadrada de dois é 1.41421356237309504880, e a soma digital dos primeiros cem dígitos decimais é 475
Para os primeiros cem números naturais, encontre o total das somas digitais dos primeiros cem dígitos decimais para todas as raízes quadradas irracionais. A resposta a esta “simples” pergunta é 40886.

Uma solução A dificuldade inicial é que o cálculo usual em computadores costuma assegurar cerca de 18 dígitos significativos, às vezes o dobro em computação estendida. Além disso, é necessário usar outra estratégia/algoritmos que não os usuais.
Uma possibilidade é usar um algoritmo conhecido como “japonês” que tem esse nome por ter origem nesse país, nas areias do velho passado. Disseminado por aqui pelo prof. Frazer Jarvis, ele calcula qualquer raiz quadrada, com qualquer quantidade de dígitos, usando apenas subtrações, somas e multiplicações. Supondo que se quer a raiz de n , ele começa com o par $5 \times n$ e 5 e depois aplica duas regras ao par (p, q) :

- regra A** Se $p \geq q$ o próximo par é $(p - q, q + 10)$
- regra B** se $p < q$, o próximo par é $(100 \times p, 10 \times q - 45)$

O número de aplicações da regra A entre sucessivas aplicações da regra B é o próximo dígito na raiz quadrada de n . Acompanhe como exemplo o cálculo da raiz quadrada de 3:

$(15, 5) \xrightarrow{A^1} (10, 15) \xrightarrow{B} (1000, 105) \xrightarrow{A^1} (895, 115) \xrightarrow{A^2} (780, 125) \xrightarrow{A^3} (655, 135) \xrightarrow{A^4} (520, 145) \xrightarrow{A^5} (375, 155) \xrightarrow{A^6} (220, 165) \xrightarrow{A^7} (55, 175) \xrightarrow{B} (5500, 1705) \xrightarrow{A^1} (3795, 1715) \xrightarrow{A^2} (2080, 1725) \xrightarrow{A^3} (355, 1735) \xrightarrow{B} (35500, 17305) \xrightarrow{A^1} (18195, 17315) \xrightarrow{A^2} (880, 17315) \xrightarrow{B} (88000, 173205) (A^0) \xrightarrow{B} (8800000, 1732005)...$

Antes que você se entusiasme muito com esta regra, vale a constatação de que p e q não param de crescer. Sem nenhum esforço adicional, no cálculo de $\sqrt{2}$, há perda de precisão a partir do 33º dígito. Para impedir esta dificuldade, sugere-se trabalhar com números representados por strings alfanuméricos (agora de tamanhos praticamente ilimitados), tendo-se que reprogramar as funções de comparação, multiplicação e adição para esta classe de números.
Alguns ambientes mais modernos (Python) já estão preparados para lidar com grandes números, e neles o algoritmo/implementação é mais fácil. Neste ambiente, um possível algoritmo seria

```
função somadigs(n)
    soma=0
    enquanto n != 0
        soma = soma + n%10
        n=n//10
    fim{enquanto}
    retorne soma
fim{função}
função raizrap(n)
    a = 5*n
    b = 5
```

```
limite = 10 ** 101 // olhe que número...
se a >= b
    a = a-b
    b = b + 10
senão
    a = a * 100
    b = (b-b%10)*10 + b%10
fim{se}
raiz = ... aqui é python: int(str(b)[:100])
retorne raiz
fim{função}
```

Vale ressaltar que Python automaticamente “vira a chave” para o modo *bignumer* sem que você precise fazer nada. Se, ao contrário for usar Java, use a classe BigInteger. Em C++, existem algumas bibliotecas, mas o mais recomendável é usar strings de caracteres para representar os números grandes.

Para você fazer

Calcule a soma dos 100 dígitos de raízes irracionais contidas entre 1 e 63
Não se esqueça de excluir as raízes racionais (1, 4, 9, 16, ...) Responda aqui:

Euler 25

Como está lá, já traduzido:
A sequência de Fibonacci é definida pela relação de recorrência ¹

$$F_n = F_{n-1} + F_{n-2}, \text{ onde } F_1 = 1, F_2 = 1.$$

Aqui, os 12 primeiros termos da sequência: $F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, F_7 = 13, F_8 = 21, F_9 = 34, F_{10} = 55, F_{11} = 89, F_{12} = 144$.

O décimo segundo termo F_{12} é o primeiro termo a conter 3 dígitos. Qual o índice do primeiro termo a ter 1000 dígitos ? A resposta é 4782.

Solução Como sempre há vários caminhos a usar. O mais óbvio, ainda que mais trabalhoso é construir números grandes usando strings alfanuméricos e neles construir a função de soma e de tamanho. Com ambas, o próximo número pode facilmente ser construído e ter o seu tamanho testado.

Outra possibilidade é usar um ambiente que tenha *bignumber* de forma primitiva (Python, Java...) e aqui, usa-se um truque para descobrir a magnitude (a quantidade de dígitos) de um determinado número. Trata-se do recurso ao logaritmo decimal.

Para calcular a quantidade de dígitos de um número usando logaritmo decimal (base 10), podemos utilizar a seguinte expressão:

$$\lfloor \log_{10}(N) \rfloor + 1$$

Onde:

$\lfloor x \rfloor$ representa a função “piso”, que retorna o maior inteiro menor ou igual a x . $\log_{10}(N)$ é o logaritmo decimal de N , ou seja, o expoente ao qual 10 precisa ser elevado para obter N . O + 1 é adicionado porque o logaritmo indica a quantidade de dígitos à esquerda da vírgula decimal (parte inteira), e queremos considerar o dígito das unidades também.

O logaritmo decimal de um número representa a quantidade de vezes que 10 precisa ser multiplicado por si mesmo para chegar a esse número. Por exemplo, $\log_{10}(100) = 2$, pois $10^2 = 100$. Para números maiores que 10, o logaritmo decimal aumenta em 1 a cada vez que o número aumenta em uma ordem de magnitude (ex: de 100 para 1000, de 1000 para 10000, etc.). A função “piso” garante que obtenhamos um número inteiro, pois o logaritmo nem sempre é um valor inteiro. Ao adicionar 1, consideramos o dígito das unidades, completando o número de dígitos totais. Exemplo:

Considere o número 12345
 $\log_{10}(12345) \approx 4.093$
 $\lfloor \log_{10}(12345) \rfloor = 4$
 $\lfloor \log_{10}(12345) \rfloor + 1 = 5$
Portanto, o número 12345 tem 5 dígitos

Mais alguns exemplos considerando os limites de passagem:

n	$\log_{10}(n)$	$\lfloor \log_{10}(n) \rfloor$
999	2.999565488	2
1000	3	3
1001	3.000434077	3

Essa expressão é válida apenas para números inteiros positivos maiores que 0. Para números entre 0 e 1, o logaritmo decimal é negativo e a expressão precisa ser modificada. Para números decimais, a parte decimal precisa ser considerada separadamente.

O algoritmo

```
*** usando bignumber ***
função euler25
    n = 10**100
    n = 1 // proximo número de fibonacci
    a = 1 // o ante-anterior
    b = 1 // o anterior
    r = 2 // a ordem do anterior
    enquanto log10(n)+1 < 1000
        n = a + b
        a = b
        b = n
        r = r+1
    fim{enquanto}
    escreva (r)
fim{função}
```

Para testar este algoritmo, verifique que o 4782º número de Fibonacci é o primeiro a ter mais de 1000 dígitos, e o 4735º é o primeiro a ter mais do que 990 dígitos.

Para você fazer

Encontre o índice do primeiro número de Fibonacci a ter mais do que 992 dígitos. Responda aqui:

¹Segundo o Gemini, *Uma relação de recorrência é uma técnica matemática para definir sequências, conjuntos, operações ou até mesmo algoritmos, especificando cada termo em função de um ou mais termos anteriores. Ela funciona como uma regra que determina como calcular cada elemento da sequência a partir de seus predecessores.*

Elementar, meu caro Watson. Holmes, S.

Construindo seqüências Para este exercício, são apresentadas diversas seqüências de 7 números. Naturalmente, elas são objeto de um programa de computador. O programa é apresentado, mas faltam nele certos parâmetros (x , y e z), cujos valores serão inteiros e sempre estarão entre 2 e 7, inclusive. Pelo aspecto da seqüência e do programa mostrado, você deve inferir tais valores. Depois disso, a seqüência já impressa deve ser verificada (preferencialmente pelo computador). Estando os 7 primeiros valores corretos, devem estar os seguintes. Para cada seqüência, você deve responder qual será o décimo valor.

Seqüência 1
1: para I de 1 até 10
2: escreva $(x + (I^2))$
3: fim{para}
Para a seqüência 5 8 13 20 29 40 53 68 85 104, pode-se ver que o 10º valor da seqüência é 104, já que $x = 4$.

Seqüência 2
1: para I de 1 até 10
2: escreva $(x + (I^3))$
3: fim{para}
Para a seqüência 4 11 30 67 128 219 346 515 732 1003, pode-se ver que o 10º valor da seqüência é 1003, já que $x = 3$.

Seqüência 3
1: para I de 1 até 10
2: escreva $((x + I)^2)$
3: fim{para}
Para a seqüência 16 25 36 49 64 81 100 121 144 169, pode-se ver que o 10º valor da seqüência é 169, já que $x = 3$.

Seqüência 4
1: para I de 1 até 10
2: escreva $((x + I)^3)$
3: fim{para}
Para a seqüência 512 729 1000 1331 1728 2197 2744 3375 4096 4913, pode-se ver que o 10º valor da seqüência é 4913, já que $x = 7$.

Seqüência 5
1: para I de 1 até 10
2: escreva $((x \times I)^2)$
3: fim{para}
Para a seqüência 36 144 324 576 900 1296 1764 2304 2916 3600, pode-se ver que o 10º valor da seqüência é 3600, já que $x = 6$.

Seqüência 6
1: para I de 1 até 10
2: escreva $((x \times I)^3)$

🔧 Para você fazer

seq	v_1	v_2	v_3	v_4	v_5	v_6	v_7	10º ?
1	5	8	13	20	29	40	53	
2	5	12	31	68	129	220	347	
3	16	25	36	49	64	81	100	
4	64	125	216	343	512	729	1000	
5	25	100	225	400	625	900	1225	
6	8	64	216	512	1000	1728	2744	
7	2	4	8	16	32	64	128	
8	16	28	48	76	112	156	208	
9	30	65	160	345	650	1105	1740	
10	196	256	324	400	484	576	676	
11	108	256	500	864	1372	2048	2916	
12	69	86	105	126	149	174	201	
13	347	516	733	1004	1335	1732	2201	
14	6	8	12	20	36	68	132	
15	8	16	32	64	128	256	512	
16	39	51	71	99	135	179	231	
17	29	57	133	281	525	889	1397	
18	348	453	572	705	852	1013	1188	
19	652	1033	1540	2191	3004	3997	5188	
20	30	42	66	114	210	402	786	

3: fim{para}
Para a seqüência 8 64 216 512 1000 1728 2744 4096 5832 8000, pode-se ver que o 10º valor da seqüência é 8000, já que $x = 2$.

Seqüência 7
1: para I de 1 até 10
2: escreva (x^I)
3: fim{para}
Para a seqüência 2 4 8 16 32 64 128 256 512 1024, pode-se ver que o 10º valor da seqüência é 1024, já que $x = 2$.

Seqüência 8
1: para I de 1 até 10
2: escreva $((I^2) + x) \times y$
3: fim{para}
Para a seqüência 12 18 28 42 60 82 108 138 172 210, pode-se ver que o 10º valor da seqüência é 210, já que $x = 5$ e $y = 2$.

Seqüência 9
1: para I de 1 até 10
2: escreva $((I^3) + x) \times y$
3: fim{para}
Para a seqüência 15 50 145 330 635 1090 1725 2570 3655 5010, pode-se ver que o 10º valor da seqüência é 5010, já que $x = 2$ e $y = 5$.

Seqüência 10
1: para I de 1 até 10
2: escreva $((x + I)^2) \times y$
3: fim{para}
Para a seqüência 180 245 320 405 500 605 720 845 980 1125, pode-se ver que o 10º valor da seqüência é 1125, já que $x = 5$ e $y = 5$.

Seqüência 11
1: para I de 1 até 10
2: escreva $((x + I)^3) \times y$
3: fim{para}
Para a seqüência 2401 3584 5103 7000 9317 12096 15379 19208 23625 28672, pode-se ver que o 10º valor da seqüência é 28672, já que $x = 6$ e $y = 7$.

Seqüência 12
1: para I de 1 até 10
2: escreva $((x + I)^2) + y$
3: fim{para}
Para a seqüência 22 31 42 55 70 87 106 127 150 175, pode-se ver que o 10º valor da seqüência é 175, já que $x = 3$ e $y = 6$.

Seqüência 13
1: para I de 1 até 10
2: escreva $((x + I)^3) + y$
3: fim{para}
Para a seqüência 131 222 349 518 735 1006 1337 1734 2203 2750, pode-se ver que o 10º valor da seqüência é 2750, já que $x = 4$ e $y = 6$.

quência é 2750, já que $x = 4$ e $y = 6$.

Seqüência 14
1: para I de 1 até 10
2: escreva $((x^I) + y)$
3: fim{para}
Para a seqüência 8 10 14 22 38 70 134 262 518 1030, pode-se ver que o 10º valor da seqüência é 1030, já que $x = 2$ e $y = 6$.

Seqüência 15
1: para I de 1 até 10
2: escreva (x^{I+y})
3: fim{para}
Para a seqüência 32 64 128 256 512 1024 2048 4096 8192 16384, pode-se ver que o 10º valor da seqüência é 16384, já que $x = 2$ e $y = 4$.

Seqüência 16
1: para I de 1 até 10
2: escreva $((x + (I^2)) \times y) + z$
3: fim{para}
Para a seqüência 25 37 57 85 121 165 217 277 345 421, pode-se ver que o 10º valor da seqüência é 421, já que $x = 4$, $y = 4$ e $z = 5$.

Seqüência 17
1: para I de 1 até 10
2: escreva $((x + (I^3)) \times y) + z$
3: fim{para}
Para a seqüência 31 59 135 283 527 891 1399 2075 2943 4027, pode-se ver que o 10º valor da seqüência é 4027, já que $x = 6$, $y = 4$ e $z = 3$.

Seqüência 18
1: para I de 1 até 10
2: escreva $((x + I)^2) \times y + z$
3: fim{para}
Para a seqüência 255 346 451 570 703 850 1011 1186 1375 1578, pode-se ver que o 10º valor da seqüência é 1578, já que $x = 5$, $y = 7$ e $z = 3$.

Seqüência 19
1: para I de 1 até 10
2: escreva $((x + I)^3) \times y + z$
3: fim{para}
Para a seqüência 2061 3075 4377 6003 7989 10371 13185 16467 20253 24579, pode-se ver que o 10º valor da seqüência é 24579, já que $x = 6$, $y = 6$ e $z = 3$.

Seqüência 20
1: para I de 1 até 10
2: escreva $((x^I) + y) \times z$
3: fim{para}
Para a seqüência 10 14 22 38 70 134 262 518 1030 2054, pode-se ver que o 10º valor da seqüência é 2054, já que $x = 2$, $y = 3$ e $z = 2$.

Responda aqui

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20



- 1 - /

Desenhando funções

Todo livro de álgebra quando passa a ensinar o comportamento e características de funções, insiste em que o estudante/leitor se acostume a desenhar o gráfico das funções em que está trabalhando. Este desenho sempre diz muito sobre a função, suas raízes, sua paridade, suas assíntotas, seus pontos característicos (máximos, mínimos, de inflexão, ...).

Dois problemas surgiam aqui:

- é difícil, dá trabalho e até exige algum talento com o desenho. Além, é necessário papel quadriculado, canetas coloridas, essas paradas...
- eventualmente, algum engano ou imprecisão no desenho poderia levar a conclusões falsas. Acontecendo isto, além de não ajudar, o desenho atrapalhava o estudo.

A novidade agora é que graças aos pacotes gráficos do python, do Maple, do Octave (só para citar alguns), ficou muito fácil produzir tais desenhos. Eles sempre estão certos e são bonitos. E, a grande sacada, é que é sempre mais ou menos o mesmo programa que faz o desenho. É ele que vai ser estudado aqui. Tangencialmente, vai se estudar um pouco do pacote simbólico (sympy) que engorda os ambientes Python e Octave. Ficará de fora a componente simbólica do Maple (que também é muito boa) e do Mathematica, mas que são compradas e pagas. Nada como ficar no ambiente *free*.

Funções

Uma função é um objeto matemático que recebe números como entradas e fornece números como saída. Usa-se a notação $f: A \rightarrow B$ para denotar uma função do conjunto A para o B . Vamos estudar aqui, funções que mapeiam reais para reais ou $f: \mathbb{R} \rightarrow \mathbb{R}$. Ainda sobre funções define-se para uma dada função:

domínio conjunto de valores de entrada permitidos.

imagem conjunto de todos os valores possíveis de saída.

contradomínio descreve o tipo de saída.

Para entender a sutil diferença entre imagem e contradomínio, considere, por exemplo, $f(x) = x^2$. O domínio e o contradomínio são os reais. Já a imagem é o conjunto dos reais positivos. Para estudar o mapeamento da função, define-se

injetiva - mapeia entradas diferentes para saídas diferentes.

sobrejetiva - cobre todo o conjunto de saída.

bijetiva - injetiva + sobrejetiva.

Dicionário de funções

Reta Descreve uma relação na qual a mudança na saída é proporcional à mudança na entrada. Tem equação $f(x) = mx + b$. Aqui m descreve a inclinação e b a interceptação no eixo y . O domínio é $x \in \mathbb{R}$. Uma única linha passa através de dois pontos (x_1, y_1) e (x_2, y_2) se $x_1 \neq x_2$. Sua função inversa também é uma reta.

Função quadrática Ou parábola. A fórmula é $f(x) = x^2$. O nome quadrático, vem do latim, *quadratus* pois a expressão para a área de um quadrado de lado x é x^2 . O domínio é $\mathbb{R}$ e a imagem é o conjunto dos positivos ou $f(x) \in [0, \infty)$. Sua função inversa é a raiz quadrada ($\sqrt{\cdot}$).

Valor absoluto Ou módulo. Responde com o tamanho de um número sem importar se ele é positivo ou negativo. Outra maneira de pensar aqui é dizer que ela multiplica negativos por -1 para cancelar seu sinal.

$$f(x) = |x| = \begin{cases} x & \text{se } x \geq 0 \\ -x & \text{se } x < 0 \end{cases}$$

Função polinomial $f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$. O valor n é o grau do polinômio e as variáveis a_i são seus coeficientes. Um polinômio de grau n tem $n+1$ coeficientes. O domínio é $\mathbb{R}$ e a imagem depende dos coeficientes. A soma de dois polinômios é um polinômio.

Seno Uma unidade fundamental de vibração. Seu gráfico atravessa várias vezes o eixo x . Seu domínio é $\mathbb{R}$ e sua imagem é $[-1, 1]$. A função é periódica com período 2π .

Cosseno É a mesma função do seno, mas mudando um comprimento π para a esquerda: $\cos(x) = \sin(x + \pi)$. Assim tudo o que se sabe do seno vale para o cosseno.

Tangente É a razão entre o seno e o cosseno. $f(x) = \tan(x) \equiv \frac{\sin(x)}{\cos(x)}$. A função é periódica em π .

Exponencial Esta função é denotada como $f(x) = e^x$. O valor $e = 2.7182818...$ A origem deste valor está associado a um limite de rendimento de uma conta investimento que paga 100% de juros considerados cada vez períodos menores (tendendo a zero). Esta função tem uma característica única: sua derivada é igual à função original. Sua função inversa é o logaritmo natural.

Gráficos

Um gráfico de função é uma associação entre valores de entrada (livremente escolhidos pelo leitor/estudioso) e os valores de saída providos pela função em estudo. Para automatizar este processo, há que haver 2 competências:

- Escolha de valores de entrada que sejam característicos do comportamento da função
- A capacidade de transformar a função em comandos de programação

Feito isto, basta usar a receita a seguir e mandar ver.

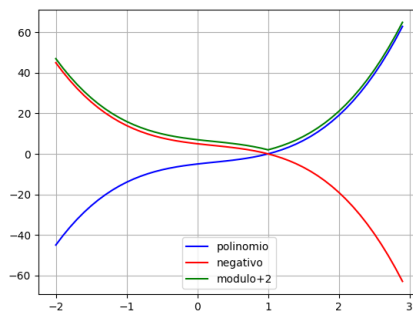
```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return...
t=np.arange(in,fi,incremento) OU
t=np.linspace(in,fi,qtde=pontos)
plt.plot(t,f1(t),'b-',label='nome')
plt.grid()
plt.legend()
plt.show()
```

`arange` e `linspace` geram vetores igualmente espaçados. A diferença é que `arange` cria pontos entre `in` e `fi` com o incremento dado. A quantidade de pontos fica por conta dele. Já em `linspace` pede-se a quantidade de pontos citada entre `in` e `fi`. Agora, o incremento fica por conta dele.

Exemplo Veja como gerar um gráfico de um polinômio

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return 3*x**3-2*x**2+4*x-5
t=np.arange(-2,3,0.1)
plt.plot(t,f(t),'b-',label='polinomio')
plt.plot(t,-f(t),'r-',label='negativo')
plt.plot(t,2+abs(f(t)),'g-',label='modulo+2')
plt.grid()
plt.legend()
plt.show()
```

que deu origem a



Outros ambientes

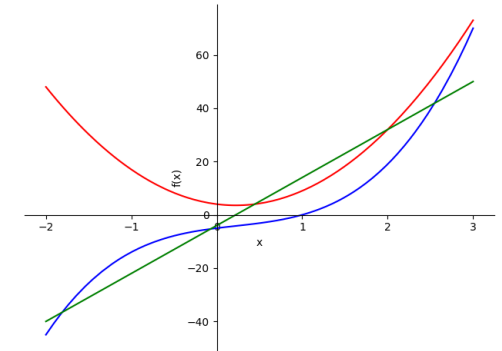
Maple: 0 mesmo gráfico acima pode ser gerado por
`plot (3*x^3-2*x^2+4*x-5,x=-2..3);`
 Octave: `x=-2:0.1:3;`
`y=3*x.^3-2*x.^2+4*x-5;`
`plot(x,y);`

Processamento simbólico

Você pode levar a produção de gráficos a um nível mais alto, juntando algum processamento simbólico antes de desenhar o gráfico. Naturalmente, quanto maior é o benefício obtido, mais custosa é a realização da demanda. Aqui vamos nos concentrar em SYMPY, que é o pacote simbólico do Python e é também usado no Octave. Dá para fazer isto em Maple, mas este vai ficar de fora (...afinal, ele é pago).

Ao invés de explicar o sympy (o seu manual básico tem 3271 páginas em fev/24) vamos comandar um gráfico composto por uma função polinomial, sua derivada primeira e sua derivada segunda. O detalhe é que o sympy é que vai calcular as derivadas. Note que o plot aqui é o `sp.plot()`

```
import sympy as sp
sp.var('x')
f=3*x**3-2*x**2+4*x-5
g=sp.diff(f)
h=sp.diff(g,x,2)
p=sp.plot(f,g,h,(x,-2,3),show=sp.false)
p[0].line_color='b'
p[1].line_color='r'
p[2].line_color='g'
p.show()
```



Jeito alternativo com mais controle: o plot é o `plt.plot()`.

```
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
sp.var('x')
f=3*x**3-2*x**2+4*x-5
g=sp.diff(f,x)
h=sp.diff(g,x)
x1=np.linspace(-3,2,100)
y1=[0.0]*100
y2=[0.0]*100
y3=[0.0]*100
for i in range(100):
    y1[i]=f.subs(x,x1[i])
    y2[i]=g.subs(x,x1[i])
    y3[i]=h.subs(x,x1[i])
plt.plot(x1,y1,'b-',label='função original')
plt.plot(x1,y2,'r-',label='derivada primeira')
plt.plot(x1,y3,'g-',label='derivada segunda')
plt.grid()
plt.legend()
plt.title("Comportamento da função")
plt.show()
```

Deixo de incluir o desenho aqui por falta de espaço. Mas peça o gráfico e veja como ele ficou.

Para você fazer

Gere o gráfico da função

$2x^3+6x^2-3x+3$

Entregue o desenho impresso, ou mostre para o professor, junto com o código Python.



Leitura dados pré-existentis

Na sua vida profissional, será comum você ser chamado a processar dados gerados em outro local, por outras pessoas e em outros ambientes operacionais. Com a constante digitalização dos conteúdos e das informações, tal demanda já existe hoje e tende a se tornar mandatária com o passar dos anos.

Tipo de arquivo A primeira questão que se precisa definir é com relação ao tipo do arquivo a manusear. Fundamentalmente existem 2 tipos, que são conhecidos como BINÁRIOS e de TEXTO. A distinção é simples, os arquivos binários tem uma lógica de construção e funcionamento que dependem do ambiente gerador. Não seguem algum padrão, nem têm operação automática. Uma maneira fácil de identificar este tipo de arquivo é dar um comando de MOSTRAR o conteúdo. Neste tipo de arquivo o conteúdo será composto de caracteres ininteligíveis, não sendo possível intuir nada. (Em Windows/DOS o comando de mostrar é TYPE. Em Linux, o comando é CAT). Outro jeito fácil é abrir o arquivo para edição usando um editor pleno (o notepad, o N++, o scintilla, ou qualquer programa similar). Tendo aberto um arquivo BINÁRIO seu conteúdo será impossível de entender.

Por não serem padrão, os arquivos binários não serão trabalhados aqui. Quando tiver que usá-los, você vai precisar pedir ajuda ao suporte técnico, ou então à pessoa ou organização que gerou o arquivo.

Os arquivos tipo TEXTO, como o nome diz, contém apenas caracteres visíveis/legíveis. Neste tipo de arquivo, os dados estão organizados em linhas, e cada linha é encerrada por dois caracteres padrão conhecidos como CR (carriage return) e LF (line feed). Estes 2 caracteres não são visíveis, mas é possível intuir sua presença pelo comportamento dos dados enquanto são mostrados.

Uma maneira segura de "olhar" dentro de qualquer arquivo seja BINÁRIO seja de TEXTO é através de um editor binário, como por exemplo o HEXEDIT (freeware, procure na Internet).

Modo de acessar Há 2 maneiras básicas de acessar um arquivo de entrada. A primeira usa uma propriedade dos sistemas operacionais, chamada **redirecionamento**. Suponha que seu programa de nome ALFA33, lê dados do teclado (através de scanf). Se, ao chamar o programa, você digitar ALFA33 < ARQENTR tudo vai funcionar como se o conteúdo do arquivo ARQENTR fosse digitado pelo teclado como resposta aos comandos scanf do programa.

Ao agir assim, o programa fonte não sofre nenhuma alteração e a vantagem é que o mesmo programa, sem nenhuma modificação, aceita dados tanto do teclado quanto de um arquivo.

Embora não vá ser tratado aqui, o redirecionamento também pode ser feito para a saída, e daí, os comandos printf ao invés de imprimir no vídeo, vão gerar um arquivo em disco. Para redirecionar a saída, fazer ALFA33 > ARQSAIDA.

Os dois redirecionamentos podem ser usados juntos: ALFA33 < ARQENTR > ARQSAIDA, e obviamente o redirecionamento só pode ser usado para arquivo TEXTO.

A outra maneira de acessar arquivos em disco, é declarar isto explicitamente, como se verá a seguir:

Open/Close: Dentro de qualquer ambiente computacional, um arquivo é conhecido pelo seu nome (e eventualmente, este nome é composto, incluindo o caminho até o mesmo). Por exemplo, um arquivo pode se chamar c:/dados/versao2/cad-pessoas-set-12.txt. Neste exemplo o arquivo é cad-pessoas-set-12.txt que reside no disco que está em C: e dentro dele, no diretório dados e sub-diretório versao2.

Já dentro dos programas C/C++, existe um nome interno, que é sempre o mesmo. (Esta estratégia é inteligente, já que não teria sentido ter que reescrever o programa cada vez que o nome do arquivo de dados fosse modificado).

A associação entre nome externo (no disco) e interno (no programa) é feita por uma operação

chamada OPEN e a dissociação destes dois elementos é feita pela operação CLOSE ou pelo encerramento do programa, o que ocorrer primeiro.

Neste exercício, você vai receber diversos arquivos, todos do tipo TEXTO, cada um deles com um determinado formato e para cada um, você deve calcular o que é pedido. Antes de começar, estude o arquivo recebido e tente entender qual é a lógica de sua construção. Use o HEXEDIT ou similar para fazer isso. Veja também o arquivo (se ele for do tipo TEXTO) com um editor comum, como o SCINTILLA ou mesmo o NOTEPAD. Descubra o tamanho dos registros e a quantidade de registros que existem no arquivo. Considere neste caso o conceito de registro como quase sinônimo de linha.

Leitura As leituras de dados (uma para cada registro) são feitas usando-se o comando fscanf do C, ou o operador CIN do C++. O comando fscanf recebe 3 parâmetros: o nome interno do arquivo é uma expressão que descreve o formato dos dados e as variáveis que serão alimentadas. Já o CIN é bem mais fácil, basta chamá-lo junto com a variável que vai ser lida.

Um exemplo completo Suponha um arquivo contendo em cada linha, um valor real referente a dinheiro e um número inteiro referente a um certo número de dias. Os códigos CR+LF não são visíveis, mas estão lá:

```
2500.00 56 CR+LF
1270.77 66 CR+LF
...
```

No C, devem ser executados pelo menos estes 4 comandos:

- a) FILE *fd;
- b) fd=fopen("c:/lixo/blabla", "r");
- c) fscanf(fd, "%f %i", &a, &b);
- d) fclose(fd);

Observações:

1. O nome externo é c:/lixo/blabla. Note que as barras são viradas à direita, por exigência do C.
2. É preciso criar uma variável do tipo FILE para conter o descritor do arquivo.
3. O dado vai ser aberto para leitura (método r, de READ).
4. O nome interno é fd.
5. O segundo parâmetro do comando fscanf se refere a dois valores. O primeiro é um real (%f=floating) e um inteiro (%i=inteiro), separados por um único espaço em branco.
6. O terceiro parâmetro identifica as variáveis a serem lidas (com o & antes do nome, não esqueça).
7. fscanf deve ser chamado tantas vezes quantos registros (linhas) há no arquivo.
8. Finalmente, o comando close deve ser emitido uma única vez ao final do processamento.

Acompanhe agora o programa C completo

```
#include<stdio.h>
main() {
FILE *fa;
float a;
int b;
if ((fa=fopen("c:/ap12/xxxf01e2", "r"))==NULL){
printf("arquivo nao pode ser aberto\n");
exit(1);
}
while(!feof(fa)){
fscanf(fa, "%f %i", &a, &b);
printf("%8.2f %i\n", a, b);
}
}
```

Para testar sua lógica: Se quiser testar a lógica de seus programas, processe os arquivos de nomes EXEF00E1, EXEF00E2 e EXEF00E3. Para eles os 10 valores calculados serão:

```
3017.39 5498 386
258648.89 249233.21 114739.33
17.7 184 2 128
```

Para você fazer

Exercício 1: Você deve pegar o arquivo de nome

XXXXF01E1

e estudá-lo. Ele é formado por 1000 números inteiros, cada um representando o rendimento mensal de um determinado empregado de uma certa empresa em reais (sem o uso de centavos, por isso o tipo inteiro). Depois de ler os dados e criar um vetor, deve:

Informe o valor médio do rendimento	1.
O maior rendimento	2.
Quantas pessoas ganham mais do que 20% do rendimento médio	3.

Exercício 2: Agora, o arquivo a ler é

XXXXF01E2

nele estão 500 duplas de dados que se referem a dívidas que pessoas/empresas têm com você. O primeiro valor é um número real significando o valor da dívida em moeda nacional. O segundo valor é um inteiro, indicando o número de dias de atraso. Leia os dados e construa uma matriz de 500 x 2 valores. Depois, responda:

Valor total da dívida	4.
Valor total da dívida se forem desconsideradas as dívidas inferiores a R\$ 200,00	5.
Valor total da dívida se forem desconsideradas as dívidas com prazo de atraso maior do que 89 dias	6.

Exercício 3: Neste exercício você deve ler o arquivo

XXXXF01E3

que contém uma lista de 365 conjunto de valores. Cada um deles descreve as condições meteorológicas de Curitiba, no ano de 2021. Para cada dia, os dados informam:

- Temperatura máxima no dia, em graus centígrados com 1 decimal.
- Temperatura mínima no dia, idem acima.
- Umidade do ar ao meio dia, em percentagem com 1 decimal.
- Precipitação pluvial, em mm como um número inteiro.
- Horas de insolação, em horas inteiras.

Processe os dados e informe:

Temperatura média no ano: (max+min)/2 a cada dia...	7.
Quantos dias tiveram mais de 3h de insolação	8.
Qual o semestre mais chuvoso (1sem=182; 2sem=183)	9.
Quantos dias tiveram umidade abaixo de 30%	10.



Leitura dados pré-existentis

Na sua vida profissional, será comum você ser chamado a processar dados gerados em outro local, por outras pessoas e em outros ambientes operacionais. Com a constante digitalização dos conteúdos e das informações, tal demanda já existe hoje e tende a se tornar mandatária com o passar dos anos.

Tipo de arquivo A primeira questão que se precisa definir é com relação ao tipo do arquivo a manusear. Fundamentalmente existem 2 tipos, que são conhecidos como BINÁRIOS e de TEXTO. A distinção é simples, os arquivos binários tem uma lógica de construção e funcionamento que dependem do ambiente gerador. Não seguem algum padrão, nem têm operação automática. Uma maneira fácil de identificar este tipo de arquivo é dar um comando de MOSTRAR o conteúdo. Neste tipo de arquivo o conteúdo será composto de caracteres ininteligíveis, não sendo possível intuir nada. (Em Windows/DOS o comando de mostrar é TYPE. Em Linux, o comando é CAT). Outro jeito fácil é abrir o arquivo para edição usando um editor pleno (o notepad, o N++, o scintilla, ou qualquer programa similar). Tendo aberto um arquivo BINÁRIO seu conteúdo será impossível de entender.

Por não serem padrão, os arquivos binários não serão trabalhados aqui. Quando tiver que usá-los, você vai precisar pedir ajuda ao suporte técnico, ou então à pessoa ou organização que gerou o arquivo.

Os arquivos tipo TEXTO, como o nome diz, contém apenas caracteres visíveis/legíveis. Neste tipo de arquivo, os dados estão organizados em linhas, e cada linha é encerrada por dois caracteres padrão conhecidos como CR (carriage return) e LF (line feed). Estes 2 caracteres não são visíveis, mas é possível intuir sua presença pelo comportamento dos dados enquanto são mostrados.

Uma maneira segura de "olhar" dentro de qualquer arquivo seja BINÁRIO seja de TEXTO é através de um editor binário, como por exemplo o HEXEDIT (freeware, procure na Internet).

Modo de acessar Há 2 maneiras básicas de acessar um arquivo de entrada. A primeira usa uma propriedade dos sistemas operacionais, chamada **redirecionamento**. Suponha que seu programa de nome ALFA33, lê dados do teclado (através de scanf). Se, ao chamar o programa, você digitar ALFA33 < ARQENTR tudo vai funcionar como se o conteúdo do arquivo ARQENTR fosse digitado pelo teclado como resposta aos comandos scanf do programa.

Ao agir assim, o programa fonte não sofre nenhuma alteração e a vantagem é que o mesmo programa, sem nenhuma modificação, aceita dados tanto do teclado quanto de um arquivo.

Embora não vá ser tratado aqui, o redirecionamento também pode ser feito para a saída, e daí, os comandos printf ao invés de imprimir no vídeo, vão gerar um arquivo em disco. Para redirecionar a saída, fazer ALFA33 > ARQSAIDA.

Os dois redirecionamentos podem ser usados juntos: ALFA33 < ARQENTR > ARQSAIDA, e obviamente o redirecionamento só pode ser usado para arquivo TEXTO.

A outra maneira de acessar arquivos em disco, é declarar isto explicitamente, como se verá a seguir:

Open/Close: Dentro de qualquer ambiente computacional, um arquivo é conhecido pelo seu nome (e eventualmente, este nome é composto, incluindo o caminho até o mesmo). Por exemplo, um arquivo pode se chamar c:/dados/versao2/cad-pessoas-set-12.txt. Neste exemplo o arquivo é cad-pessoas-set-12.txt que reside no disco que está em C: e dentro dele, no diretório dados e sub-diretório versao2.

Já dentro dos programas C++, existe um nome interno, que é sempre o mesmo. (Esta estratégia é inteligente, já que não teria sentido ter que reescrever o programa cada vez que o nome do arquivo de dados fosse modificado).

A associação entre nome externo (no disco) e interno (no programa) é feita por uma operação

chamada OPEN e a dissociação destes dois elementos é feita pela operação CLOSE ou pelo encerramento do programa, o que ocorrer primeiro.

Neste exercício, você vai receber diversos arquivos, todos do tipo TEXTO, cada um deles com um determinado formato e para cada um, você deve calcular o que é pedido. Antes de começar, estude o arquivo recebido e tente entender qual é a lógica de sua construção. Use o HEXEDIT ou similar para fazer isso. Veja também o arquivo (se ele for do tipo TEXTO) com um editor comum, como o SCINTILLA ou mesmo o NOTEPAD. Descubra o tamanho dos registros e a quantidade de registros que existem no arquivo. Considere neste caso o conceito de registro como quase sinônimo de linha.

Leitura As leituras de dados (uma para cada registro) são feitas usando-se o comando getline do C++. Acompanhe o exemplo para ver. **Um exemplo completo** Suponha um arquivo contendo em cada linha, um valor real referente a dinheiro e um número inteiro referente a um certo número de dias. Os códigos CR+LF não são visíveis, mas estão lá:

2500.00 56 CR+LF
1270.77 66 CR+LF
...

No C, devem ser executados pelo menos estes comandos:

```
#include<iostream>
#include<fstream>
#include<sstream>
using namespace std;
int main(){
    float a,b,c;
    int d,e;
    fstream arquivo;
    string linha;
    arquivo.open("c:/apl2/XXXF01E3", ios::in);
    while (1==1){
        getline(arquivo, linha);
        istringstream iss(linha);
        if (iss>>a>>b>>c>>d>>e){
            cout<<a<<" "<<b<<" "<<c<<
            " "<<d<<" "<<e<<b<<endl;
        }
        if (arquivo.eof()){break;}
    }
    arquivo.close();
    return 0;
}
```

Observações:

1. O nome externo é c:/apl2/XXXF01E3. Note que as barras são viradas à direita, por exigência do C++.
2. É preciso criar uma variável do tipo fstream para conter o descritor do arquivo.
3. O dado vai ser aberto para leitura (método ios::in, de input).
4. O nome interno é arquivo.
5. getline deve ser chamado tantas vezes quantos registros (linhas) há no arquivo.
6. Finalmente, o comando close deve ser emitido uma única vez ao final do processamento.

Para testar sua lógica: Se quiser testar a lógica de seus programas, processe os arquivos de nomes EXEF00E1, EXEF00E2 e EXEF00E3. Para eles os 10 valores calculados serão:

3017.39 5498 386
258648.89 249233.21 114739.33
17.7 184 2 128

Para você fazer

Exercício 1: Você deve pegar o arquivo de nome

XXXF01E1

e estudá-lo. Ele é formado por 1000 números inteiros, cada um representando o rendimento mensal de um determinado empregado de uma certa empresa em reais (sem o uso de centavos, por isso o tipo inteiro). Depois de ler os dados e criar um

vetor, deve:

Informe o valor médio do rendimento	1.
O maior rendimento	2.
Quantas pessoas ganharam mais do que 20% do rendimento médio	3.

Exercício 2: Agora, o arquivo a ler é

XXXF01E2

nele estão 500 duplas de dados que se referem a dívidas que pessoas/empresas têm com você. O primeiro valor é um número real significando o valor da dívida em moeda nacional. O segundo valor é um inteiro, indicando o número de dias de atraso. Leia os dados e construa uma matriz de 500 x 2 valores. Depois, responda:

Valor total da dívida	4.
Valor total da dívida se forem desconsideradas as dívidas inferiores a R\$ 200,00	5.
Valor total da dívida se forem desconsideradas as dívidas com prazo de atraso maior do que 89 dias	6.

Exercício 3: Neste exercício você deve ler o arquivo

XXXF01E3

que contém uma lista de 365 conjunto de valores. Cada um deles descreve as condições meteorológicas de Curitiba, no ano de 2021. Para cada dia, os dados informam:

- Temperatura máxima no dia, em graus centígrados com 1 decimal.
- Temperatura mínima no dia, idem acima.
- Umidade do ar ao meio dia, em percentagem com 1 decimal.
- Precipitação pluvial, em mm como um número inteiro.
- Horas de insolação, em horas inteiras.

Processe os dados e informe:

Temperatura média no ano: (max+min)/2 a cada dia...	7.
Quantos dias tiveram mais de 3h de insolação	8.
Qual o semestre mais chuvoso (1sem=182; 2sem=183)	9.
Quantos dias tiveram umidade abaixo de 30%	10.

Autovalores e autovetores

O conjunto de autovetores de uma matriz (quadrada) é um conjunto especial de vetores de entrada para os quais a ação da matriz é descrita como um escalamento simples. A decomposição de uma matriz em seus autovalores e autovetores fornece informações valiosas sobre as propriedades da matriz. Certos cálculos (como calcular a potência de uma matriz) tornam-se mais fáceis quando se usa a composição própria (auto) da matriz. Para tornar este exemplo mais concreto suponhamos a matriz $M = \begin{bmatrix} 9 & -2 \\ -2 & 6 \end{bmatrix}$

da qual precisamos saber M^4 . Queremos calcular $M^4 = \begin{bmatrix} 9 & -2 \\ -2 & 6 \end{bmatrix} \begin{bmatrix} 9 & -2 \\ -2 & 6 \end{bmatrix} \begin{bmatrix} 9 & -2 \\ -2 & 6 \end{bmatrix} \begin{bmatrix} 9 & -2 \\ -2 & 6 \end{bmatrix}$.

Isso é muita multiplicação. Agora imagine M^{77} . Cada matriz corresponde a uma operação linear. Portanto, pode-se perguntar *O que a matriz M faz?* Depois de obter essa resposta basta fazer isso 4 vezes (ou 77 vezes). Para entender a matriz M você deve olhar seus autovalores e autovetores (*eigen* é a palavra alemã para auto). A autocomposição de M é um produto de 3 matrizes: $A = \begin{bmatrix} 9 & -2 \\ -2 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 10 \end{bmatrix} \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & -\frac{1}{5} \end{bmatrix}$

A seguir, uma prova de que isso é verdade

```
import sympy as sp
q=sp.Matrix([[1,2],[2,-1]])
auto=sp.Matrix([[5,0],[0,10]])
qinv=sp.Matrix([[0.2,0.4],[0.4,-0.2]])
q*auto*qinv
Matrix([
[ 9.0, -2.0],
[-2.0,  6.0]])
```

Em resumo $M = Q\Lambda Q^{-1}$ A matriz do meio Λ tem entradas apenas na diagonal. Ela está impressada entre a matriz Q à sua esquerda e a sua inversa Q^{-1} à direita. A autodecomposição de M permite calcular M^4 de maneira facilitada:

$$\begin{aligned} M^4 &= M M M M \\ &= \underbrace{Q \Lambda Q^{-1}}_1 \underbrace{Q \Lambda Q^{-1}}_1 \underbrace{Q \Lambda Q^{-1}}_1 \underbrace{Q \Lambda Q^{-1}}_1 \\ &= Q \Lambda^4 Q^{-1} \\ &= Q \Lambda \Lambda \Lambda \Lambda Q^{-1} \\ &= Q \Lambda^4 Q^{-1}. \end{aligned}$$

Todos os Q^{-1} internos se cancelam com os Q adjacentes. Como Λ é diagonal é fácil calcular sua quarta potência: $\Lambda^4 = \begin{bmatrix} 5 & 0 \\ 0 & 10 \end{bmatrix}^4 = \begin{bmatrix} 5^4 & 0 \\ 0 & 10^4 \end{bmatrix} = \begin{bmatrix} 625 & 0 \\ 0 & 10000 \end{bmatrix}$. Assim, podemos expressar nosso cálculo de M^4 como

$$M^4 = \begin{bmatrix} 9 & -2 \\ -2 & 6 \end{bmatrix}^4 = \underbrace{\begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}}_Q \begin{bmatrix} 625 & 0 \\ 0 & 10000 \end{bmatrix} \underbrace{\begin{bmatrix} \frac{1}{5} & 0 \\ 0 & -\frac{1}{5} \end{bmatrix}}_{Q^{-1}}.$$

Ainda precisamos multiplicar estas três matrizes, mas reduzimos o trabalho de quatro multiplicações de matrizes para duas. A resposta é

$$M^4 = Q \Lambda^4 Q^{-1} = \begin{bmatrix} 8125 & -3750 \\ -3750 & 2500 \end{bmatrix}.$$

Com esta técnica, podemos calcular M^{17} com a mesma facilidade:

$$M^{17} = Q \Lambda^{17} Q^{-1} =$$

$$\begin{bmatrix} 80000152587890625 & -39999694824218750 \\ -39999694824218750 & 20000610351562500 \end{bmatrix}.$$

Antes de continuar, vamos a algumas definições:

M a matriz quadrada sob análise.

$auto(M) = \{\lambda_1, \lambda_2, \dots\}$ a lista de autovalores de M . Obtidos a partir de $\det|M - \lambda \mathbb{I}| = 0$

$p(\lambda) = \det|M - \lambda \mathbb{I}|$ polinômio característico de M .

$\{\vec{v}_{\lambda_1}, \vec{v}_{\lambda_2}, \dots\}$ autovetores de M . Obtidos a partir de $M\vec{v} = \lambda\vec{v}$.

$\Lambda \equiv diag(\lambda_1, \lambda_2, \dots)$ versão diagonalizada de M

Q matriz cujas colunas são os autovetores de M .

$$Q \equiv \begin{bmatrix} | & & | \\ \vec{v}_{\lambda_1} & \dots & \vec{v}_{\lambda_n} \\ | & & | \end{bmatrix}$$

Q^{-1} = a matriz inversa de Q .

Como calcular

Primeiro vamos resolver à mão 2 problemas: de ordem 2 e de ordem 3. Com isso vai-se aprender como as manipulações são feitas. Depois de entendida essa parte, vamos ver como usar o ferramental computacional à nossa disposição. Dou a palavra a Willian Press e outros, no seu livro *Métodos Numéricos Aplicados - rotinas em C++* que em sua página 588 diz *...você já deve ter concluído que achar autovalores e autovetores é complicado. Sim, é. É um dos poucos tópicos deste livro para o qual recomendamos que use rotinas enlatadas.*

Caso 1: Seja calcular os autovalores e autovetores

de $A = \begin{bmatrix} -3 & 4 \\ 1 & -2 \end{bmatrix}$

Usa-se a definição $\det|A - \lambda I| = 0$

$$\det \begin{vmatrix} -3-\lambda & 4 \\ -1 & 2-\lambda \end{vmatrix} = 0$$

$$(-3-\lambda)(2-\lambda) - (-1 \times 4) = 0$$

$$-6 + 3\lambda - 2\lambda + \lambda^2 + 4 = 0$$

$$\lambda^2 + \lambda - 2 = 0 \text{ (chamada equação característica).}$$

Agora você pode (deve) resolver esta equação como quiser: baskhara, $x^2 - Sx + P$, $(x - r_1)(x - r_2)$... Fazendo isso tem-se $\lambda_1 = -2$ e $\lambda_2 = 1$, que são os nossos autovalores.

Achando os autovetores pela definição $A\vec{v} = \lambda\vec{v}$ e fazendo $\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ fica:

$$\begin{bmatrix} -3 & 4 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}, \text{ para } \lambda = 1$$

$$\begin{bmatrix} -3 & 4 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 1 \begin{bmatrix} x \\ y \end{bmatrix} \text{ dá origem ao sistema}$$

$$\begin{aligned} -3x + 4y &= x \\ -1x + 2y &= y \end{aligned}$$

$$\begin{aligned} -x &= -y \text{ ou } x = y. \text{ O primeiro autovetor tem } x = y \text{ ou seja quaisquer dois valores iguais formam um autovetor. Pegando o caso mais simples nosso primeiro autovetor é } \vec{v}_1 = (1, 1) \end{aligned}$$

$$\text{para } \lambda = -2$$

$$\begin{bmatrix} -3 & 4 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = -2 \begin{bmatrix} x \\ y \end{bmatrix} \text{ dá origem ao sistema}$$

$$\begin{aligned} -3x + 4y &= -2x \\ -1x + 2y &= -2y \end{aligned}$$

$$\begin{aligned} -x &= -4y \text{ ou } x = 4y. \text{ O segundo autovetor tem } x = 4y \text{ ou seja nosso segundo autovetor é } \vec{v}_2 = (4, 1) \end{aligned}$$

$$\text{Finalmente se } A = \begin{bmatrix} -3 & 4 \\ -1 & 2 \end{bmatrix} \text{ então } \lambda =$$

$$(1, -2) \text{ e } Q = \begin{bmatrix} 1 & 4 \\ 1 & 1 \end{bmatrix}. \text{ Use estes valores para com-}$$

$$\text{provar (usando um computador) as expressões envolvendo } Q, \Lambda \text{ e } Q^{-1} \text{ acima.}$$

$$(1, -2) \text{ e } Q = \begin{bmatrix} 1 & 4 \\ 1 & 1 \end{bmatrix}. \text{ Use estes valores para com-}$$

$$\text{provar (usando um computador) as expressões envolvendo } Q, \Lambda \text{ e } Q^{-1} \text{ acima.}$$

Caso 2: Seja calcular os autovalores e autovetores

$$\text{de } A = \begin{bmatrix} 1 & -6 & 3 \\ 0 & 5 & -2 \\ 0 & 2 & 0 \end{bmatrix}.$$

Para calcular os autovalores faz-se

$P(\lambda) = \det|A - \lambda I| = 0$ e daqui

$$P(\lambda) = \det \begin{vmatrix} 1-\lambda & -6 & 3 \\ 0 & 5-\lambda & -2 \\ 0 & 2 & -\lambda \end{vmatrix} = 0$$

Usando o Teorema de Laplace para calcular o determinante e aproveitando-se de que a fila representada pela 1ª coluna tem 2 zeros, fica

$$(1 - \lambda) \cdot (-1)^{1+1} \cdot \begin{vmatrix} 5-\lambda & -2 \\ 2 & -\lambda \end{vmatrix} + 0 \dots + 0 \dots = 0$$

$$(1 - \lambda)[- \lambda(5 - \lambda) - (-4)] = (1 - \lambda)[\lambda^2 - 5\lambda + 4] = 0 \text{ e resolvendo tem-se } \lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 4 \text{ ou os autovalores são } \lambda = (1, 1, 4).$$

Calculando os autovetores, e começando com $\lambda_1 =$

1 (valor duplicado), da definição $A\vec{v}_1 = \lambda_1\vec{v}_1$ e fica

$$A = \begin{bmatrix} 1 & -6 & 3 \\ 0 & 5 & -2 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 1 \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Esta equação dá origem ao sistema

$$\begin{aligned} x - 6y + 3z &= x \\ 5y - 2z &= y \\ 2y &= z \end{aligned}$$

que resolvida dá a resposta $(x, y, 2y)$ com $x, y \in \mathbb{R}^*$ ou melhor escrevendo $\vec{v} = x(1, 0, 0) + y(0, 1, 2)$. Note que achamos as respostas para x e y uma vez que o valor $\lambda_1 = \lambda_2 = 1$.

Agora achando o autovetor para $\lambda_3 = 4$

O sistema agora é

$$\begin{aligned} x - 6y + 3z &= 4x \\ 5y - 2z &= 4y \\ 2y &= 4z \end{aligned}$$

que resolvida dá a resposta $x = -3z, y = 2z, z = z$ com $z \in \mathbb{R}^*$ ou melhor escrevendo $\vec{v}_3 = z(-3, 2, 1)$. Daqui, para o A acima, tem-se $\lambda = \{1, 1, 4\}$ e

$$Q = \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}$$

De novo, o convite para refazer os cálculos usando um computador. Finalmente, não esqueça que qualquer múltiplo de um autovetor é também um autovetor.

Usando o computador

`q=sp.Matrix([[1,0,-3],[0,1,2],[0,2,1]])`

`qi=q.inv()`

`qi`

`Matrix([`

`[1, 2, -1],`

`[0, -1/3, 2/3],`

`[0, 2/3, -1/3]])`

`lam=sp.Matrix([[1,0,0],[0,1,0],[0,0,4]])`

`q*lam*qi`

`Matrix([`

`[1, -6, 3],`

`[0, 5, -2],`

`[0, 2, 0]])`

`q=sp.Matrix([[1,4],[1,1]])`

`qi=q.inv()`

`qi`

`Matrix([`

`[-1/3, 4/3],`

`[1/3, -1/3]])`

`lam=sp.Matrix([[1,0],[0,-2]])`

`q*lam*qi`

`Matrix([`

`[-3, 4],`

`[-1, 2]])`

`import sympy as sp # importação do pacote sympy`

`sp.var('x') # define x como simbólica`

`m = sp.Matrix([[1,2,...],[7,8,...],...]) # matriz`

`d=sp.det(m) # acha o determinante de m`

`sp.solve(sp.det(m)) # supondo que matriz contem x`

`m.eigenvals() #devolve dic {valor:multiplic.}`

`z=m.diagonalize() # retorna Q e LAMBDA`

`z[0]*z[1]*z[0]**-1 # esta expressão retorna m`

`m.eigenvecs() # autovetores: lista de tuplas:`

`# autovalor, multiplicidade, autovetores`

Exemplos feitos: considere o mesmo enunciado com os valores $[5 \ 1 \ 3 \ 4]$. Os autovalores serão $9/2 - \sqrt{13}/2$ e $9/2 + \sqrt{13}/2$.

Mais um: com os valores $[2 \ 6 \ 1 \ 1]$ os autovalores serão -1 e 4 .

Para você fazer

Você deve achar os autovalores para 2 matrizes, ambas 2×2 . Desenvolva os raciocínios e os cálculos envolvendo entregando todo o material junto com esta folha. Ache os valores de λ_1 e λ_2 e transcreva no local adequado. Pode deixar as raízes eventualmente existentes sem precisar calculá-las.

1. Considere a matriz 2×2 contendo os seguintes elementos

$$\begin{bmatrix} 5 & 1 & 4 & 5 \end{bmatrix}$$

2. E também outra matriz de mesma forma

$$\begin{bmatrix} 2 & 5 & 2 & 3 \end{bmatrix}$$

Responda aqui

1. λ_1	1. λ_2
2. λ_1	2. λ_2



Aplicações da Álgebra Linear

Sequência de Fibonacci: Vamos estudar um truque interessante para calcular o N -ésimo termo da sequência de Fibonacci = $\{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots\}$. Estes termos começam com $a_0 = 0$, $a_1 = 1$, $a_2 = 1$ e a partir daí cada novo termo é obtido somando-se os dois anteriores:

$$a_0 = 0, a_1 = 1, a_n = a_{n-1} + a_{n-2} \quad \forall \quad n \geq 2$$

O nome técnico para uma fórmula como esta é relação de recorrência. Ao usar esta fórmula para calcular o 500º termo, você terá que executar cerca de 500 cálculos para obter o valor desejado. Vamos estudar um truque usando autovalor que vai nos permitir calcular a_N em apenas 5 etapas de cálculo simbólico não importando quando grande seja N .

Primeiro, vamos traduzir a relação de recorrência em um produto matricial.

$$\begin{matrix} a_{n+1} = a_n + a_{n-1} \\ a_n = a_n \end{matrix} \Rightarrow \underbrace{\begin{bmatrix} a_{n+1} \\ a_n \end{bmatrix}}_{\vec{a}_n} = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_A \underbrace{\begin{bmatrix} a_n \\ a_{n-1} \end{bmatrix}}_{\vec{a}_{n-1}}$$

Agora, para calcular o n -ésimo termo, começa-se pelo vetor coluna inicial $\vec{a}_0 = (a_1, a_0)^T$ e multiplica-se repetidamente pela matriz A :

$$\begin{bmatrix} a_{n+1} \\ a_n \end{bmatrix} = A^N \begin{bmatrix} a_1 \\ a_0 \end{bmatrix}$$

Pode-se extrair a_N do vetor $\vec{a}_N$ calculando o produto escalar de $\vec{a}_N$ com o vetor $(0, 1)$. Esta operação de produto escalar tem o efeito de selecionar a segunda entrada do vetor $\vec{a}_N$. Assim, se obtém a fórmula compacta para calcular o N -ésimo termo da sequência

$$a_N = (0, 1)A^N(1, 0)^T$$

Usando o truque já estudado (vivx078) da auto-composição pode-se continuar. Inicialmente deve-se encontrar a autocomposição de A

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 \\ \frac{1}{\varphi} & -\varphi \end{bmatrix}}_Q \underbrace{\begin{bmatrix} \varphi & 0 \\ 0 & \frac{-1}{\varphi} \end{bmatrix}}_\Lambda \underbrace{\begin{bmatrix} \frac{5+\sqrt{5}}{10} & \frac{\sqrt{5}}{5} \\ \frac{5-\sqrt{5}}{10} & -\frac{\sqrt{5}}{5} \end{bmatrix}}_{Q^{-1}}$$

Aqui $\lambda_1 = \varphi = \frac{1+\sqrt{5}}{2} \approx 1.618\dots$ (proporção áurea) e $\lambda_2 = \frac{-1}{\varphi} = \frac{1-\sqrt{5}}{2} \approx 0.618\dots$ (o inverso negativo da proporção áurea) são os dois autovalores de A . As colunas de Q contêm os autovetores correspondentes de A . Pode-se calcular A^N usando a fórmula

$$A^N = \underbrace{AA \dots A}_{N \text{ vezes}} = Q \underbrace{\Lambda^N}_{N \text{ vezes}} Q^{-1} = Q \Lambda^N Q^{-1}$$

Para calcular A^N é suficiente calcular as n -ésimas potências dos autovalores $\lambda_1 = \varphi$ e $\lambda_2 = \frac{-1}{\varphi}$. Por exemplo, para calcular a_5 o quinto elemento da sequência, calcula-se A^5 usando $Q \Lambda^5 Q^{-1}$ e então usa-se a fórmula $a_5 = (0, 1)A^5(1, 0)^T$:

$$\begin{aligned} a_5 &= [0 \ 1] \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= [0 \ 1] Q \begin{bmatrix} \varphi^5 & 0 \\ 0 & \frac{-1}{\varphi^5} \end{bmatrix} Q^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= [0 \ 1] \begin{bmatrix} 8 & 5 \\ 5 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = [0 \ 1] \begin{bmatrix} 8 \\ 5 \end{bmatrix} = 5. \end{aligned}$$

Pode-se calcular A^{55} com a mesma facilidade:

$$A^{55} = Q \Lambda^{55} Q^{-1} =$$

$$\begin{bmatrix} 225851433717 & 139583862445 \\ 139583862445 & 86267571272 \end{bmatrix},$$

então calcule a_{55} usando a fórmula

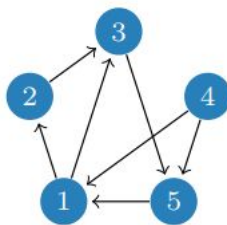
$$a_{55} = (0, 1)A^{55}(1, 0)^T = 139583862445.$$

Como não existe almoço grátis, há alguns problemas com esta abordagem:

- O cálculo de φ^N não é um cálculo de tempo constante, o que não é realista em nenhum computador.
- Manipulações em variáveis simbólicas sobre o número para números muito grandes são aproximadas, de modo que este cálculo não será exato para N grandes.

Não foi por outra razão que o exemplo escolhido acima é de $N = 55$ o que permite uma raiz exata no cálculo de φ .

Movimentação em grafos: Um grafo é um modelo matemático que descreve uma rede. Esta é formada por um conjunto de nodos e de interconexões entre eles. Chamam-se os nodos de vértices e as conexões de arestas. O grafo então é definido como um par de conjuntos $G = \{V, E\}$ onde V é o conjunto de nodos e E o de arestas. As arestas também podem ser definidas pela matriz de adjacência do grafo. Ao invés de aprofundar a definição matemática de grafo, vamos a um exemplo prático. Seja o grafo



no qual há 7 estradas de mão única entre 5 cidades. O grafo da Figura é representado matematicamente como $G = (V, E)$, onde $V = \{1, 2, 3, 4, 5\}$ é o conjunto de vértices, e $E = \{(1, 2), (1, 3), (2, 3), (3, 5), (4, 1), (4, 5), (5, 1)\}$ é o conjunto de arestas. Observe que a aresta do vértice i ao vértice j é representada como o par (i, j) .

A representação da matriz de adjacência deste gráfico na figura é uma matriz A 5×5 que contém as informações sobre as arestas do gráfico. Especificamente, $A_{ij} = 1$ se a aresta (i, j) existir, caso contrário, $A_{ij} = 0$ se a aresta não existir:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Cada linha contém 1 nas posições onde existem arestas. A representação da matriz de adjacência funciona em conjunto com os rótulos inteiros dos vértices: o vértice 1 corresponde à primeira linha de A , o vértice 2 à segunda linha e assim por diante para as outras linhas. Não precisamos de rótulos para os vértices, pois os rótulos podem ser deduzidos de sua posição na matriz A .

A matriz de adjacência de um grafo pode ser usada para responder certas questões sobre as propriedades do grafo. Por exemplo, as potências da matriz de adjacência A nos fornecem informações sobre a conectividade no gráfico. O número de maneiras de ir do vértice i ao vértice j em duas etapas é $(A^2)_{ij}$ - a ij -ésima entrada de A^2 :

$$A^2 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

O número de maneiras de ir do vértice i ao vértice j em zero, uma ou duas etapas é $1 + A + A^2$:

$$1 + A + A^2 = \begin{bmatrix} 1 & 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Observe que a maior parte do grafo G é bem conectada, exceto o vértice 4, que não possui arestas de entrada. A única maneira de chegar ao vértice 4 é começarmos por aí. O fato de podermos discutir a conectividade de grafos fazendo álgebra matricial é incrível quando você pensa sobre isso! A entrada $(1 + A + A^2)_{41} = 2$ nos diz que há duas maneiras de ir do vértice 4 ao vértice 1 em duas etapas ou menos. Na verdade, podemos fazer a transição diretamente através da aresta $(4, 1)$ em uma etapa, ou indiretamente através do Nó 5 em duas etapas, passando pelas arestas $(4, 5)$ e $(5, 1)$. Em vez de contar manualmente todos os caminhos possíveis entre os vértices, podemos calcular todos os caminhos possíveis de uma só vez usando álgebra matricial na matriz de adjacência A .

A análise da conectividade entre vértices é utilizada em muitos domínios. Os grafos são usados para descrever fluxos de rede, problemas de correspondência, redes sociais, páginas da web, mapas rodoviários, conectividade aérea e muitos outros. Em todos esses domínios, a matriz de adjacência desempenha um papel fundamental na representação gráfica.

Para você fazer

1. Calcule o termo 60 da sequência de Fibonacci, usando o truque da decomposição $Q \Lambda Q^{-1}$ vista acima. Se quiser, use este trecho de programação Python, para calcular a mesma coisa e compare os resultados obtidos. **Para facilitar a digitação/correção, só escreva os 3 ÚLTIMOS DÍGITOS do resultado.**

```
difibo = {0:1, 1:1} #um dicionario
def fibona(n):
    if n in difibo:
        return difibo[n]
    else:
        novo = fibona(n-1) + fibona(n-2)
        difibo[n] = novo
        return novo
print(fibona(n))
```

Para seu governo, os 3 últimos dígitos do 80º elemento da sequência de Fibonacci são 685.

2. Suponha uma rede de cidades e estradas de mão única determinadas pela seguinte matriz 9×9 de adjacência.

0	1	0	0	0	0	1	1	0
0	0	1	0	1	0	0	0	1
0	1	0	0	0	1	1	1	0
1	0	0	0	1	1	1	1	0
0	0	0	1	0	1	0	0	0
0	0	1	1	1	0	0	1	0
1	0	1	1	1	0	0	0	1
1	0	0	0	0	1	0	0	0
1	1	0	1	0	1	0	0	0

Descubra e escreva abaixo quantos caminhos há entre o nodo 7 e o nodo 9, considerando caminhos diretos, e passando por 1 ou por 2 nodos intermediários.

Para seu governo a quantidade de caminhos direto, passando por 1 ou 2 nodos, nessa mesma matriz entre o nodo 5 e o nodo 3 é de 3.

Responda aqui:

3 últimos díg.	qtd caminhos



- 1 -

Introdução a Python

Nascido no finzinho do século XX, e portanto herdeiro de:

- centenas de boas linguagens de programação que vieram antes e que – cada uma – e todas, tinham coisas boas ou ótimas que deviam ser aproveitadas (C, smalltalk, APL, Pascal, C++, Java,...). Aproveitadas é modo educado de falar: as coisas deviam ser copiadas *as is*.
- máquina sobrando: até aqui, faziam-se das tripas coração para economizar uns poucos bytes ou alguns ciclos de CPU. Isto não era mais necessário: havia e há máquina sobrando.

Tenha essas duas coisas em mente ao procurar entender porque o Python é a linguagem mais usada no mundo em 2024 e mais além. Se tiver alguma dúvida consulte <https://www.tiobe.com/tiobe-index/>. Aqui vale a pena lembrar aquela velha propaganda: *Tostines vende mais porque é fresquinho ou é fresquinho porque vende mais ?*

Instalação

Na continuação, você precisa instalar e usar algum Python. As alternativas

- sob windows: vá em python.org.br e siga as instruções para ter o ambiente no seu computador (você precisa ser o administrador).
- sob android: vá na loja do android e busque `python`. Escolha uma opção (pode ser o QPython 3) e mande ver. Mas, tenha em mente que a ergonomia aqui não é o ponto forte (basicamente pela dificuldade de usar o teclado).
- em um pendrive: Se você tiver que usar computadores públicos e/ou emprestados, uma alternativa é ir a algum repositório público (por exemplo sourceforge.net) e lá baixar o `windowspython` possivelmente em um pendrive: você terá um ambiente Python completo podendo ser usado em qualquer computador. Só precisa ter alguma paciência que o dito cujo é grande.
- Jupyter: um ambiente sui-generis, mistura de compilador com editor de textos (este usando o padrão Markdown). Funciona em uma interface HTML (usada na web) e permite misturar textos e códigos. Aceita igualmente JULIA e há outros ambientes na fila (C++?).

Freeware - Não é desimportante que o Python entregue tudo o que ele entrega a custo zero. Seus concorrentes em boa medida cobram licenças de centenas ou milhares de dólares. Sem contar que a comunidade Python é a maior do mundo.

Pacotes - milhares deles. Eis alguns que podem interessar:

NUMPY,	serão objeto de aulas mais à frente
SYMPY	e frente
MATPLO-TLIB	
ASTROPY	Astronomia, mas com profundo impacto no tratamento de imagens
TENSORFLOW	A rede neural do Google. A base para aprendizado profundo
PYGAME	A base para gerar jogos e animações de maneira leve
PANDAS	ferramentas para Data analisys
OPENCV	Bibliotecas para computer vision
TURTLE	A célebre Tartaruga de Seymour Papert no ambiente Logo
NLTK	Natural Language Toolkit
PILLOW	Ferramentas para tratar imagens digitais

Algumas pérolas

Digite o programa abaixo. Ele calcula e retorna o fatorial de um número inteiro e positivo (a indentação precisa ser religiosamente observada)

```
def fat(x):
    if x>1:
        return x*fat(x-1)
    else:
        return 1
```

Tendo digitado e salvo (escolha um nome qualquer) o programa acima, é hora de executá-lo: Chame `fat(5)` e a resposta deverá ser, como sabemos, `120`. Agora chame `fat(100)` e você deverá ter um número enorme. Veja que nenhuma modificação foi necessária no código.

Baskara: A seguir, digite o programa abaixo, que como se pode ver, acha as 2 raízes de uma equação de segundo grau: $ax^2 + bx + c = 0$.

```
def baskara(a,b,c):
    return [(-b+(b**2-4*a*c)**0.5)/2*a, \
            (-b-(b**2-4*a*c)**0.5)/2*a]
```

A seguir, de volta ao prompt do Python digite `baskara(1,4,-10)` e veja o resultado. Aparecem x_1 e x_2 , nenhuma novidade. Mas, agora execute `baskara(1,4,10)`. Perceba que agora as raízes são complexas, mas, de novo, nenhuma novidade, exceto que talvez a unidade imaginária que geralmente é i agora virou j pelo risco de confusão com alguma coisa.

Volume de uma esfera: A questão agora é calcular o volume de uma esfera de raio dado. Como se sabe da geometria $V = 4/3\pi \times r^3$. Então, o programa Python que calcula este volume é

```
def volesf(r):
    pi=3.14159265
    return pi*r**3
```

Agora chamando `volesf(1)` teremos a resposta que é igual a `3.14159265`

Uma linguagem fácil de aprender

A seguir, alguns conceitos introdutórios que nos ajudarão a começar a navegação no oceano chamado Python. Lembrando que estamos falando de máquinas de Von Neumann e seus 5 comandos: (E/S, movimentação, aritmética e lógica, condicional e desvio).

atribuição a criação de uma variável, ou a alteração de seu valor

Pseudo-cód	C++	Python
A ← valor	int a=valor;	a=valor

O comando de atribuição pode ser dividido em 3 possibilidades:
var = constante, como por exemplo: `a=0`
var = var2, como por exemplo: `a=b`
var = expressão, como por exemplo: `a=a+1`

Aritmética Tudo o que sabemos da matemática vale aqui, sem exceção.

Pseudo-cód	C++	Python
$a = a + 1$	<code>a++;</code>	<code>a=a+1</code>
$a = a^2$	<code>pow(a,2);</code>	<code>a=a**2</code>
$a = a - 3$	<code>a=a-3;</code>	<code>a=a-3</code>
$a = a * x$	<code>a=a*x;</code>	<code>a=a*x</code>
$a = a/x$	<code>a=a/x;</code>	<code>a=a/x (int)</code> <code>a=a/x (float)</code>
$\sqrt{x}$	<code>sqrt(x);</code>	<code>a**0.5</code>
$a + bi$	<code>?</code>	<code>a+bj</code>
$a * (b + c)$	<code>a*(b+c);</code>	<code>a*(b+c)</code>
$b = sen(a)$	<code>b=sin(a);</code>	<code>import numpy as np</code> <code>b=np.sin(a)</code>

entrada de dados - Como os dados são obtidos

Pseudo-cód	C++	Python
leia a	<code>int a;</code> <code>cin>>a;</code>	<code>a=int(input('msg'))</code>

Uma maneira alternativa de fazer a “entrada de dados” meio fake é definir uma função recebendo parâmetros e imediatamente a seguir chamar essa mesma função com valores já digitados no mesmo script. Por exemplo

```
def dobro(a):
    return a*2
dobro(5)
```

Ajuda durante os testes não ter que redigitar as vezes grandes volumes de dados a cada teste. Quando tudo funcionar adequadamente, a função estará apta a funcionar com os dados da hora.

Tipos de dados - Embora o Python assuma esta responsabilidade (ao contrário de C++, por exemplo), ainda assim às vezes será necessário especificar algum tipo particular. Lembrando que os tipos básicos são `int`, `float`, `str` e `logic (True e False)`. Existem funções Python que fazem algumas conversões:

<code>int(a)</code>	converte o string a em inteiro (se possível)
<code>float(a)</code>	converte o string a em flutuante (idem)
<code>str(b)</code>	converte o numérico b em string

Saída de dados - Como os resultados são comunicados

Pseudo-cód	C++	Python
escreva a	<code>cout<< a;</code>	<code>print(a)</code>
		<code>print('msg')</code>
		<code>print('m',a)</code>

Uma grande vantagem do `print()` é que não é necessário compatibilizar os formatos numérico e caracter (como por exemplo em `print('valor = ',val)`. A função `print()` se encarrega disso.

Python é interpretado

Se isso acarreta um desempenho menor (e quem se importa ?) em compensação ele têm diversas vantagens

- uso como calculadora
- aceita scripts e não apenas programas
- permite execução passo a passo, interativa, inclusive mudando o valor de variáveis

Conceito de bloco

Para atender à programação estruturada, qualquer linguagem de programação precisa implementar o conceito de bloco: um conjunto que qualquer quantidade de comandos e/ou blocos que são tratados como se fossem um único comando.

Em C++, tal conceito é implementado pelo uso de chaves. Tudo o que estiver entre `{ e }` é tratado como um bloco. Em Python, e com o objetivo de não poluir o código, usa-se a indentação para estabelecer os blocos. Portanto, o que lá no C++ era apenas uma recomendação para ajudar o leitor (humano) a entender o código, em Python é mandatório: Um código bagunçado no Python não levará a lugar algum, certamente dando erros sintáticos ou semânticos.

Ainda falta muita coisa

Não é muita, mas é bem importante. Ainda esteve limitado aqui no Python por não dispor do comando condicional (`if condição:`) nem dos comandos de repetição (`while condição:` ou `for...`), mas eles virão nas próximas aulas.

👉 Para você fazer

Você deve arrumar um acesso a algum Python e digitar os 3 scripts acima e descobrir:

- Os 3 primeiros dígitos de `fat( 38 )`
- As raízes de $6x^2 + 23x + 17$
- O volume da esfera de raio `3.20000`

Responda neste espaço

1.	2.	3.



Intercalação

Se lembrarmos que em francês o computador chama-se *l'ordinateur* e em espanhol *el ordenador*, vê-se que o conceito de ordem é muito importante na ciência da computação. Matematicamente, um conjunto (vetor) V de n elementos está em ordem [crescente] quando $V_{i+1} \geq V_i, \forall 1 \leq i < n$.

Como ordenar um conjunto (isto é, receber um conjunto fora de ordem e devolvê-lo ordenado) é assunto vasto na ciência da computação. No fundo, ele é simples, basta trocar os elementos de lugar, até a relação acima ficar satisfeita para todos os elementos. É provavelmente o algoritmo mais estudado na história. Será visto com profundidade e rigor no segundo ano do curso, mas hoje nossa preocupação é mais simples:

Dados dois conjuntos de valores ORDENADOS, como obter um terceiro também ORDENADO que agregue os valores de ambos os conjuntos.

A este importante processo dá-se o nome de **intercalação** ou o mais comum nome em inglês de **merge**.

Por exemplo, sejam os conjuntos A e B , como segue:

A é 2 5 6 6 8 10 16 60 88 89 90 90 e
 B é 1 2 2 7 9 11 14 55

A intercalação desses conjuntos levará ao vetor

1 2 2 2 5 6 6 7 8 9 10 11 14 16 55 60 88 89 90 90

Antes de estudar o algoritmo, veja-se algumas observações:

- as entradas NÃO precisam ter o mesmo tamanho.
- O algoritmo só pode funcionar se ambas as entradas estiverem ordenadas.
- Se alguma delas não estiver, ANTES de chamar a intercalação a entrada desordenada deve ser ordenada.
- Embora o algoritmo seja estudado para duas entradas, ele pode facilmente ser generalizado para n entradas.
- Quando o algoritmo tiver duas entradas, e uma delas for vazia, a saída reproduz a outra entrada (condição de contorno).
- Perceba-se que não existe nenhuma obrigatoriedade de sincronia entre as leituras aos dois conjuntos. Não é “um lá, um cá”.
- Em resumo, o algoritmo SEMPRE manda ler (e processar) o elemento que deu menor na comparação.

O algoritmo de Intercalação

```

1: algoritmo INTERCALAÇÃO
2: // A e B são os conjuntos de entrada.
3: KA, KB : inteiro
4: se tem dados em A então
5:   KA ← leia (A)
6: senão
7:   KA ← 99999
8: fim se
9: se tem dados em B então
10:   KB ← leia (B)
11: senão
12:   KB ← 99999
13: fim se
14: enquanto KA ≠ 99999 ∨ KB ≠ 99999 faça
15:   se KA < KB então
16:     grava (KA)
17:     se tem dados em A então
18:       KA ← leia (A)
19:   senão
20:     KA ← 99999
21:   fim se
22:   senão
23:     se KB < KA então
24:       grava (KB)
25:       se tem dados em B então
26:         KB ← leia (B)
27:     senão
28:       KB ← 99999
29:   fim se
30:   senão
31:     grava (KA)
32:     grava (KB)
33:     se tem dados em A então
34:       KA ← leia (A)
35:   senão
36:     KA ← 99999
37:   fim se
38:   se tem dados em B então
39:     KB ← leia (B)
40:   senão
41:     KB ← 99999
42:   fim se
43:   fim se
44: fim enquanto
45: fim enquanto
    
```

O exercício

Esta folha, além de apresentar o algoritmo de intercalação serve também para treinar a realização de testes de mesa (chinês). Então, para poder desempenhar nesta folha:

- usando uma caneta marca-texto, destaque os comandos: 4, 5, 7, 9, 10, 12, 14, 15, 16, 17, 18, 20, 23, 24, 25, 26, 28, 31, 32, 33, 34, 36, 38, 39 e 41
- faça um chinês rigoroso com os dados que forem fornecidos, anotando quais os comandos marcados é que são executados durante o chinês
- Ao final, responda quais os comandos executados que foram pedidos.

Por exemplo, suponha que o conjunto $A = \{2, 5\}$ e o conjunto $B = \{1, 3, 4\}$. Acompanhando a execução do algoritmo, os comandos executados serão:

4 5 9 10 14 15 23 24 25 26 14 15 16 17 18 14 15 23 24 25
 26 14 15 23 24 25 26 14 15 23 24 25 28 14 15 16 17 20 14

Outro exemplo: $A = \{8, 9, 9\}$ e $B = \{7, 8, 9\}$. A execução dará:

4 5 9 10 14 15 23 24 25 26 14 15 23 31 32 33 34 38 39 14
 15 23 31 32 33 34 38 41 14 15 16 17 20 14

Agora três exemplos iguais ao que terá que ser respondido:

$A = \{5, 6, 7, 8\}$
 $B = \{6, 7, 8, 9\}$

Qual comando será o 36°, o 39° e o 40° ?

Resposta: 39 23 24

Obs: eis a sequência completa dos comandos

4 5 9 10 14 15 16 17 18 14 15 23 31 32 33 34 38 39 14 15 23
 31 32 33 34 38 39 14 15 23 31 32 33 36 38 39 14 15 23 24 25
 28 14

$A = \{7, 8, 9\}$
 $B = \{5, 6, 7, 9\}$

Qual comando será o 30°, o 35° e o 38° ?

Resposta: 18 32 38

Obs: eis a sequência completa dos comandos

4 5 9 10 14 15 23 24 25 26 14 15 23 24 25 26 14 15 23 31 32
 33 34 38 39 14 15 16 17 18 14 15 23 31 32 33 36 38 41 14

$A = \{6, 7, 8\}$
 $B = \{6, 8, 9\}$

Qual comando será o 23°, o 27° e o 30° ?

Resposta: 32 39 23

Obs: eis a sequência completa dos comandos

4 5 9 10 14 15 23 31 32 33 34 38 39 14 15 16 17 18 14 15 23
 31 32 33 36 38 39 14 15 23 24 25 28 14

Para você fazer

Seja o conjunto $A =$

4 6 7 9

E o conjunto $B =$

5 6 7 9

Informe na tabela a seguir, quais os comandos executados na vez indicada

32 °	35 °	37 °



- 1 - /

1

Exercício : 1

_____ / _____ / _____

Algoritmo BALANCE-LINE

Este é o algoritmo básico quando se pretende efetuar a manutenção “em lote” (batch) de uma massa de dados. “em lote” aqui significa o oposto de interativo. Um sistema ou um programa é interativo quando exige a presença (e o raciocínio) de seu usuário humano por perto enquanto é executado. Um programa equivalente “em lote” realiza as mesmas tarefas, porém **sem supervisão**. Em outras palavras, ao invés de comandar e verificar cada operação realizada, o usuário simplesmente relaciona o que ele quer que seja feito, coloca isso em uma fila de execução e horas ou dias depois, verifica o resultado.

Há vantagens e desvantagens em ambas as abordagens. Por exemplo, a atualização interativa é mais confiável e mais precisa: as ordens emitidas são verificadas no ato. Qualquer erro cometido é imediatamente reportado e sua correção é instantânea. Até porque, o sistema se recusa a prosseguir enquanto o erro estiver lá.

Na atualização em lote, os erros quando ocorrem, não são corrigidos, apenas reportados, e o sistema tem que dar um jeito de prosseguir. A correção virá a posteriori e eventualmente, algum retrabalho haverá, além do inevitável atraso.

Por outro lado, o ser humano é muito lento quando comparado ao computador. Atrelar a velocidade deste (a máquina) àquele (o homem) torna todo o processo muito menos eficiente. Um exemplo prático: quando a inscrição no vestibular da UFPR era em batch, um único computador inscrevia 30.000 candidatos. Hoje, são necessários 50 a 100 computadores para dar conta da mesma tarefa.

O algoritmo balance line consome poucos recursos. Não exige bancos de dados (até porque é prévio a este conceito). Usa apenas arquivos sequenciais, que só precisam estar ordenados por uma chave comum. Esta chave precisa ser única, isto é não pode haver dois registros distintos com a mesma chave.

O algoritmo possui duas entradas: o cadastro anterior e o movimento. Ambas tem o mesmo layout, a menos do código de atualização que acompanha o movimento e que pode ser (grosso modo) I=inclusão, A=alteração e E=exclusão.

Quando o código for de inclusão, 100% dos dados devem estar informados no movimento. Quando for alteração, além da chave comum, apenas os campos a alterar devem estar preenchidos. Quando for uma exclusão apenas a chave deve estar lá.

Além destas verificações (chamadas genericamente de consistências), há outras, chamadas de compatibilidades, que só podem ser descobertas quando há uma verificação cruzada entre o cadastro e o movimento. São elas: a inclusão para já existente (quando se tenta incluir um registro com a chave de outro que já existe no cadastro), alteração para inexistente (quando há uma alteração para alguém que não existe no cadastro) e exclusão para inexistente.

Embora a grande maioria dos sistemas atuais seja interativo (os computadores são baratos e o mais importante é evitar erros nos cadastros), este algoritmo segue importante por uma razão singela: Os melhores sistemas (por exemplo, o cadastro LATTES do CNPQ) oferecem as duas modalidades de atualização. Quando o volume é muito, obrigar o usuário a usar o modo interativo é submetê-lo a horas de quase sofrimento. Assim, os bons sistemas permitem que para grandes volumes o usuário use a modalidade em lote e depois para os ajustes finos, use a modalidade interativa.

1. Considere o seguinte algoritmo

```

1: Algoritmo Balance Line
2: estrutura CAD
3: inteiro CODIGO {número da pessoa}
4: cadeia NOME[12]
5: inteiro IDADE
6: pflutuante DIVIDA
7: fim {estrutura}
8: estrutura MOV
9: caracter CODMOV {pode ser I, A ou E}
10: CAD DA
11: fim {estrutura}
12: CAD ENT, SAI
13: MOV ME
14: inteiro função LECAD
15: leia ENT
16: se fim-de-arquivo então
17:   devolva 999999
18: senão
19:   devolva ENT.CODIGO
20: fim se
21: fim {função}
22: inteiro função LEMOV
23: leia ME
24: se fim-de-arquivo então
25:   devolva 999999
26: senão
27:   devolva ME.DA.CODIGO
28: fim se
29: fim {função}
30: inteiro CC, CM
31: CC ← LECAD
32: CM ← LEMOV
33: enquanto (CC ≠ 999999 ∨ CM ≠ 999999) faça

```

```

34: se (CC < CM) então
35:   SAI ← ENT
36:   imprima (SAI)
37:   CC ← LECAD
38: senão
39:   se (CM < CC) então
40:     se (ME.CODMOV = 'A') então
41:       "Erro: ", ME.DA.CODIGO, "nao existe para alterar"
42:     senão
43:       se (ME.CODMOV = 'E') então
44:         "Erro: ", ME.DA.CODIGO, "nao existe p/ excluir"
45:       senão
46:         SAI.CODIGO ← ME.DA.CODIGO
47:         SAI.NOME ← ME.DA.NOME
48:         SAI.IDADE ← ME.DA.IDADE
49:         SAI.DIVIDA ← ME.DA.DIVIDA
50:         imprima (SAI)
51:       fim se
52:     fim se
53:   CM ← LEMOV
54: senão
55:   se (ME.CODMOV = 'I') então
56:     "Erro: ", ME.DA.CODIGO, "ja existente"
57:     SAI ← ENT
58:     imprima (SAI)
59:   senão
60:     se (ME.CODMOV = 'A') então
61:       SAI ← ENT
62:       se (ME.DA.NOME ≠ " ") então
63:         SAI.NOME ← ME.DA.NOME
64:       fim se
65:       se (ME.DA.IDADE ≠ 0) então
66:         SAI.IDADE ← ME.DA.IDADE
67:       fim se
68:       se (ME.DA.DIVIDA ≠ 0.0) então
69:         SAI.DIVIDA ← ME.DA.DIVIDA
70:       fim se
71:       imprime (SAI)
72:     fim se
73:   fim se
74:   CC ← LECAD
75:   CM ← LEMOV
76:   fim se
77:   fim se
78: fim enquanto
79: fim algoritmo

```

2. Considere os seguintes dados:

==== CADASTRO ====			==== MOVIMENTO ====		
1zuza	18	425.00	I 1kica	18	773.00
4jose	24	260.00	I 2biba	15	1050.00
5kica	17	1057.00	A 6	19	.00
6paula	19	284.00	I 7paula	29	235.00
10zuza	27	260.00	A 8hugo	25	.00
12carmem	24	564.00	A 11	23	.00
13paulo	20	425.00	A 12luis	19	.00
15antonio	20	360.00	I 13kica	22	133.00
19jorge	19	857.00	A 15antonio	24	.00
20ana	24	597.00	E 16	0	.00
22ana	19	1033.00	I 18sara	25	381.00
23ana	18	970.00	I 22sara	27	121.00
24luis	17	857.00	A 23felipe	22	.00
25carmem	23	440.00	E 26	0	.00
27felipe	19	492.00	I 28jane	22	917.00
33paulo	19	803.00	A 29joao	25	603.00
34hugo	20	662.00	E 30	0	.00
36felipe	25	123.00	E 31	0	.00
40hugo	21	1032.00	I 32biba	28	312.00
43helio	22	530.00	I 34jorge	20	381.00
44jorge	19	764.00	I 37pedro	22	754.00
46biba	19	560.00	A 43	25	290.00
47sara	19	451.00	A 44	19	851.00
49felipe	25	451.00	A 45biba	16	.00
50jose	23	782.00	A 50biba	27	.00

3. Numere o cadastro de saída e as mensagens de erro, e responda:

- Qual o código (o número da pessoa) que existe no cadastro de saída, na linha numerada ?

26	27	31

- Qual o código (número da pessoa) que está na mensagem de erro numerada ?

1	12



Algoritmo de quebra

Este algoritmo é parte integrante de qualquer gerador de relatórios. Sempre que dados aparecem dentro de uma estrutura hierárquica de “pertença”, a sua listagem exige o cuidado da quebra.

Vamos exemplificar com um caso real. Seja um arquivo de quanto dinheiro cada aluno desta instituição gastou na cantina no último ano.

Ora, cada aluno aqui dentro pertence a:

turma turma à qual aquele aluno pertence

ano junção de todas as turmas (“A” e “B” ou “M” e “N”)

curso junção de todos os anos daquele curso

Uma possível massa de dados poderia ser:

curso	ano	turma	aluno	valor
SI	1	A	antonio	200,00
SI	1	A	ana	250,00
SI	2	A	marília	100,00
SI	4	A	paulo	50,00
SI	4	A	jose	300,00
SI	4	B	maria	20,00
TUR	1	A	quica	150,00
TUR	3	B	josefina	100,00
0	0	0		-1

Para que este algoritmo funcione adequadamente, os dados precisam estar ordenados pelos campos de quebra (no exemplo: curso, ano e turma). Se isto não ocorrer, ANTES de executar o algoritmo, os dados precisam ser ordenados.

A lógica do algoritmo é criar 4 totalizadores (um geral e mais três: para curso, ano e turma) e ir somando os valores gastos sempre que o atributo de quebra atual for igual ao mesmo atributo do registro anterior.

Submetendo estes dados ao algoritmo explicado, o resultado deverá ser:

			T.TURMA	T.ANO	T.CURSO	T.GERAL
SI	1	A	450,00			
SI	1			450,00		
SI	2	A	100,00			
SI	2			100,00		
SI	4	A	350,00			
SI	4	B	20,00			
SI	4			370,00		
SI					920,00	
TUR	1	A	150,00			
TUR	1			150,00		
TUR	3	B	100,00			
TUR	3			100,00		
TUR					250,00	
TOTAL						1170,00

Formalmente falando, eis o algoritmo, que além de totalizar o valor também incrementa um contador de alunos em cada categoria. Esta último informação é boa para fazer estatísticas, obter médias, etc.

Veja-se o algoritmo

```

1: algoritmo quebra
2: TOTT, TOTA, TOTC, TOTG, QT, QA, QC, QQ : inteiro
3: TOTT ← TOTA ← TOTC ← TOTG ← 0
4: QT ← QA ← QC ← QQ ← 0
5: leia (CUR,ANO,TUR,VALOR)
6: CURANT ← CUR
7: ANOANT ← ANO
8: TURANT ← TUR
9: enquanto VALOR ≠ -1 faça
10:   TOTG ← TOTG+VALOR
11:   QQ ← QQ+1
12: se CUR=CURANT ∧ ANO=ANOANT ∧ TUR=TURANT então
13:   TOTC ← TOTC + VALOR
14:   TOTA ← TOTA + VALOR
15:   TOTT ← TOTT + VALOR
16:   QC ← QC+1; QA ← QA+1; QT ← QT+1
17:   ...opcional... imprime os dados do aluno...
18: senão
19: se CUR=CURANT ∧ ANO=ANOANT então
20:   TOTC ← TOTC + VALOR
21:   TOTA ← TOTA + VALOR
22:   imprime "TOTAL DA TURMA ", TURANT, TOTT, QT
23:   QC ← QC+1; QA ← QA+1
24:   QT ← 1
25:   TURANT ← TUR
26:   TOTT ← VALOR
27: senão
28: se CUR=CURANT então
29:   TOTC ← TOTC + VALOR
30:   imprime "TOTAL DA TURMA ", TURANT, TOTT, QT
31:   imprime "TOTAL DO ANO ", ANOANT, TOTA, QA
32:   QC ← QC+1
33:   QA ← QT ← 1
34:   TURANT ← TUR
35:   ANOANT ← ANO
  
```

```

36:   TOTA ← TOTT ← VALOR
37: senão
38:   imprime "TOTAL DA TURMA ", TURANT, TOTT, QT
39:   imprime "TOTAL DO ANO ", ANOANT, TOTA, QA
40:   imprime "TOTAL DO CURSO ", CURANT, TOTC, QC
41:   QC ← QA ← QT ← 1
42:   TURANT ← TUR
43:   ANOANT ← ANO
44:   CURANT ← CUR
45:   TOTC ← TOTA ← TOTT ← VALOR
46: fim se
47: fim se
48: fim se
49: leia (CUR,ANO,TUR,VALOR)
50: fim enquanto
51: imprime "TOTAL DA TURMA ", TURANT, TOTT, QT
52: imprime "TOTAL DO ANO ", ANOANT, TOTA, QA
53: imprime "TOTAL DO CURSO ", CURANT, TOTC, QC
54: imprime "TOTAL GERAL ", TOTG, QQ
55: finalgoritmo
  
```

Note que sempre que uma chave quebra (por exemplo curso=TUR e cursoanterior=SI, todos os contadores menores (turma e ano) quebram também, mesmo que o seu valor não se modifique.

Um exemplo

dados originais:	devidamente ordenados:	resultaram:
1 33 203 250	1 31 201 160	1 31 201 160 1
3 33 201 230	1 31 202 350	1 31 202 350 1
3 33 201 330	1 32 201 280	1 31 510 2
1 31 201 160	1 32 201 180	1 32 201 460 2
2 33 203 240	1 32 202 310	1 32 202 310 1
1 33 203 30	1 32 203 300	1 32 203 350 2
1 33 203 190	1 32 203 50	1 32 1120 5
3 31 203 70	1 33 203 250	1 33 203 470 3
2 31 202 340	1 33 203 30	1 33 470 3
1 31 202 350	1 33 203 190	1 2100 10
2 33 203 110	2 31 202 340	2 31 202 400 2
3 32 201 120	2 31 202 60	2 31 400 2
3 32 201 170	2 32 202 130	2 32 202 210 2
1 32 201 280	2 32 202 80	2 32 210 2
1 32 202 310	2 33 202 140	2 33 202 140 1
3 32 202 320	2 33 203 240	2 33 203 350 2
1 32 201 180	2 33 203 110	2 33 490 3
2 33 202 140	3 31 203 70	2 1100 7
1 32 203 300	3 32 201 120	3 31 203 70 1
3 32 202 20	3 32 201 170	3 31 70 1
1 32 203 50	3 32 202 320	3 32 201 290 2
2 32 202 130	3 32 202 20	3 32 202 540 3
3 32 202 200	3 32 202 200	3 32 830 5
2 32 202 80	3 33 201 230	3 33 201 560 2
2 31 202 60	3 33 201 330	3 33 560 2
As perguntas abaixo sobre estes dados		3 1460 8
teriam como respostas 3-33-201 e 1-32		4660 25

Para você fazer

Seja o conjunto de entrada (desordenado) =

3	33	201	100
3	32	201	210
1	32	201	020
1	31	203	220
1	32	203	310
3	31	202	060
1	33	201	010
1	31	201	160
3	32	201	030
1	33	203	300
2	32	202	190
2	31	201	270
1	33	203	050
2	33	202	290
3	32	203	080
3	32	201	150
2	31	201	110
1	32	202	230
2	33	202	350
2	32	202	250
1	31	203	140
2	31	202	320
2	31	201	280
1	31	201	120
1	32	201	130

1.Qual o curso/ano/turma mais gastador?	2. Qual o curso/ano que teve mais alunos cantineiros?

Se houver empate, pegue o primeiro empatante.



Sequências

Uma sequência é uma lista ordenada de números que segue algum padrão. Por exemplo, eis algumas sequências:

$$a_n = (0, 1, 2, 3, \dots) \quad a_n = n$$

$$a_n = (5, 7, 9, \dots) \quad a_n = 3 + 2n$$

$$a_n = (2, 3, 5, 7, 11, 13, \dots) \quad a_n = \text{primo}_n$$

$$a_n = (0, 1, 1, 2, 3, 5, 8, 13, \dots) \quad a_n = \text{fibbo}(n)$$

Uma sequência também pode ser estudada como uma função com a entrada n , onde n é um inteiro (o índice na sequência) e a saída é algum número $a_n \in \mathbb{R}$.

Convergência Diz-se que uma sequência a_n converge para um limite L e escreve-se como

$$\lim_{n \rightarrow \infty} a_n = L,$$

se para valores grandes de n os termos da sequência se tornam arbitrariamente próximos de L . A noção de limite de sequência é a mesma de limite de função. Por exemplo, os números da sequência de Fibonacci crescem indefinidamente ($\lim_{n \rightarrow \infty} a_n = \infty$) mas a fração $\frac{a_{n+1}}{a_n}$ converge para uma constante (conhecida como proporção áurea):

$$\lim_{n \rightarrow \infty} \frac{a_{n+1}}{a_n} = \varphi = \frac{1 + \sqrt{5}}{2} \approx 1.618033\dots$$

Se uma sequência a_n for como uma função $f(x)$ devemos ser capazes de realizar cálculos nela. Já vimos como obter limites de sequências, mas podemos calcular derivadas e integrais de sequências? As derivadas são proibidas porque dependem de $f(x)$ ser contínua e as sequências são definidas apenas para valores inteiros. Entretanto pode-se calcular integrais de sequências e esse é o assunto que vem a seguir:

Séries Uma série é uma soma de uma sequência. Ela é denotada com Σ e este sinal nos permite escrever de maneira sintética somas longas (e eventualmente de comprimento infinito). Acompanhe

$$a_3 + a_4 + a_5 + a_6 + a_7 = \sum_{3 \leq a_n \leq 7} a_n = \sum_{n=3}^7 a_n$$

Desta maneira vai-se definir $\sum a_n$ como a série a_n que vem a ser a soma de todos os termos da sequência a_n .

A propósito qual a diferença entre a sequência a_n e a série a_n ? Vamos lá: a sequência a_n é o conjunto de valores: $(a_1, a_2, a_3, \dots, a_n)$, enquanto a série a_n é um único número obtido fazendo a soma dos valores $a_1 + a_2 + a_3 + \dots + a_n$.

Somas exatas Existem algumas fórmulas para somas exatas de algumas séries. Por exemplo

$$S_N = \sum_{n=0}^N r^n = 1 + r + r^2 + \dots + r^N = \frac{1 - r^{N+1}}{1 - r}$$

Se $|r| < 1$ a expressão acima fica

$$S_N = \sum_{n \rightarrow \infty} r^n = 1 + r + r^2 + \dots + r^n = \frac{1}{1 - r}$$

$$\sum_{n=1}^N n = \frac{N(N+1)}{2}, \quad \sum_{n=1}^N n^2 = \frac{N(N+1)(2N+1)}{6}$$

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} = \ln(2)$$

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}, \quad \sum_{n=1}^{\infty} \frac{1}{n^4} = \frac{\pi^4}{90}$$

Convergência e Divergência

Mesmo quando não se sabe calcular uma expressão exata para a soma de uma série, é importante distinguir séries que convergem ou não. Diz-se que uma série $\sum a_n$ converge se a soma infinita $S_{\infty} \equiv \sum_{n \in \mathbb{N}} a_n$ é igual a algum número finito $L \in \mathbb{R}$. Há inúmeros testes de convergência e teoremas que ajudam a fazer esta análise. Veja mais em **No Bullshit guide to Math and Physics** de Ivan Savov e **Cálculo - vol 2** de James Stewart.

Série de Taylor É um polinômio aproximativo de uma função qualquer. Em muitas áreas da matemática, física e engenharia usam-se as séries de Taylor para encontrar soluções de equações diferenciais, aproximar funções complexas e estudar o comportamento de funções em um ponto.

Acompanhe:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 +$$

$$\frac{f'''(a)}{3!}(x-a)^3 + \dots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Você pode calcular a série de Taylor $c_0 + c_1(x-a) + c_2(x-a)^2 + \dots$ de qualquer função pela escolha de c_n igual ao valor na n -ésima derivada de $f(x)$ dividida pelo fatorial apropriado. Quanto mais termos da série você calcular, mais acurada sua aproximação será. A aproximação de ordem zero de uma $f(x)$ em $x = a$ é $f(a)$. Esta aproximação não é muito precisa em geral, mas pelo menos é correta quando $x = a$. A melhor aproximação linear para $f(x)$ em $x = a$ é a reta tangente $T_1(x)$ que é uma reta que passa pelo ponto $(a, f(a))$ e tem inclinação igual a $f'(a)$. Isso é exatamente o que a fórmula da série de Taylor de primeira ordem nos diz para calcular. O coeficiente na frente de x é obtido primeiro calculando $f'(x)$ e depois avaliando o resultado em $x = a$:

$$f(x) \approx f(a) + f'(a)(x-a) = T_1(x)$$

Para encontrar a melhor aproximação quadrática para $g(x)$ devemos calcular a segunda derivada $f''(x)$ e o coeficiente na frente será $f''(x)$ dividido por $2! = 2$ e

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 = T_2(x)$$

e assim por diante. Note que a escolha de a no desenvolvimento define um ponto em torno do qual a série é expandida. Quanto mais próximo de x estiver a mais precisa será a aproximação.

Veamos um exemplo

Aproxime a função $f(x) = \sqrt[3]{x}$ por um polinômio de Taylor de grau 2 em $a = 8$. Eis a solução:

$$f(x) = x^{1/3} \quad f(8) = 2$$

$$f'(x) = \frac{1}{3}x^{-2/3} \quad f'(8) = \frac{1}{12}$$

$$f''(x) = -\frac{2}{9}x^{-5/3} \quad f''(8) = -\frac{1}{144}$$

$$f'''(x) = \frac{10}{27}x^{-8/3}$$

O polinômio de segundo grau é $T_2(x) = f(8) + \frac{f'(8)}{1!}(x-8) + \frac{f''(8)}{2!}(x-8)^2$

$$\sqrt[3]{x} \approx T_2(x) = 2 + \frac{1}{12}(x-8) - \frac{1}{288}(x-8)^2$$

Veja um Python sobre o assunto

```
import sympy as sp
sp.var('x a')
a=8
f= x**(1/3.0)
f1=sp.diff(f,x)
f21=sp.diff(f1,x)
f321=sp.diff(f21,x)
t=f.subs(x,a)+f1.subs(x,a)*(x-a)+ \
    f21.subs(x,a)*(1/2.0)*(x-a)**2 + \
    f321.subs(x,a)*(1/6.0)*(x-a)**3
print(t.subs(x,9))
print('certo=',9**0.333333333333)
```

Série de Maclaurin É um caso particular da Série de Taylor obtida quando $a = 0$. Agora a série de Taylor fica

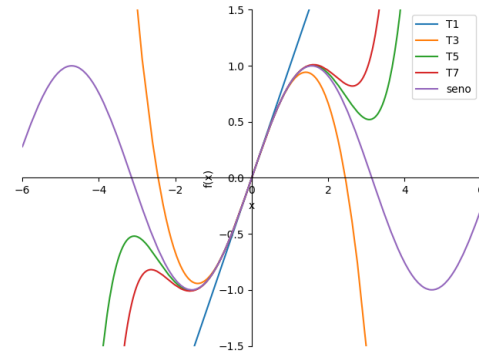
$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \dots$$

Um exemplo famoso

Encontre a série de Maclaurin para $\text{sen}(x)$. Considerando que se $f(x) = \text{sen}(x)$ então $f'(x) = \cos(x)$, $f''(x) = -\text{sen}(x)$ e $f'''(x) = -\cos(x)$ e $f^{(4)}(x) = \text{sen}(x)$ e o ciclo recomeça, pode-se escrever $\text{sen}(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots$ que é igual a $\text{sen}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$ e finalmente

$$\text{sen}(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Veja este resultado



obtido com

```
t1=x
t3=x-(x**3)/6
t5=x-(x**3)/6+(x**5)/120
t7=x-(x**3)/6+(x**5)/120+(x**7)/5040
p1=sp.plot(t1,(x,-6,6),legend=True,show=False, \
    label='T1',xlim=[-6,6],ylim=[-1.5,1.5])
p3=sp.plot(t3,show=False,label='T3')
p5=sp.plot(t5,show=False,label='T5')
p7=sp.plot(t7,show=False,label='T7')
pseno=sp.plot(sp.sin(x),(x,-6,6),show=False, \
    label='seno')
p1.show()
```

Uma razão pela qual as séries de Taylor são importantes é que elas permitem integrar funções com as quais não se pode lidar. Newton frequentemente integrava funções como séries de potências. Por exemplo, a função $f(x) = e^{-x^2}$ não pode ser integrada porque sua antiderivada (primitiva) não é uma função elemental.

Finalmente, fica a dica: programe a solução via Taylor e se tiver que achar uma série de Maclaurin, basta fazer $a = 0$ no programa que escreveu.

Para você fazer

Resolva as seguintes séries conforme pedido. Para cada uma, calcule 4 parcelas (portanto até $f'''(x)$) e depois calcule $f(x)$ no ponto pedido.

1. Ache a série de Maclaurin para $f(x) = \text{senh}(x)$ e depois ache o valor numérico para $x = 0.4$ com 3 casas decimais.
2. Ache a série de Maclaurin para $f(x) = (1-x)^{-2}$ e depois ache o valor numérico para $x = 0.3$ com 3 casas decimais.
3. Ache a série de Maclaurin para $f(x) = xe^x$ e depois ache o valor numérico para $x = 0.4$ com 3 casas decimais.

Responda aqui:

1	2	3



- 1 -

Listagem de todos os problemas de 088

1. Ache a série de Maclaurin para $f(x) = (1 - x)^{-2}$ e depois ache o valor numérico para $x = 0.2$ com 3 casas decimais. 1.552
2. Ache a série de Maclaurin para $f(x) = (1 - x)^{-2}$ e depois ache o valor numérico para $x = 0.3$ com 3 casas decimais. 1.978
3. Ache a série de Maclaurin para $f(x) = 2^x$ e depois ache o valor numérico para $x = 0.3$ com 3 casas decimais. 1.231
4. Ache a série de Maclaurin para $f(x) = 2^x$ e depois ache o valor numérico para $x = 0.2$ com 3 casas decimais. 1.148
5. Ache a série de Maclaurin para $f(x) = \sinh(x)$ e depois ache o valor numérico para $x = 0.4$ com 3 casas decimais. 0.410
6. Ache a série de Maclaurin para $f(x) = xe^x$ e depois ache o valor numérico para $x = 0.4$ com 3 casas decimais. 0.592

Otimização de funções

Este material está baseado no livro **No bullshit calculus** do autor Ivan Savov, e o livro **How to solve Calculus Problems** de Eugene Don e Benay Don, com o material traduzido por p.kantek.

Algoritmo de otimização Esta seção mostra e explica os detalhes do algoritmo para encontrar o máximo de uma função. Isso é chamado de otimização, como encontrar a solução ótima para um problema. Digamos que você tenha a função $f(x)$, que representa um fenômeno do mundo real. Por exemplo, $f(x)$ pode representar o quanto você se diverte em função do álcool consumido durante uma noite. Todos nós sabemos que com muito x , a diversão acaba e você se vê, como dizem os irlandeses, *falando com Deus no grande telefone branco*. Muito pouco x e você pode não ter coragem holandesa suficiente para conversar com aquela garota / cara da mesa do outro lado da sala. Para se divertir o máximo possível, você deseja encontrar o consumo de álcool x^* onde f assume seu valor máximo. Esta é uma das aplicações proeminentes do cálculo (estou falando de otimização, não de consumo de álcool). É por isso que você aprendeu sobre todos esses limites, fórmulas de derivadas e regras de diferenciação nas seções anteriores.

Definições

- x : a variável que quer controlar.
- $[x_i, x_f]$: o intervalo de valores a partir do qual x pode ser escolhido. Os valores de x devem obedecer $x_i \leq x \leq x_f$. Estas são as *restrições* do problema de otimização. Para o problema de otimização da bebida, $x > 0$, já que você não pode beber álcool negativo, e provavelmente $x < 2$ (em litros de bebida forte), porque mais ou menos nesse ponto uma pessoa morrerá de envenenamento por álcool. Portanto, estamos procurando a quantidade ideal de álcool x no intervalo $[0, 2]$.
- $f(x)$: a função que se deseja otimizar. Deve ser uma função *diferenciável* significando que se pode obter suas derivadas.
- $f'(x)$: a derivada de $f(x)$. A derivada informa sobre a inclinação de $f(x)$.
- máximo**: o lugar onde a função atinge um pico. Quando há muitos picos, define-se o **máximo global** como o maior deles. Todos os demais são chamados **máximos locais**.
- mínimo**: o lugar onde a função atinge um fundo de vale. Se há vários mínimos, o **mínimo global** como o menor deles. Todos os demais são chamados **mínimos locais**.
- extremo**: termo genérico que descreve tanto máximos quanto mínimos.
- ponto de sela**: um lugar onde $f'(x) = 0$ em um ponto que não nem máximo nem mínimo. Por exemplo $f(x) = x^5$ tem um ponto de sela em $x = 0$.

Suponha que alguma função $f(x)$ tenha um máximo global em x e que o valor desse máximo é $f(x) = x^*$: As seguintes notações matemáticas se aplicam

- $\operatorname{argmax}_x f(x) = x^*$: a localização (o argumento da função) onde o máximo ocorre.
- $\max_x f(x) = M$: o valor máximo.

Algoritmo para encontrar extremos

Entrada: uma função $f(x)$ e uma região de restrição $C = [x_i, x_f]$

Saída: a localização e o valor de todos os máximos de $f(x)$.

Siga este algoritmo passo a passo para encontrar os extremos de uma função:

- Primeiro, olhe para $f(x)$. Se você pode planejar, planeje. Se não, tente imaginar como é a função.
- Encontre a derivada $f'(x)$.
- Resolva a equação $f'(x) = 0$. Normalmente, haverá várias soluções. Fazer uma

lista deles. Chamaremos isso de lista de candidatos.

- Agora ache a segunda derivada de $f(x)$, a quem chamaremos de $f''(x)$. Para cada candidato x^* na lista verifique se ele é máximo, mínimo ou ponto de sela
 - se $f''(x^*) < 0$ então x^* é máximo.
 - se $f''(x^*) > 0$ então x^* é mínimo.
 - se $f''(x^*) = 0$ então o teste da segunda derivada falha. Você deve retornar para a verificação de pontos próximos $f'(x^* - \delta)$ e $f'(x^* + \delta)$ para determinar a natureza do ponto x^* .
- percorra a lista mais uma vez e rejeite todos os candidatos x^* que não satisfazem as restrições C . Em outras palavras, se $x \in [x_i, x_f]$ o candidato permanece, mas se $x \notin [x_i, x_f]$ nós o removemos já que ele não é viável. Voltando ao exemplo do consumo de álcool, se você tem uma solução candidata que diz que você deve beber 5 [L] de bebida, você deve rejeitá-la porque senão você morreria.
- Adicione x_i e x_f à lista de candidatos. Esses são os limites da região de restrição e também devem ser considerados. Se nenhuma restrição foi especificada, use a região de $-\infty < x < \infty$ e adicione $-\infty$ e ∞ à lista de candidatos.
- Para cada candidato x^* calcule o valor da função $f(x^*)$.

A lista resultante é uma coleção de extremos locais: máximos, mínimos e pontos finais. O máximo global é o maior valor da lista de máximos locais. O mínimo global é o menor dos mínimos locais. Observe que, ao lidar com pontos no infinito, como $x^* = \infty$, na verdade não calculamos um valor; em vez disso, calculamos o limite $\lim_{x \rightarrow \infty} f(x)$. Geralmente a função "explode" como $f(\infty) = \infty$ (como x , x^2 , e^x ...) cai indefinidamente $f(\infty) = -\infty$ (como $-x$, $-x^2$, $-e^x$,...) ou alcança algum valor como $\lim_{x \rightarrow \infty} \frac{1}{x} = 0$, $\lim_{x \rightarrow \infty} e^{-x} = 0$. Se a função vai para ∞ ela não tem máximo global e continua crescendo indefinidamente. Similarmente, funções que vão a $-\infty$ não tem mínimo global.

Estratégia alternativa Se você não quiser ou não puder calcular a derivada segunda, o passo que a exige pode ser substituído por

- Se $f'(x^* - 0.1)$ é positivo e $f'(x^* + 0.1)$ é negativo então x^* é máx. A função sobe, achata e depois desce. Portanto x é um pico.
- Se $f'(x^* - 0.1)$ é negativo e $f'(x^* + 0.1)$ é positivo então x^* é mín. A função desce, achata e depois sobe. Portanto x é um fundo de vale.
- Se $f'(x^* - 0.1)$ e $f'(x^* + 0.1)$ têm o mesmo sinal, o ponto x^* é um ponto de sela. Remova-o da lista de candidatos.

Exemplo1 Encontre todos os pontos de máximo e mínimo de $f(x) = x^4 - 8x^2 + 356$. Como nenhum intervalo é especificado, usaremos o intervalo padrão $x \in \mathbb{R}$. Vamos percorrer as etapas do algoritmo.

- Não se sabe o que x^4 parece, mas provavelmente é semelhante a x^2 , ela vai a ∞ tanto à esquerda como à direita.
- Usando a fórmula de derivada de polinômios, encontramos $f'(x) = 4x^3 - 16x$.
- Agora precisamos resolver $f'(x) = 4x^3 - 16x = 0$, que é o mesmo que $4x(x^2 - 4) = 0$, que é o mesmo que $4x(x - 2)(x + 2) = 0$. A lista de candidatos é $\{x = -2, x = 0, x = 2\}$.
- Para cada um destes pontos vamos verificar se é máximo, mínimo ou sela.
 - Para $x = -2$, checamos: $f'(-2.1) = 4(-2.1)(-2.1 - 2)(-2.1 + 2) < 0$ e $f'(-1.9) = 4(-1.9)(-1.9 - 2)(-1.9 + 2) > 0$ e se conclui que $x = -2$ é mínimo.
 - Para $x = 0$, checamos: $f'(-0.1) = 4(-0.1)(-0.1 - 2)(-0.1 + 2) > 0$ e $f'(0.1) = 4(0.1)(0.1 - 2)(0.1 + 2) < 0$ e se conclui que $x = 0$ é máximo.
 - Para $x = 2$, checamos: $f'(1.9) = 4(1.9)(1.9 - 2)(1.9 + 2) < 0$ e $f'(2.1) = 4(2.1)(2.1 - 2)(2.1 + 2) > 0$ e se conclui que $x = 2$ é mínimo.
- Não se tem restrição, então todos os candidatos acima são selecionados.
- Adicionamos os dois limites de restrição $-\infty$ e ∞ à lista de candidatos. Neste ponto,

nossa lista final contém $\{x = -\infty, x = -2, x = 0, x = 2, x = \infty\}$.

- Agora avaliamos a função $f(x)$ para cada um dos valores para obter pares de valores de localização $(x, f(x))$, assim $\{-\infty, \infty\}$, $(-2, 340)$, $(0, 356)$, $(2, 340)$ e (∞, ∞) . Note que $f(\infty) = \lim_{x \rightarrow \infty} f(x) = \infty^4 - 8\infty^2 + 356 = \infty$ e o mesmo vale para $f(-\infty) = \infty$.

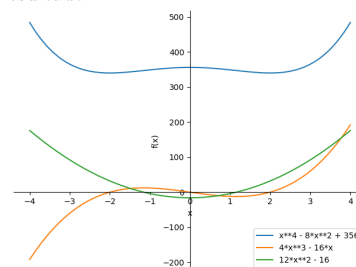
Acabamos. A função não tem máximo global, pois cresce até o infinito. Ele tem um máximo local em $x = 0$ com valor 356. Ele também tem dois mínimos globais em $x = -2$ e $x = 2$, ambos com valor 340.

Acompanhe a solução gráfica

```
import sympy as sp
x=sp.symbols('x')
e=x**4-8*x**2+356
s=sp.solve(sp.diff(e,x),x)
for i in range(len(s)):
    print(s[i].n())
f=sp.diff(e)
g=sp.diff(f)
p=sp.plot(e,f,g,(x,-4,4),legend=True)
```

Imprime: -2, 0 e 2

e o resultado



Limitações

O algoritmo de otimização acima se aplica a funções *diferenciáveis* de uma única variável. Nem todas as funções são diferenciáveis. Funções com cantos agudos, como a função de valor absoluto $|x|$, não são diferenciáveis em todos os lugares e, portanto, não funcionarão com o algoritmo acima. As funções com saltos, como a função degrau de Heaviside, não são contínuas e, portanto, não diferenciáveis o algoritmo também não pode ser usado nelas.

Podemos generalizar o procedimento de otimização, que nos ajuda a otimizar funções de múltiplas variáveis $f(x, y)$. Você aprenderá como fazer isso no curso cálculo multivariável. As técnicas de otimização serão semelhantes às etapas anteriores, mas com mais variáveis e regiões de restrição mais complexas.

Para você fazer

Resolva os seguintes problemas de maximização ou minimização.

- Genivaldo quer cercar um lote retangular com a maior área possível. Ele tem disponível 100m de cerca. Quais as dimensões do lote que ele cercará?
- Os regulamentos postais exigem que a soma do comprimento e do barbante de um pacote retangular não pode exceder 108 polegadas (barbante é a maior perimetro das 3 possibilidades de uma caixa). Qual é o volume máximo de um pacote com extremidades quadradas que atenda a este critério?
- Encontre os pontos na parábola $y = x^2$ mais próximos do ponto $(0, 1)$.

Responda aqui:

1	2	3



===== 01/01/2026 11:51:28.7 =====E=PL089

1=25 x 25m

11664 pol2

(- 0.707, 0.5)

Listagem de todos os problemas de 089

1. Genivaldo quer cercar um lote retangular com a maior área possível. Ele tem disponível 100m de cerca. Quais as dimensões do lote que ele cercará ? 25 x 25m
2. Uma caixa aberta é formada cortando 4 quadrados de igual tamanho nos cantos de uma chapa retangular de 24 por 15 polegadas e dobrando os lados. Determine as dimensões do corte que maximizam o volume da caixa que não tem tampa. $x=3\text{pol}$ e $v=486\text{pol}^3$
3. Ache as dimensões do retângulo de maior área cuja base está no eixo x e cujos dois vértices superiores estão sobre a parábola $y = 12 - x^2$. Qual a área máxima ? 4 x 8, com área=32 u.a.
4. Uma janela de igreja tem a forma de um retângulo embaixo de um semicírculo. Se o perímetro da janela é de 20 m, qual a sua área máxima ? 28 m2
5. Ache o retângulo com a maior área que pode ser inscrito em um triângulo equilátero de lado 20. $x=5$, $y=8.66$ e $\text{area}=86.6$
6. Um rio tem largura de 1km. Bento deseja ir do ponto A ao ponto B do lado oposto do rio, 3 km a jusante. Se Bento consegue correr a 5 km/hora e nadar a 3 km/hora e supondo que a velocidade da correnteza é desprezível, qual a menor quantidade de tempo que ele deve gastar para ir de A até B ? 52 minutos, com $x=3/4$
7. Qual o maior volume possível para um cilindro inscrito em uma esfera de raio=5 ? 302.3
8. Uma caixa retangular fechada com base quadrada deve ter área de 150 metros quadrados. Qual o volume máximo que essa caixa pode comportar ? 125 m3
9. Uma lata cilíndrica deve conter 2000 polegadas cúbicas de tinta. Que dimensões minimizam a quantidade de metal usado para construir a lata ? $r=6.82$ e $h=13.65\text{ pol}$
10. Qual o ponto da reta $3x + y = 6$ mais próximo do ponto (2, 3) ? (1.1, 2.7)
11. Um poster retangular com 50 polegadas quadradas de impressão deve ter margens de 2 polegadas em cada lado e de 4 polegadas em cima e embaixo. Que dimensões do poster minimizam o uso do papel ? 18 x 9 pol
12. À meia noite o barco B está a 100km à leste do barco A . Se o barco B navega a oeste a 10 km/hora e o barco A navega a sul a 20 km/hora, em que momento os dois barcos estarão mais próximos ? A que distância ? 2 horas com $d=89.44\text{ km}$
13. Um copo cônico de mate é construído com uma peça circular de papel de raio 5 polegadas pela extração de um corte em um setor e juntando o resultado das extremidades, com uma cola mágica que não exige justaposição do papel. Qual o volume máximo do copo ? 50.38 pol3
14. Um campo aberto é delimitado por um lago com uma linha costeira reta. Uma limitação retangular deve ser construída usando 500 pés de cerca ao longo de três lados e o lago como um limite natural no quarto lado. Quais dimensões irão maximizar a área fechada? Qual é a área máxima? 31.25 pes2
15. Ryan tem 800 pés de cerca. Ele deseja formar um terreno retangular e dividi-lo em três seções duas cercas paralelas a um lado. Quais devem ser as dimensões da área de forma a maximizar a área encerrada? $x=100, y=200, \text{Area}=20.000\text{ pes}^2$
16. 20 metros de cerca devem ser dispostos em forma de um triângulo retângulo. Quais devem ser suas dimensões para maximizar a área fechada? $x=y=5.85$ $z=8.284$ (isosceles)
17. Encontre a área máxima de um retângulo inscrito em um semicírculo de raio de 5 polegadas se sua base estiver ao longo do diâmetro do semicírculo. 25 pol2
18. Uma caixa aberta deve ser construída a partir de um pedaço de 12 x 12 polegadas de papelão, cortando quadrados de tamanho igual dos quatro cantos e dobrando os lados. Determine o tamanho do recorte que maximiza o volume da caixa. 128 pol3
19. Uma janela deve ser construída na forma de um triângulo isósceles no topo de um retângulo. Se o seu perímetro é de 600 cm, Qual é a área máxima possível da janela? 21087 cm2
20. Os regulamentos postais exigem que a soma do comprimento e do barbante de um pacote retangular não pode exceder 108 polegadas (barbante é a maior perímetro das 3 possibilidades de uma caixa). Qual é o volume máximo de um pacote com extremidades quadradas que atenda a este critério ? 11664 pol2
21. Um retângulo está inscrito em um triângulo retângulo cujos lados são 5, 12 e 13 polegadas. Dois lados adjacentes do retângulo ficam ao longo das pernas do O triângulo. Quais são as dimensões do retângulo de máxima área? Qual é a área máxima? 6 por 2.5 com $\text{area}=15\text{ pol}^2$
22. Encontre as dimensões do cilindro circular direito de máximo volume que pode ser inscrito em um cone circular reto cujo raio é 3 polegadas e cuja altura é 10 polegadas. Qual é o volume máximo? 41.88 pol3
23. Qual é a quantidade mínima de cerca necessária para construir um recinto retangular contendo 1.800 pés² usando um rio como um limite natural de um lado? 120 pes
24. Uma caixa retangular aberta deve ter uma base com o dobro do comprimento e largura. Se seu volume deve ser de 972 cm³, quais dimensões irão minimizar o quantidade de material utilizado em sua construção? 9x18x6 cm
25. Encontre os pontos na parábola $y = x^2$ mais próximos do ponto (0, 1). (- 0.707, 0.5)
26. Uma editora deseja imprimir um livro cujas páginas tenham cada uma área de 96 in². As margens devem ser de 1 polegada em cada um dos três lados e 2 pol no quarto lado para permitir espaço para a encadernação. Quais dimensões para as folhas vai permitir a área máxima para a região impressa? 12x8 pol
27. Uma lata cilíndrica fechada deve ter um volume de 1000 pol³. Quais dimensões irão minimizar sua área de superfície? $r=5.419$ e $h=2r=10.838\text{ pol}$
28. Uma lata cilíndrica fechada deve ter um volume de 1000 pol³. A superfície lateral deve ser construída a partir de uma peça retangular de metal e sua parte superior e inferior devem ser estampadas com peças quadradas de metal e o resto do quadrado descartado. Quais dimensões irão minimizar a quantidade de metal necessária na construção da lata? $r=5$ e $h=12.73\text{ pol}$
29. Um retângulo deve ser inscrito na elipse

$$\frac{x^2}{200} + \frac{y^2}{50} = 1$$

Determine sua área máxima possível. 200

Comparando se...então (proposição) e se...então (comando)

Duas estruturas parecidas, mas conceitualmente diferentes.

Na lógica proposicional, dadas duas proposições simples (atômicas) p e q , a junção das duas, usando o conectivo $\rightarrow$ gera uma nova proposição agora composta, R que se escreve como $R : p \rightarrow q$. O valor verdade de R vai depender dos valores verdade de p e de q de acordo com a tabela já conhecida

p	q	$\rightarrow$	$\wedge$	$\vee$	$\leftrightarrow$	$\sim$
V	V	V				
V	F	F				
F	V	V				
F	F	V				

Nas linguagens de programação (Fortran, Cobol, Basic, APL, C, C++, Java, Prolog, Lisp, Python, Maple, Freemat, Pascal, Clipper, ...) existe um comando (chamado condicional) cujo nome é **se...então (if...then)** e que determina quando um comando ou conjunto de comandos devem ou não ser executados. O formato do comando condicional **se... então ... [senão ...] fimse** é

```
se <condição> então
  comando-1
  comando-2
  ...
[senão
  comando-11
  ...]
fimse
```

Valem as seguintes observações:

- * **<condição>** é uma condição cujo resultado precisa ser binário: Verdadeiro ou Falso
- * **<condição>** pode ser simples (atômica) ou composta. No segundo caso ela pode conter $\wedge$, $\vee$ ou $\sim$. Neste caso, parênteses devem ser usados para indicar a prioridade.
- * A cláusula **senão** é opcional (por isso ela foi escrita entre colchetes). Se presente ela indica o que será executado se a condição original for falsa. Se ausente ela indica que NADA deve ser executado quando a condição original for FALSA.
- * dentro de cada ramo do comando (então e senão, este se existir) pode haver um único comando ou uma série deles. Por causa disso é que os blocos precisam ser claramente delimitados. Inclusive dentro de um comando condicional pode haver outro comando condicional: isto é um bloco dentro de outro bloco.

A programação se apropriou do conectivo se...então da lógica matemática adaptando-o a suas necessidades, como se viu. Apropriou também os conectivos E ($\wedge$), OU ($\vee$) e NÃO ($\sim$), nestes casos com adaptações mínimas. A única mudança sensível é a mudança conceitual de proposição por condição. A segunda é igual à primeira, só que precisa ser instanciada. Fora isso, é a mesma coisa.

Exemplos

Ache os valores verdade de:

1. $((\sim p) \leftrightarrow q) \leftrightarrow ((\sim q) \rightarrow p) \wedge ((\sim p) \rightarrow q)$
 2. $((p \wedge q) \rightarrow (p \leftrightarrow q))$
 3. $((p \vee q) \wedge (p \wedge p)) \vee ((\sim q) \wedge (\sim q))$
 4. $((p \vee q) \leftrightarrow (p \leftrightarrow (\sim q))) \wedge (p \wedge (\sim p)) \vee (q \vee p)$
 5. $((r \vee q) \vee (p \vee (\sim r))) \wedge ((\sim r) \vee p)$
 6. $((q \vee r) \rightarrow (p \leftrightarrow r)) \vee (q \wedge (\sim p))$
 7. $((\sim q) \leftrightarrow r) \vee (r \wedge p) \wedge (p \wedge q) \leftrightarrow ((\sim q) \leftrightarrow p)$
- Que darão como resposta: 1. FVVV 2. VVVV 3. VVVV 4. VVVV 5.VVVVVFV 6. VFVVVVVFV 7. FFFFFFFV

Ache os valores das variáveis citadas:

- 1: $A \leftarrow 7$; $B \leftarrow 5$; $C \leftarrow 3$; $D \leftarrow 8$
- 2: $se (A > 4) \wedge (B \leq 8)$

- 3: $se (C < D)$
 - 4: $C \leftarrow C + 1$
 - 5: **senão**
 - 6: $D \leftarrow D + 3$
 - 7: **fim{se}**
 - 8: **senão**
 - 9: $se (C > 3) \vee (D < 2)$
 - 10: $C \leftarrow C \times 2$
 - 11: **fim{se}**
 - 12: **fim{se}**
 - 13: **escreva** $(A + B + C)$ -D
- Cuja resposta é 8.

- 1: $A \leftarrow 5$; $B \leftarrow 6$; $C \leftarrow 8$
 - 2: $se ((A+C) > (B+6))$
 - 3: $A \leftarrow A + 5$
 - 4: $se (B > 4)$
 - 5: $A \leftarrow A + 2$
 - 6: **fim{se}**
 - 7: $se (C < 5)$
 - 8: $se (A > 11)$
 - 9: $C \leftarrow B + 2$
 - 10: **fim{se}**
 - 11: $C \leftarrow A + B$
 - 12: **fim{se}**
 - 13: **fim{se}**
 - 14: **escreva** $(A + B + C)$
- Cuja resposta é 26.

- 1: $G \leftarrow 7$; $B \leftarrow 2$; $F \leftarrow 3 + 1$
 - 2: $se (((G > B) \vee (F < 12)))$
 - 3: $se (B \geq 3)$
 - 4: $F \leftarrow F + 2$
 - 5: **fim{se}**
 - 6: $F \leftarrow F + 3$
 - 7: **fim{se}**
 - 8: $se (G \leq 5)$
 - 9: $se (((B + 2) > 6) \wedge (F > 3))$
 - 10: $B \leftarrow B + 2$
 - 11: **fim{se}**
 - 12: **fim{se}**
 - 13: $G \leftarrow G + 1$
 - 14: **escreva** $G + B + (2 \times F)$
- Cuja resposta é 24.

- 1: $H \leftarrow 3$; $B \leftarrow 5$; $J \leftarrow 1$; $L \leftarrow 4$
 - 2: $se (H < 6)$
 - 3: $se (B < 5)$
 - 4: $se (J \leq 5)$
 - 5: $K \leftarrow 8$
 - 6: **senão**
 - 7: $K \leftarrow 4$
 - 8: **fim{se}**
 - 9: $L \leftarrow K + 8$
 - 10: **fim{se}**
 - 11: $L \leftarrow L + 1$
 - 12: **fim{se}**
 - 13: **escreva** $(J + L - H)$
- Cuja resposta é 3.

👉 Para você fazer

Resolva as proposições a seguir:

1. $((p \wedge q) \vee (q \wedge p))$
2. $((q \vee p) \vee (p \vee (\sim q))) \wedge ((\sim p) \wedge q)$
3. $((p \vee q) \wedge (q \wedge (\sim q)))$
4. $((p \wedge q) \vee (q \wedge q)) \wedge (p \vee (\sim p))$
5. $((p \wedge q) \wedge (q \vee (\sim p))) \rightarrow (p \wedge q)$
6. $((\sim q) \vee p) \wedge (q \vee q) \vee (q \wedge p)$

Responda aqui: (apenas V e F)

1	2	3	4	5	6

Agora, resolva os seguintes trechos de código de programação

7.

- 1: $A \leftarrow 1$
- 2: $B \leftarrow 5$
- 3: $C \leftarrow 6$
- 4: $D \leftarrow 6$
- 5: $se (A > 5) \wedge (B \leq 8)$
- 6: $se (C < D)$
- 7: $C \leftarrow C + 1$
- 8: **senão**
- 9: $D \leftarrow D + 3$
- 10: **fim{se}**
- 11: **senão**
- 12: $se (C > 7) \vee (D < 4)$
- 13: $C \leftarrow C \times 2$
- 14: **fim{se}**
- 15: **fim{se}**
- 16: **escreva** $(A + B + C)$ -D

8.

- 1: $A \leftarrow 7$
- 2: $B \leftarrow 8$
- 3: $C \leftarrow 3$
- 4: $se ((A+C) > (B+6))$
- 5: $A \leftarrow A + 5$
- 6: $se (B > 4)$
- 7: $A \leftarrow A + 2$
- 8: **fim{se}**
- 9: $se (C < 2)$
- 10: $se (A > 11)$
- 11: $C \leftarrow B + 2$
- 12: **fim{se}**
- 13: $C \leftarrow A + B$
- 14: **fim{se}**
- 15: **fim{se}**
- 16: **escreva** $(A + B + C)$

9.

- 1: $G \leftarrow 7$
- 2: $B \leftarrow 3$
- 3: $F \leftarrow 5 + 2$
- 4: $se ((G > B) \vee (F < 12))$
- 5: $se (B \geq 3)$
- 6: $F \leftarrow F + 2$
- 7: **fim{se}**
- 8: $F \leftarrow F + 3$
- 9: **fim{se}**
- 10: $se (G \leq 5)$
- 11: $se (((B + 2) > 6) \wedge (F > 3))$
- 12: $B \leftarrow B + 2$
- 13: **fim{se}**
- 14: **fim{se}**
- 15: $G \leftarrow G + 1$
- 16: **escreva** $G + B + (2 \times F)$

10.

- 1: $H \leftarrow 8$
- 2: $B \leftarrow 8$
- 3: $J \leftarrow 6$
- 4: $L \leftarrow 2$
- 5: $se (H < 6)$
- 6: $se (B < 5)$
- 7: $se (J \leq 5)$
- 8: $K \leftarrow 6$
- 9: **senão**
- 10: $K \leftarrow 3$
- 11: **fim{se}**
- 12: $L \leftarrow K + 2$
- 13: **fim{se}**
- 14: $L \leftarrow L + 1$
- 15: **fim{se}**
- 16: **escreva** $(J + L - H)$

Responda aqui: (apenas números)

7	8	9	10



- 1 - /

Lógica Proposicional

1. Sejam as proposições p : está calor e q : está tempo seco. Traduza para a linguagem corrente as seguintes proposições:

$\sim p \wedge \sim q$	
$\sim p$	
$p \leftrightarrow \sim q$	
$q \leftrightarrow q$	

2. Sejam as proposições p : juca é alegre e q : maria é bonita. Traduza para a linguagem corrente as seguintes proposições:

$q \leftrightarrow \sim p$	
$\sim p \wedge q \rightarrow p$	
$\sim \sim p$	

3. Sejam as proposições p : berta fala espanhol e q : berta fala francês. Traduza para a linguagem corrente as seguintes proposições:

$\sim p \wedge \sim q$	
$p \wedge \sim q$	
$p \vee q$	

4. Sejam as proposições p : pedro é paranaense e q : carlos é carioca. Traduza para a linguagem corrente as seguintes proposições:

$\sim p \leftrightarrow \sim q$	
$p \rightarrow \sim q$	
$\sim (\sim p \vee \sim q)$	

5. Sejam as proposições p : maria é bonita e q : maria é esbelta. Traduza para a linguagem simbólica as seguintes proposições:

nao e verdade que maria e feia ou esbelta	
maria e bonita ou e feia e esbelta	
maria nao e nem bonita nem esbelta	

6. Sejam as proposições p : grace é gorda e q : grace é infeliz. Traduza para a linguagem simbólica as seguintes proposições:

grace e magra e feliz	
grace e magra, mas infeliz	

7. Sejam as proposições p : quica fala francês e q : quica fala inglês, e r : quica fala alemão. Traduza para a linguagem simbólica as seguintes proposições:

e falso que quica fala ingles ou alemao mas que nao fala frances	
e falso que quica fala frances mas que nao fala alemao	

8. Traduzir para a linguagem simbólica as seguintes proposições matemáticas:

$x = 1$ ou $x > 1$	
$x \neq 1$ e $x \neq 1$	

9. Traduzir para a linguagem simbólica as seguintes proposições matemáticas:

$x = 1$ e $(y + z > x$ ou $z = 1)$	
$x \neq 1$ ou $(x = 1$ e $Y < 1)$	

10. Traduzir para a linguagem simbólica as seguintes proposições matemáticas:

se $x > 2$ entao $(x = 0$ ou $x > 3)$	
se $z > 6$ entao $x \neq 0$ e $x \neq 3$	
se $x + y = 3$ entao $z < 0$	
se $x + y > z$ e $z \neq 1$ entao $x + y < 1$	

11. Simbolizar as seguintes proposições matemáticas:

x e maior do que 5 ou (x e menor que 1 e maior do que 0)	
(x e maior que 6 e menor que 8) ou x nao e igual a 6	

12. Determinar o valor lógico (V ou F) de cada uma das seguintes proposições

$((3 + 2) = 7)$ e $((5 + 5) = 10)$	
$((3 > 0)$ e $(2 + 2) \geq 4)$	
$0 > 1$ e $\sqrt{5}$ e irracional	
$(\sqrt{2}) < 1 \vee \sqrt{6}$ e racional	

13. Determinar o valor lógico (V ou F) de cada uma das seguintes proposições

$2 \leq 2$ e $\sin 90^\circ \neq \tan 45^\circ$	
einstein descobriu a penicilina ou $\sin 30^\circ = \frac{2}{5}$	
$5^2 = 10 \vee \pi > 3$	
$\sqrt{-9} = 3\sqrt{-1} \vee 17$ e um numero primo	
$-6 < -8 \vee -2 = -2$	

14. Determinar o valor lógico (V ou F) de cada uma das seguintes proposições

se $ -1 = 0$ entao $\sin 30^\circ = \frac{1}{2}$	
$\pi < 3 \rightarrow 3 > \sqrt{5}$	
se $0 < \sqrt{3}$ entao $\sqrt{3}$ e irracional	
se $\sqrt{4} > 1$ entao $-1 < -3$	

15. Determinar o valor lógico (V ou F) de cada uma das seguintes proposições

$\sqrt{2}.\sqrt{8} = 4$ se e somente se $\sqrt{3} = 0$	
$\sin 20^\circ \geq 1 \leftrightarrow \cos 20^\circ \geq 2$	
$-1 > -2 \leftrightarrow \pi^3 < 30$	
$(3^2 + 4^2 = 5^2) \leftrightarrow \pi$ e racional	
$\tan \pi = 1$ se e somente se $\sin \frac{\pi}{2} = 1$	

16. Determinar o valor lógico (V ou F) de cada uma das seguintes proposições

$1 + 1 = 2 \rightarrow (3 + 2 = 6 \leftrightarrow 1 + 1 = 3)$	
e falso que $2 + 2 = 4$ ou $\sqrt{-1} = 0$	
nao e verdade que 13 e um numero par	
$\sim (1 + 1 \neq 2$ e $2 + 6 = 8)$	

17. Determinar o valor lógico (V ou F) de cada uma das seguintes proposições

$(4^3 \neq 64) \rightarrow ((2 + 2 = 5) \leftrightarrow (2 + 2 = 4))$	
$\sim ((3^3 = 27 \rightarrow 3 = 5) \wedge 0^2 = 0)$	
$(3^4 = 81) \rightarrow \sim ((2 + 2 = 4) \wedge (5.0 = 0))$	
$\sim (2^3 \neq 8$ ou $4^2 \neq 4^3)$	

18. Sabendo que os valores lógicos das proposições p e q são respectivamente V e F, determinar o valor lógico de cada uma das seguintes proposições

$\sim p \wedge \sim q$	
$p \wedge \sim q$	
$\sim p \wedge q$	

19. Determinar $V(p)$ em cada um dos casos, sabendo:

$V(q) = F$ e $V(q \rightarrow p) = V$	
$V(q) = V$ e $V(p \leftrightarrow q) = F$	
$V(q) = F$ e $V(q \leftrightarrow p) = V$	

20. Determinar $V(p)$ e $V(q)$ em cada um dos casos, sabendo:

$V(p \leftrightarrow q) = V$ e $V(p \vee q) = V$	
$V(p \leftrightarrow q) = F$ e $V(\sim p \vee q) = V$	

Tabelas Verdade

1. Construir as tabelas verdade das seguintes proposições

$q \leftrightarrow \sim q \wedge p$	
$(p \leftrightarrow \sim q) \leftrightarrow q \rightarrow p$	
$(p \leftrightarrow \sim q) \rightarrow \sim p \wedge q$	
$\sim p \rightarrow (q \rightarrow p)$	

2. Construir as tabelas verdade para as seguintes proposições:

$p \rightarrow (p \rightarrow \sim r) \leftrightarrow q \vee r$	
$(p \wedge q \rightarrow r) \vee (\sim p \leftrightarrow q \vee \sim r)$	

3. Determinar $P(VV, VF, FV, FF)$ em cada um dos seguintes casos

$P(p, q) = (p \vee q) \wedge \sim (p \wedge q)$	
$P(p, q) = \sim ((p \vee q) \wedge (\sim p \vee \sim q))$	
$P(p, q) = \sim (\sim p \leftrightarrow q)$	
$P(p, q) = (p \wedge \sim q) \vee (\sim q)$	

4. Determinar $P(VVV, VVF, VFV, VFF, FVV, FVF, FFF, FFF)$ em cada um dos seguintes casos:

$P(p, q, r) = (p \wedge \sim q) \vee r$	
$P(p, q, r) = \sim (p \vee \sim q) \wedge (\sim p \vee r)$	
$P(p, q, r) = p \vee (q \wedge r)$	

5. Determinar $P(VFV)$ em cada um dos seguintes casos:

$P(p, q, r) = \sim p \wedge (q \vee \sim r)$	
$P(p, q, r) = p \wedge \sim r \rightarrow \sim q$	
$P(p, q, r) = (p \vee (q \rightarrow \sim r)) \wedge (\sim p \vee r \leftrightarrow \sim q)$	

6. Sabendo que os valores lógicos das proposições p e q são respectivamente F e V, determinar o valor lógico das proposições:

$(p \wedge (\sim q \rightarrow p)) \wedge \sim ((p \leftrightarrow q) \rightarrow q \vee \sim p)$	
p	

7. Sejam as proposições $p : \tan(\pi - x) = \cot x$ e $q : \pi < 2$. Determinar o valor lógico de cada uma das seguintes proposições:

$(\sim p \wedge q) \vee (p \wedge \sim q)$	
$\sim (p \wedge q) \leftrightarrow \sim p \vee \sim q$	

8. Sabendo que os valores lógicos das proposições p , q e r são respectivamente V, F e F, determinar o valor lógico de cada uma das seguintes proposições:

$(p \wedge q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$	
$(p \rightarrow \sim q) \leftrightarrow ((p \vee r) \wedge q)$	

9. Sabendo que as proposições p e q são verdadeiras e que as proposições r e s são falsas determinar o valor lógico de cada uma das seguintes proposições:

$\sim ((r \rightarrow p) \vee (s \rightarrow q))$	
$(s \leftrightarrow r) \leftrightarrow (p \leftrightarrow q)$	
$p \rightarrow \sim (r \wedge s)$	
$(q \rightarrow s) \rightarrow r$	
$(p \wedge \sim q) \vee r$	
$r \rightarrow q \leftrightarrow (\sim p \leftrightarrow r)$	

10. Sabendo que os valores lógicos das proposições p , q , r e s são respectivamente V, V, F e F determinar o valor lógico de cada uma das seguintes proposições:

$(r \rightarrow p) \rightarrow (p \rightarrow r)$	
$(p \rightarrow r) \rightarrow (\sim p \rightarrow \sim r)$	
$\sim (p \wedge s) \rightarrow \sim p \wedge \sim s$	

11. Sabendo que $V(p)=V(r)=V$ e $V(q)=V(s)=F$, determinar o valor lógico de cada uma das proposições:

$p \rightarrow \sim q \leftrightarrow (p \vee r) \wedge s$	
$(p \wedge q) \wedge (r \wedge s) \rightarrow p \vee s$	
$(\sim p \rightarrow q) \rightarrow (s \rightarrow r)$	
$p \wedge q \leftrightarrow r \wedge \sim s$	

12. Sabendo que as proposições $x = 0$ e $x = y$ são verdadeiras e que as proposições $y = z$ e $y = t$ são falsas, determinar o valor lógico de cada uma das proposições:

$x \neq y \vee y \neq z \rightarrow y = t$	
$x \neq 0 \vee x \neq y \rightarrow y \neq z$	

13. Sabendo que a condicional $p \rightarrow q$ é verdadeira, determinar o valor lógico da condicional:

$p \wedge r \rightarrow q \wedge r$	

14. Determinar o valor lógico (V ou F) de cada uma das seguintes proposições

$p \wedge q \rightarrow p \vee r$, sabendo que $V(p)=V(r)=V$	
$p \leftrightarrow q \wedge \sim r$, sabendo que $V(p)=V(r)=V$	



Proposição

Definição: uma proposição é um conjunto de palavras ou símbolos que exprimem um fato de sentido completo. Obviamente, o fato pode ser um pensamento ou juízo que se forma a respeito de determinado ente. Eis alguns exemplos de proposição:

- Curitiba é uma cidade
- Londrina fica longe do mar
- π é maior do que 3
- $\cos(0) = 1$
- esta aula é chata

A lógica proposicional adota os seguintes princípios: Uma proposição deve ser verdadeira ou falsa, nunca podendo ser as duas coisas ao mesmo tempo (princípio da não contradição) e se ela não for verdadeira será falsa e vice-versa (princípio do terceiro excluído). Por conta destes dois princípios diz-se que a lógica proposicional é bivalente.

O valor lógico de uma proposição é VERDADE (V ou T) se a proposição é verdadeira e FALSO (F) se a proposição é falsa.

Exercício 1: Ache o valor lógico das proposições a seguir, ou seja, informe se elas são V (verdadeiras) ou F (falsas).

1. Está chovendo
2. O campus da UP é grande
3. Hoje é terça-feira
4. Estou com fome
5. Vasco da Gama descobriu o Brasil
6. π é um número racional
7. falta pouco para esta aula acabar

Simples e compostas

Todas as proposições que foram vistas até agora são formadas por apenas um fato. Chama-se a estas proposições de simples ou atômicas. Nada impede que uma proposição seja composta, tendo dentro dela várias proposições simples. Uma definição formal de proposição simples é aquela que não contém nenhuma outra proposição como parte integrante de si mesma. As proposições compostas são também conhecidas como moleculares.

As proposições simples são geralmente designadas por letras minúsculas, usualmente começando na letra *p*, por exemplo

- *p* : paulo é estudante
- *q* : maria é bonita
- *r* : haroldo é um tigre
- *s* : obelix é gordo

As proposições compostas ou não atômicas são as formadas por mais de uma proposição simples e geralmente representadas pelas letras maiúsculas, a partir de *P*. Também são conhecidas como fórmulas proposicionais ou apenas fórmulas. Veja os exemplos:

- *P* : paulo é estudante e maria é bonita
- *Q* : Se andréia é estudiosa então andréia vai passar de ano
- *R* : antônio é coxabranca ou carlos é atleta-cano

Note que esta definição de fórmula é recursiva, ou seja, fórmulas podem ser compostas por fórmulas também.

Conectivos

Para construir uma proposição composta, existe a necessidade de um conectivo para juntar as duas proposições simples que a formam. Existem na lógica 5 conectivos, representados pelas palavras E, OU, NÃO, SE... ENTÃO... e SE E SOMENTE SE. Veja-se alguns exemplos de conectivos

- *P* : paulo é estudante E maria é bonita
- *Q* : SE andréia é estudiosa ENTÃO andréia vai passar de ano
- *R* : antônio é coxabranca OU carlos é atleta-cano

Segundo o princípio do terceiro excluído toda proposição simples é V ou F. Quando a proposição é composta, a determinação do seu valor lógico, pressupõe conhecidos os valores lógicos das suas proposições componentes e se obtém pelo seguinte princípio: o valor lógico de uma proposição composta depende unicamente dos valores lógicos

de seus componentes e fica por eles univocamente determinado.

A partir deste princípio para resolver o valor lógico de uma proposição composta normalmente usa-se um dispositivo simples chamado TABELA VERDADE na qual figuram todos os valores lógicos possíveis para todas as proposições simples que compõe a proposição composta.

Eis o exemplo de uma proposição composta $P : q \text{ OU } r$. Note que *p* pode ter 2 valores lógicos e igualmente *r*. Assim os valores verdade de *P* são em número de 4, a saber VV, VF, FV ou FF. Na forma de uma tabela verdade fica:

	q	r
1	V	V
2	V	F
3	F	V
4	F	F

Note que a quantidade de linhas da tabela verdade de uma proposição composta é sempre uma potência de 2. Seja por exemplo a tabela verdade de $P : ((q \text{ OU } r) \text{ E } s)$. A tabela verdade fica:

	q	r	s
1	V	V	V
2	V	V	F
3	V	F	V
4	V	F	F
5	F	V	V
6	F	V	F
7	F	F	V
8	F	F	F

Para construir a tabela verdade, deve-se usar a sequência de potências de 2, a saber: $2^n = 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024...$

NÃO

É a única operação monádica (1 único operando) e simplesmente inverte o valor lógico do operando. Assim, a expressão NÃO *p*, que se escreve em símbolos lógicos como $\sim p$ tem o valor V quando *p* é F e tem valor F quando *p* é V. Sua tabela verdade é

p	$\sim p$
V	F
F	V

Note que, a dupla negação pode ser simplificada, formando a expressão $\sim \sim p \equiv p$.

E

Também chamada CONJUNÇÃO (ou na filosofia, copulativo) de duas proposições *p* e *q* e denotado por $p \wedge q$ a proposição composta cujo valor é V quando *p* é V e *q* é V, e que vale F quando *p* ou *q* ou ambas valem F. A expressão $p \wedge q$ é lida como “*p* E *q*”.

OU

Também chamada DISJUNÇÃO de duas proposições *p* e *q* e denotado por $p \vee q$ a proposição composta cujo valor é V quando *p* é V ou *q* é V, ou ambas são V e que vale F quando *p* e *q* valem F. A expressão $p \vee q$ é lida como “*p* OU *q*”.

CONDICIONAL

Uma proposição é condicional quando se escreve “se *p* então *q*”. Representa-se esta fórmula por $p \rightarrow q$ e seu valor lógico é F apenas quando *p* é V e *q* é F, sendo verdadeira em todos os demais casos.

Na expressão $p \rightarrow q$, *p* é o antecedente e *q* o consequente. Matematicamente em $p \rightarrow q$ pode-se afirmar que

1. *p* é condição suficiente para *q* e
2. *q* é condição necessária para *p*

A condicional $p \rightarrow q$ NÃO afirma que o consequente *q* se deduz ou é consequência do antecedente *p*. Assim, por exemplo, na condicional se $3 \geq 2 \rightarrow 4 \neq 5$ não se afirma que $4 \neq 5$ se deduz ou é consequência de $3 \geq 2$. Uma coisa não precisa ter nada a ver com a outra. O que a condicional afirma é unicamente uma relação entre os valores lógicos do antecedente e do consequente de acordo com a tabela verdade.

BICONDICIONAL

Uma proposição é bicondicional quando se escreve “se *p* se e somente se *q*”. s vezes, simplifica-se esta expressão como *p* sse *q*. Representa-se esta fórmula por $p \leftrightarrow q$ e seu valor lógico é V quando *p* e *q* são ambas verdadeiras ou ambas falsas. É F nos demais casos.

Resumo

Eis a seguir um resumo de tudo o que se viu

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
V	V	V	V	V	V
V	F	F	V	F	F
F	V	F	V	V	F
F	F	F	F	V	V

Para você fazer

1. Usando duas variáveis lógicas *p* e *q*, conforme explicado no exemplo, calcule e resolva a tabela verdade para a expressão $(q \wedge p) \wedge (q \wedge p)$
2. Idem para a expressão $(p \leftrightarrow q) \wedge ((\sim q) \leftrightarrow q)$
3. Idem para a expressão $(p \rightarrow q) \wedge ((\sim q) \vee (\sim p))$
4. Idem para a expressão $(p \wedge q) \wedge (q \wedge q)$
5. Usando agora as variáveis lógicas *p*, *q* e *r* inicializadas como

p	coluna 1	VVVVVFFF
q	coluna 2	VVFFVVVF
r	coluna 3	VFVFVFVF

Calcule e resolva a tabela verdade para a expressão

$(p \vee q) \wedge ((\sim r) \wedge p)$

6. Idem para $(r \vee (\sim p)) \wedge (q \rightarrow (\sim p))$
7. Idem para $(q \wedge p) \vee (r \leftrightarrow p)$
8. Idem para $(r \wedge (\sim p)) \rightarrow ((\sim p) \wedge q)$

Responda na tabela a seguir

1	2	3	4	5	6	7	8
X	X	X	X				
X	X	X	X				
X	X	X	X				
X	X	X	X				



Prática em tabela verdade

Construção de tabelas verdade

A construção de tabelas verdade de quaisquer expressão lógica composta é a maneira de interpretar a semântica (significado) da expressão composta determinando exatamente quando ela vale V e quando vale F para todas as combinações das proposições originais.

Tal construção sempre é possível já que se sabe que o significado de uma proposição composta está sempre univocamente determinado pelo significado de cada uma de suas proposições componentes.

Uma tabela verdade sempre terá 2ⁿ linhas, onde n é a quantidade de proposições originais. A propósito na tabela acima, há duas proposições, p e q e a tabela verdade tem 4 linhas (2² = 4). Essas linhas vão conter os arranjos com repetição dos valores V e F para as proposições. Antes de prosseguir vale relembrar a sequência de valores de 2ⁿ que vem a ser:

n	1	2	3	4	5	6	7	8	9	10	n
2 ⁿ	2	4	8	16	32	64	128	256	512	1024	2 ⁿ

Exemplo

Seja construir a tabela verdade da proposição composta P:

P(p,q) = ~ (p ∧ ~ q)

Método de construção

- 1. Construa as colunas referentes às proposições originais, neste caso p e q.
- 2. Distribua valores V e F nestas colunas, seguindo a regra. Coloque 2ⁿ⁻¹ valores V e 2ⁿ⁻¹ valores F na primeira coluna. Depois 2ⁿ⁻² valores V e 2ⁿ⁻² valores F na segunda coluna e assim por diante até que na última coluna V e F se intercalem na coluna.

No exemplo acima, eis como ficariam as duas primeiras colunas

p	q
V	V
V	F
F	V
F	F

- 3. Agora, analise a expressão e vá colocando colunas à medida em que partes da expressão vão sendo resolvidas. No exemplo acima, a próxima coluna é ~ q.

p	q	~ q
V	V	F
V	F	F
F	V	V
F	F	V

- 4. Continue a criar colunas, NA ORDEM em que a expressão original vai sendo resolvida. No caso:

p	q	~ q	p ∧ ~ q	~ (p ∧ ~ q)
V	V	F	F	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	V

- 5. Agora comece a preencher a tabela, usando as entradas à esquerda (correspondendo às proposições originais) e a tabela resumo (no início desta aula). Eis como fica o exemplo

p	q	~ q	p ∧ ~ q	~ (p ∧ ~ q)
V	V	F	F	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	V

- 6. A coluna mais à direita contém a tabela verdade da proposição composta pedida.

Prioridade Na hora de resolver uma expressão composta (cabeluda) precisa-se ter uma ordem de prioridade das operações a resolver. Eis um exemplo do impasse a que se pode chegar: O que fazer diante de p ∨ q ∧ r. As possibilidades são (p ∨ q) ∧ r ou p ∨ (q ∧ r). Ambas expressões são conceitualmente diferentes. A primeira é uma conjunção e a segunda é uma disjunção (embora neste caso, ambas tenham a mesma tabela verdade, são equivalentes, se não acredita, contrua a tabela verdade).
As regras de prioridade são as seguintes:
★ Os parênteses podem alterar a prioridade normal de acordo com as regras usuais da matemática básica.
★ Bicondicional ou se e somente se (↔).
★ condicional ou implica (→)
★ E e OU, indistintamente (∧ e ∨).
★ operação negação (~).

Resumo

Eis a seguir um resumo de tudo o que se viu

p	q	p ∧ q	p ∨ q	p → q	p ↔ q
V	V	V	V	V	V
V	F	F	V	F	F
F	V	F	V	V	F
F	F	F	F	V	V

Para você fazer

- 1. Usando duas variáveis lógicas p e q, conforme explicado no exemplo, calcule e resolva a tabela verdade para a expressão (((p ↔ q) ∧ (p ∨ (~ q))) ∨ (p ∧ p))
- 2. Idem para a expressão ((q ∨ p) ∧ ((~ p) ∧ (~ p)))
- 3. Idem para a expressão (((p ↔ q) ↔ (p ↔ (~ p))) ∧ (q ∧ q))
- 4. Idem para a expressão (((((p → q) ↔ (p → (~ q))) ∨ (q ∧ (~ p))) ∨ (q ↔ q))
- 5. Usando agora as variáveis lógicas p, q e r inicializadas como

p	coluna 1	VVVVFFFF
q	coluna 2	VVFFVVFF
r	coluna 3	VFVFVFVF

Calcule e resolva a tabela verdade para a expressão (((q → p) ∨ ((~ p) → (~ q))) ↔ (r → p))

- 6. Idem para (((r ∨ p) ∨ (p ↔ q)) ∧ (r ∧ q))
- 7. Idem para (((((~ r) ∧ p) ↔ (q ∧ (~ p))) ↔ ((~ q) ∧ (~ p))) ∨ (p ↔ (~ q)))
- 8. Usando agora as variáveis lógicas p, q, r e s inicializadas como

p	coluna 1	VVVVVVVVFFFFFFFF
q	coluna 2	VVVVFFFFVVVVFFFF
r	coluna 3	VVFFVVFFVVFFVVFF
s	coluna 4	VFVFVFVFVFVFVFVF

Calcule e resolva a tabela verdade para a expressão (((((q ∨ p) ∨ (p ∧ q)) ∧ (s ∨ r)) ∨ (r ↔ (~ p)))

- 9. Idem para ((((((~ s) → r) ↔ (r → (~ p))) ∧ (p → (~ s))) ∨ (r ↔ q))
- 10. Idem para ((((((r ∨ p) ↔ (q → (~ s))) ↔ ((~ r) ∧ q)) → ((~ p) → s)) ∨ (s ∧ p))

Responda na tabela a seguir

1	2	3	4	5	6	7	8	9	10
X	X	X	X						
X	X	X	X						
X	X	X	X						
X	X	X	X						
X	X	X	X	X	X	X			
X	X	X	X	X	X	X			
X	X	X	X	X	X	X			
X	X	X	X	X	X	X			
X	X	X	X	X	X	X			
X	X	X	X	X	X	X			
X	X	X	X	X	X	X			

Prática em tabela verdade

Nesta folha você vai ter que resolver muitas tabelas verdade. O método é parecido com o Kumon e sugere fazer dezenas de T.V. até que o método de resolução fique mais ou menos automático.
Por exemplo:
 $((p \vee q) \rightarrow (p \wedge q))$
Esta tabela verdade quando resolvida dá a resposta VFFV. Para efeitos de correção, esta resposta deve ser entendida como um vetor binário, onde V vale 1 e F vale 0. Fazendo esta tradução, VFFV ficará 1001. A seguir, este número obtido deve ser entendido como um número binário e convertido para decimal. Acompanhe o começo da tabela de conversão:

0	0	0	0	0	0	0
0	0	0	0	1	0	1
0	0	0	1	0	0	2
0	0	0	1	1	1	3
0	0	1	0	0	0	4
0	0	1	0	1	1	5
0	0	1	1	0	0	6
0	0	1	1	1	1	7
0	1	0	0	0	0	8
0	1	0	0	1	1	9
...						

0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18
1	0	0	1	1	19

Fazendo a conversão do exemplo visto acima, (VFFV ou 1001) a resposta será 9.

Exemplos

$((\sim p) \vee q) \vee (q \rightarrow (\sim q))$	15
$(q \vee p) \leftrightarrow (q \wedge (\sim p))$	3
$(q \leftrightarrow p) \wedge (q \wedge (\sim q))$	0
$((q \vee p) \rightarrow (p \vee q))$	15
$(p \wedge (\sim q)) \leftrightarrow (p \wedge p)$	7

📖 Para você fazer

Enunciado	Resposta
1. $((r \rightarrow q) \vee (r \wedge p))$	
2. $((p \vee q) \vee (q \vee p))$	
3. $((q \leftrightarrow (\sim p)) \leftrightarrow ((\sim q) \wedge q))$	
4. $((((\sim r) \vee p) \vee (r \rightarrow p)) \leftrightarrow (q \wedge (\sim p)))$	
5. $((((q \vee r) \wedge (r \vee q)) \vee (p \leftrightarrow (\sim p)))$	
6. $((q \wedge (\sim p)) \wedge ((\sim p) \rightarrow p))$	
7. $((r \wedge p) \wedge (r \leftrightarrow (\sim q)))$	
8. $((((\sim r) \wedge q) \vee ((\sim p) \leftrightarrow q))$	
9. $(((((\sim q) \leftrightarrow (\sim p)) \wedge (q \wedge (\sim p))) \leftrightarrow (q \vee p))$	
10. $((p \wedge r) \wedge (q \vee q))$	
11. $(((((\sim p) \rightarrow q) \wedge (p \wedge q)) \wedge (p \rightarrow (\sim p)))$	
12. $((p \rightarrow q) \vee (q \wedge p))$	
13. $((p \leftrightarrow (\sim q)) \vee (q \leftrightarrow r))$	
14. $((q \rightarrow p) \leftrightarrow ((\sim r) \wedge r))$	
15. $((p \wedge q) \wedge (q \leftrightarrow p)) \leftrightarrow (p \vee q)$	
16. $(((((\sim r) \wedge q) \vee (r \wedge p)) \vee ((\sim r) \leftrightarrow p))$	
17. $((p \vee q) \vee ((\sim r) \rightarrow r))$	
18. $((p \vee r) \vee (r \vee q))$	
19. $((p \rightarrow q) \leftrightarrow (r \vee q)) \wedge (q \wedge r)$	
20. $((\sim p) \rightarrow q) \wedge (p \leftrightarrow (\sim q))$	

Eis um resumo de tudo o que se viu

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
V	V	V	V	V	V
V	F	F	V	F	F
F	V	F	V	V	F
F	F	F	F	V	V

Semântica das proposições
tautologia

Uma tautologia é qualquer proposição cuja última coluna na tabela verdade seja composta apenas por verdadeiro (V). Em outras palavras, não importa o que afirmem suas proposições componentes, a tautologia sempre é verdadeira. São também conhecidas como proposições tautológicas ou proposições logicamente verdadeiras.

Eis alguns exemplos:

$p \rightarrow p$	princípio da identidade
$p \leftrightarrow p$	princípio da identidade
$\sim (p \wedge \sim p)$	princípio da não contradição
$p \vee \sim p$	terceiro excluído

Contradição

É o contrário da tautologia, ou seja qualquer proposição cuja última coluna na tabela verdade contenha apenas falsidades (F). Em outras palavras, não importa qual o valor verdade das proposições que componham uma contradição, o valor verdade final é sempre falso.

Note que sempre se pode afirmar que a negação de uma contradição é uma tautologia e a negação de uma tautologia é sempre uma contradição. As contradições são também conhecidas como proposições contraválidas ou proposições logicamente falsas.

Eis alguns exemplos

$p \wedge \sim p$	p não pode V e F ao mesmo tempo
$p \leftrightarrow \sim p$	

Contingência

Chama-se contingência a toda proposição composta em cuja última coluna da sua tabela verdade figuram V e F pelo menos uma vez cada. Em outras palavras, uma contingência é uma proposição que não é nem tautologia nem contradição. Também recebem o nome de proposições contingentes ou proposições indeterminadas.

Eis alguns exemplos

$p \rightarrow \sim p$	

Implicação lógica

Diz-se que P implica logicamente Q se Q é verdadeira todas as vezes que P o é. Note que tanto P quanto Q são (ou podem ser) proposições compostas.

Em outros termos, P implica Q se na mesma linha de suas correspondentes tabela verdade não ocorre V (em P) e F (em Q). Escreve-se $P \Rightarrow Q$.

Esta relação (a implicação) goza das propriedades reflexiva e transitiva. Ou seja $P \Rightarrow P$ (reflexiva) e se $P \Rightarrow Q$ e $Q \Rightarrow R$ então $P \Rightarrow R$ (transitiva).

Veja-se alguma manipulação destes fatos: vai-se construir as tabelas-verdade de $p \wedge q$, $p \vee q$ e $p \leftrightarrow q$ e fica

p	q	$p \wedge q$	$p \vee q$	$p \leftrightarrow q$
V	V	V	V	V
V	F	F	V	F
F	V	F	V	F
F	F	F	F	V

Da análise da tabela acima, pode-se afirmar que $p \wedge q \Rightarrow p \vee q$ e $p \wedge q \Rightarrow p \leftrightarrow q$. A mesma tabela verdade ilustra as importantes regras de inferência:

- $p \Rightarrow p \vee q$ e $q \Rightarrow p \vee q$ (regra da adição)
- $p \wedge q \Rightarrow p$ e $p \wedge q \Rightarrow q$ (regra da simplificação)

Sejam agora as tabelas verdade das proposições $p \leftrightarrow q$, $p \rightarrow q$, e $q \rightarrow p$, que são:

p	q	$p \leftrightarrow q$	$p \rightarrow q$	$q \rightarrow p$
V	V	V	V	V
V	F	F	F	V
F	V	F	V	F
F	F	V	V	V

A proposição $p \leftrightarrow q$ é verdadeira nas linhas 1 e 4 e nestas linhas as proposições $p \rightarrow q$ e $q \rightarrow p$ também são. Logo a primeira proposição implica cada uma das outras, ou: $p \leftrightarrow q \Rightarrow p \rightarrow q$ e $p \leftrightarrow q \Rightarrow q \rightarrow p$.

Seja agora a tabela verdade da proposição $(p \vee q) \wedge \sim p$, que é:

p	q	$p \vee q$	$\sim p$	$(p \vee q) \wedge \sim p$
V	V	V	F	F
V	F	V	F	F
F	V	V	V	V
F	F	F	V	F

Esta proposição é V somente na linha 3 e aqui q também é verdadeira. Logo subsiste a implicação lógica $(p \vee q) \wedge \sim p \Rightarrow q$. Esta regra é denominada Silogismo Disjuntivo. Outra forma desta mesma regra é $(p \vee q) \wedge \sim q \Rightarrow p$.

Mais uma tabela verdade, seja agora a proposição $(p \rightarrow q) \wedge p$

p	q	$p \rightarrow q$	$(p \rightarrow q) \wedge p$
V	V	V	V
V	F	F	F
F	V	V	F
F	F	V	F

Esta proposição só é verdadeira na linha 1 e nesta linha q também é verdadeira. Logo subsiste a implicação lógica $(p \rightarrow q) \wedge p \Rightarrow q$, denominada regra Modus Ponens. Seja este exemplo: Se chove $\rightarrow$ a calçada fica molhada. Se chove ... então a calçada fica molhada.

Sejam agora as tabelas verdade das proposições $(p \rightarrow q) \wedge \sim q$ e $\sim p$ e tem-se

p	q	$p \rightarrow q$	$\sim q$	$(p \rightarrow q) \wedge \sim q$	$\sim p$
V	V	V	F	F	F
V	F	F	V	F	F
F	V	V	F	F	V
F	F	V	V	V	V

A proposição $(p \rightarrow q) \wedge \sim q$ é verdadeira só na linha 4 e nesta a proposição $\sim p$ também o é. Logo subsiste a implicação lógica: $(p \rightarrow q) \wedge \sim q \Rightarrow \sim p$, cujo nome em latim é regra Modus tollens. Para exemplificar: Se chove $\rightarrow$ a calçada fica molhada. Se a calçada não está molhada ... então não chove. MUITO IMPORTANTE: Cuidado, porque ~~se não é possível misturar estas duas regras, a saber Se chove $\rightarrow$ a calçada fica molhada. Se não chove... então a calçada não fica molhada.~~ A afirmação anterior não é verdade pois pode ter acontecido: Se brincamos com água... então a calçada fica molhada.

As mesmas tabelas verdade mostram que $\sim p$ implica $p \rightarrow q$ ou seja, $\sim p \Rightarrow p \rightarrow q$. A propósito os símbolos $\rightarrow$ e $\Rightarrow$ são distintos, pois $\rightarrow$ é uma operação lógica (por exemplo quando aplicado às proposições p e q dá a nova proposição $p \rightarrow q$, enquanto que $\Rightarrow$ é uma relação. Uma regra importante é que se $P \Rightarrow Q$ isso significa que $P \rightarrow Q$ é uma tautologia.

De Morgan

Nos exercícios a seguir vai ser necessário empregar as leis de De Morgan. Elas descrevem o que fazer quando se quer negar uma proposição composta formada por $\wedge$ ou por $\vee$. As expressões que descrevem essas leis são:

$\sim (X \wedge Y)$	equivale a	$(\sim X) \vee (\sim Y)$
$\sim (X \vee Y)$	equivale a	$(\sim X) \wedge (\sim Y)$

📖 Para você fazer

Exercício 1 Nas expressões a seguir, identifique cada uma com uma das letras: **T** para uma tautologia, **C** para uma contradição e **X** para uma contingência.

- $((p \vee (\sim q)) \rightarrow (q \rightarrow (\sim p))) \vee (q \leftrightarrow q))$
- $((p \leftrightarrow (\sim q)) \rightarrow ((\sim r) \vee q)) \vee (p \leftrightarrow r))$
- $((((\sim p) \leftrightarrow (\sim r)) \wedge (q \leftrightarrow (\sim r))) \wedge ((\sim r) \wedge r))$
- $((q \vee (\sim r)) \wedge (p \vee q))$

- $((r \wedge q) \wedge ((\sim r) \vee (\sim r))) \wedge ((\sim r) \wedge p))$
- $((p \wedge q) \vee (q \vee (\sim q)))$
- $((p \vee (\sim q)) \wedge ((\sim q) \rightarrow q)) \rightarrow (p \rightarrow q))$
- $((p \vee q) \rightarrow (p \vee (\sim p))) \vee ((\sim q) \wedge (\sim p))$
- $((q \vee p) \leftrightarrow ((\sim p) \vee q))$
- $(((((\sim q) \rightarrow (\sim p)) \leftrightarrow (q \wedge (\sim r))) \vee ((\sim q) \vee r)) \leftrightarrow (q \wedge (\sim q)))$

Exercício 2 Nas frases abaixo, identifique as proposições devidamente enunciadas em português e verifique se elas caracterizam um argumento modus ponens, modus tollens, ou nada disso. Responda **P**, **T** ou **X** em cada caso.

- (se o livro é grande e é agradável então a flor é bonita e cheira mal) e o livro é grande e é agradável. Logo a flor é bonita e cheira mal
- (se a picanha é deliciosa e suculenta então o pinguim é simpático ou é bizarro) e a picanha é deliciosa e suculenta. Logo o pinguim é simpático ou é bizarro
- (se Binho é um gato maravilhoso então a rosa é cheirosa) e Binho não é um gato maravilhoso. Logo a rosa não é cheirosa
- (se o pinguim é simpático ou é bizarro então a tesoura é grande e afiada) e o pinguim é simpático ou é bizarro. Logo a tesoura é grande e afiada
- (se o dinheiro é farto então o celular é leve) e o dinheiro é farto. Logo o celular é leve
- (se antônio e lucinda são namorados então o pinguim é simpático ou é bizarro) e o pinguim é antipático e não é bizarro. Logo antonio e lucinda não são namorados
- (se o dinheiro é farto então Binho é um gato maravilhoso) e Binho não é um gato maravilhoso. Logo o dinheiro não é farto
- (se o celular é leve então rex é um cachorro babão) e o celular é leve. Logo rex é um cachorro babão
- (se faz calor então o óculos serve para enxergar) e faz calor. Logo o óculos serve para enxergar
- (se o óculos serve para enxergar então a conversa fluiu fácil e foi rápida) e o óculos serve para enxergar. Logo a conversa fluiu fácil e foi rápida

Escreva aqui as respostas

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20



Equivalência Lógica

Diz-se que P e Q são logicamente equivalente (ou apenas equivalentes) se as tabelas verdade destas duas proposições são idênticas. Escreve-se neste caso $P \Leftrightarrow Q$. Em particular se P e Q são ambas tautologias, ou são ambas contradições então elas são equivalentes.

Não se deve confundir $P \Leftrightarrow Q$ com $P \Leftrightarrow Q$. No primeiro caso tem-se uma operação lógica que dá um resultado de acordo com aulas passadas. (Uma boa sugestão aqui é relembrar o resultado da operação $\leftrightarrow$). O segundo é uma relação que pode ou não ser verdadeira.

A relação da equivalência lógica ($\Leftrightarrow$) entre proposições goza das 3 propriedades: reflexiva, simétrica e transitiva. Acompanhe:

- reflexiva** P é sempre equivalente a P , seja P quem for.
- simétrica** Se P é equivalente a Q , então Q é sempre equivalente a P .
- transitiva** Se P é equivalente a Q , e Q é equivalente a R , então P é equivalente a R .

Redução ao absurdo

A bicondicional $(p \wedge \sim q \rightarrow c) \leftrightarrow (p \rightarrow q)$, onde c é uma proposição lógica cujo valor lógico é sempre F, é uma tautologia pois a última coluna de sua tabela verdade é sempre V.

p	q	$(p \wedge \sim q \rightarrow c)$	$(p \rightarrow q)$
V	V	F	V
V	F	V	F
F	V	F	V
F	F	F	V

Esta proposição recebe o nome de demonstração por absurdo, e ela é muito importante na matemática e na inteligência artificial, sendo a base do método de demonstração automática de teoremas.

Negação conjunta

A negação conjunta de duas proposições p e q é a proposição $\sim p \wedge \sim q$, que é lida como **não p e não q**. Usa-se para esta representação o símbolo $\downarrow$, e escreve-se $p \downarrow q$. O símbolo $\downarrow$ junto com o $\uparrow$ são conhecidos como conectivos de Scheffer. Neste caso, tem-se

$$p \downarrow q \Leftrightarrow \sim p \wedge \sim q$$

A tabela verdade de $p \downarrow q$ é a seguinte

p	q	$(\sim p \wedge \sim q)$
V	V	F
V	F	F
F	V	F
F	F	V

Negação Disjunta

A negação disjunta de duas proposições p e q é a proposição **não p ou q**, ou simbolicamente $\sim p \vee \sim q$. A negação disjunta também é representada pela notação $p \uparrow q$, ou

$$p \uparrow q \Leftrightarrow \sim p \vee \sim q$$

e tem a seguinte tabela verdade

p	q	$(\sim p \vee \sim q)$
V	V	F
V	F	V
F	V	V
F	F	V

👉 Para você fazer

Exercício 1 Nos exercícios a seguir, são dadas 2 expressões lógicas. Responda **S** se elas forem equivalentes e **N** se não forem.

1. $((p \wedge (\sim q)) \wedge (q \wedge q)) \vee (p \wedge p)$ E $((q \rightarrow p) \vee (p \wedge (\sim q))) \vee (q \vee (\sim q))$

2. $((p \vee q) \wedge (p \wedge (\sim q))) \vee (p \rightarrow q)$ E $((p \wedge (\sim q)) \vee (q \vee (\sim p))) \vee (p \rightarrow (\sim p))$

3. $((p \vee q) \leftrightarrow ((\sim q) \wedge p)) \vee (q \wedge (\sim p))$ E $((p \wedge (\sim q)) \rightarrow ((\sim q) \leftrightarrow (\sim p))) \leftrightarrow ((\sim p) \leftrightarrow (\sim p))$

4. $((q \leftrightarrow (\sim p)) \leftrightarrow (q \leftrightarrow (\sim p))) \wedge (p \wedge (\sim p))$ E $((q \rightarrow (\sim p)) \wedge (p \wedge q)) \wedge (q \vee q)$

5. $((q \vee p) \vee (p \wedge q)) \vee (p \leftrightarrow q)$ E $((q \wedge p) \vee (q \vee (\sim q))) \vee (q \wedge q)$

6. $((p \leftrightarrow (\sim q)) \leftrightarrow ((\sim q) \leftrightarrow (\sim p))) \wedge (p \vee q)$ E $((p \wedge q) \wedge (p \wedge q)) \wedge (q \leftrightarrow p)$

Exercício 2 Nos exercícios a seguir são dadas as expressões a e b . Responda **S** se for possível concluir que $a \rightarrow b$ por redução ao absurdo e **N** se não for possível. Use sempre as proposições parciais $p, q, r, s, \dots$ nessa ordem.

7. $a = ((q \vee p) \vee ((\sim p) \wedge q))$
 $b = ((q \rightarrow (\sim p)) \wedge (q \wedge q))$

8. $a = ((p \rightarrow q) \vee (p \wedge (\sim q)))$
 $b = ((q \vee p) \vee ((\sim q) \wedge (\sim p)))$

9. $a = (((\sim q) \leftrightarrow (\sim p)) \vee (q \wedge p))$
 $b = ((q \wedge p) \vee (q \leftrightarrow (\sim q)))$

10. $a = (((\sim q) \rightarrow p) \vee (q \wedge (\sim q)))$
 $b = ((q \rightarrow (\sim p)) \wedge ((\sim p) \wedge (\sim p)))$

11. $a = (((q \wedge p) \vee (p \vee p)) \rightarrow (p \vee q))$
 $b = (((\sim q) \wedge p) \vee (p \rightarrow q)) \vee (q \wedge (\sim p))$

12. $a = (((p \vee q) \wedge (q \wedge q)) \leftrightarrow (p \rightarrow q))$
 $b = (((p \leftrightarrow q) \rightarrow ((\sim q) \leftrightarrow (\sim p))) \vee ((\sim q) \vee p))$

Exercício 3 Nas frases abaixo, identifique se a resposta dada está certa (respondendo **S**) ou está errada (respondendo **N**) nas negações de conjunção ($\downarrow$) ou nas negações de disjunção ($\uparrow$). Use sempre as proposições parciais $p, q, r, s, \dots$ nessa ordem.

13. $a \uparrow b$. $a = (((\sim p) \leftrightarrow r) \wedge ((\sim p) \wedge (\sim q))) \wedge (r \vee p)$ e $b = (((q \wedge r) \leftrightarrow (q \wedge p)) \leftrightarrow (q \leftrightarrow r))$ deu resultado = FVVVVVVV
14. $a \downarrow b$. $a = ((q \vee p) \rightarrow (p \wedge (\sim p)))$ e $b = ((q \leftrightarrow p) \leftrightarrow (q \leftrightarrow (\sim q)))$ deu resultado = VFFF
15. $a \uparrow b$. $a = ((q \leftrightarrow p) \wedge (p \vee (\sim q)))$ e $b = ((p \vee q) \leftrightarrow (q \rightarrow p))$ deu resultado = VVFFV
16. $a \downarrow b$. $a = ((p \vee (\sim q)) \rightarrow (q \vee p))$ e $b = ((q \rightarrow (\sim p)) \rightarrow (p \rightarrow (\sim p)))$ deu resultado = VFFF
17. $a \downarrow b$. $a = (((p \rightarrow r) \wedge (p \rightarrow (\sim r))) \vee (q \wedge (\sim p)))$ e $b = (((r \vee q) \wedge (p \vee (\sim r))) \wedge (q \rightarrow r))$ deu resultado = FVFFVFFFF
18. $a \uparrow b$. $a = ((q \leftrightarrow (\sim p)) \vee (p \leftrightarrow q))$ e $b = ((p \vee (\sim q)) \wedge (p \rightarrow q))$ deu resultado = FVFF
19. $a \uparrow b$. $a = (((\sim q) \vee p) \wedge (p \vee p))$ e $b = ((q \leftrightarrow p) \wedge ((\sim q) \leftrightarrow (\sim p)))$ deu resultado = VVVFF
20. $a \uparrow b$. $a = ((q \leftrightarrow p) \wedge (q \vee q))$ e $b = ((p \wedge (\sim q)) \leftrightarrow (q \rightarrow p))$ deu resultado = FVVV

Escreva aqui

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20

Álgebra das proposições - 1. parte

Agora a disciplina de lógica começa a ficar mais emocionante e cheia de drama e paixão. Neste momento, começa-se a provar coisas, construindo o enorme edifício da lógica matemática.

A seguir, algumas propriedades algébricas das proposições, conjunções ($\wedge$) e disjunções ($\vee$).

Conjunção

Começando com a operação $\wedge$.

Idempotente: A conjunção é idempotente. Trocando em miúdos, $p \wedge p \Leftrightarrow p$. Uma maneira de provar isto é estudar a proposição $p \wedge p \leftrightarrow p$. Esta segunda proposição deve ser uma tautologia (Se for, está provada a equivalência acima). Analisando a tabela verdade

p	$p \wedge p$	$p \wedge p \leftrightarrow p$
V	V	V
F	F	V

Comutativa: A conjunção é comutativa, a saber, $p \wedge q \Leftrightarrow q \wedge p$. Para demonstrar essa afirmação deve-se notar que as tabelas verdade de $p \wedge q$ e de $q \wedge p$ são idênticas, ou seja a bicondicional $p \wedge q \leftrightarrow q \wedge p$ é uma tautologia.

p	q	$p \wedge q$	$q \wedge p$	$p \wedge q \leftrightarrow q \wedge p$
V	V	V	V	V
V	F	F	F	V
F	V	F	F	V
F	F	F	F	V

Associativa: A conjunção é associativa, como em $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$. Fica como exercício ao leitor provar esta afirmação: Para fazer isso, ache as tabelas verdade de $(p \wedge q) \wedge r$ e de $p \wedge (q \wedge r)$. Ambas devem ser iguais. Daqui, afirma-se que $(p \wedge q) \wedge r \leftrightarrow p \wedge (q \wedge r)$ é uma tautologia. Finalmente, a tautologia prova a proposição onde $\leftrightarrow$ foi trocada por $\Leftrightarrow$.

Identidade: A conjunção de uma proposição p com a tautologia é igual a p e a conjunção de p com uma contradição é uma contradição. Em símbolos: $p \wedge t \Leftrightarrow p$ e $p \wedge c \Leftrightarrow c$, onde t é uma tautologia (tudo V) e c é uma contradição (tudo F). Destes fatos, pode-se afirmar que, em relação à conjunção, a tautologia é o elemento neutro, e a contradição é o elemento absorvente.

Disjunção

Seguindo com a operação $\vee$.

Idempotente: A disjunção é idempotente. Trocando em miúdos, $p \vee p \Leftrightarrow p$. Uma maneira de provar isto é estudar a proposição $p \vee p \leftrightarrow p$. Esta segunda proposição deve ser uma tautologia (Se for, está provada a equivalência acima). Analisando a tabela verdade

p	$p \vee p$	$p \vee p \leftrightarrow p$
V	V	V
F	F	V

Comutativa: A disjunção é comutativa, a saber, $p \vee q \Leftrightarrow q \vee p$. Para demonstrar essa afirmação deve-se notar que as tabelas verdade de $p \vee q$ e de $q \vee p$ são idênticas, ou seja a bicondicional $p \vee q \leftrightarrow q \vee p$ é uma tautologia.

p	q	$p \vee q$	$q \vee p$	$p \vee q \leftrightarrow q \vee p$
V	V	V	V	V
V	F	V	V	V
F	V	V	V	V
F	F	F	F	V

Associativa: A disjunção é associativa, como em $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$. Fica como exercício ao leitor provar esta afirmação: Para fazer isso, ache as tabelas verdade de $(p \vee q) \vee r$ e de $p \vee (q \vee r)$. Ambas devem ser iguais. Daqui, afirma-se que $(p \vee q) \vee r \leftrightarrow p \vee (q \vee r)$ é uma tautologia. Finalmente, a tautologia prova a proposição onde $\leftrightarrow$ foi trocada por $\Leftrightarrow$.

Identidade: A disjunção de uma proposição p com a tautologia é igual a t e a disjunção de p com uma contradição é p . Em símbolos: $p \vee t \Leftrightarrow t$ e $p \vee c \Leftrightarrow p$, onde t é uma tautologia (tudo V) e c é uma contradição (tudo F). Destes fatos, pode-se afirmar que, em relação à disjunção, a tautologia é o elemento absorvente, e a contradição é o elemento neutro.

Conjunção e disjunção

Dadas as proposições p , q e r quaisquer, pode-se afirmar:

propriedade	expressão
distributiva	$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
idem	$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
absorção	$p \wedge (p \vee q) \Leftrightarrow p$
idem	$p \vee (p \wedge q) \Leftrightarrow p$
De Morgan	$\sim (p \wedge q) \Leftrightarrow \sim p \vee \sim q$
idem	$\sim (p \vee q) \Leftrightarrow \sim p \wedge \sim q$
negação da condicional	$\sim (p \rightarrow q) \Leftrightarrow p \wedge \sim q$
negação da bicondicional	$\sim (p \leftrightarrow q) \Leftrightarrow (p \wedge \sim q) \vee (\sim p \wedge q)$

Note-se que graças as regras de De Morgan, é possível definir a disjunção a partir da conjunção + negação. Idem definir a conjunção a partir da disjunção + negação. Em símbolos: $p \vee q \Leftrightarrow \sim (\sim p \wedge \sim q)$ e $p \wedge q \Leftrightarrow \sim (\sim p \vee \sim q)$.

Método Dedutivo

O método de trabalho usado até agora é o da montagem das tabelas verdade. Fácil, porém trabalhoso, já que as tabelas verdade crescem exponencialmente em linhas. Relembrando a existência de n proposições simples, exige 2^n linhas nas tabelas verdade.

Trata-se de um método conhecido em ciência da computação como **força bruta**. O nome vem do fato de que para resolver o problema, realizam-se todas as combinações possíveis de valores V e F. Na prática, o que se quer obter ao final dos cálculos é o valor de uma determinada coluna e uma determinada linha. Aqui fica bem, claro como há desperdício de energia no método das T.V.

A alternativa é o método dedutivo, que ao contrário da TV se concentra naquilo que se quer provar e através de manipulações lógicas perfeitamente válidas, chega-se ao que se quer provar.

Antes algumas propriedades

$p \wedge p \Leftrightarrow p$	idempotência da conjunção
$p \wedge q \Leftrightarrow q \wedge p$	comutativa da conjunção
$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	associativa da conjunção
$p \wedge t \Leftrightarrow p$	identidade da conjunção
$p \wedge c \Leftrightarrow c$	identidade da conjunção
$p \vee p \Leftrightarrow p$	idempotência da disjunção
$p \vee q \Leftrightarrow q \vee p$	comutativa da disjunção
$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$	associativa da disjunção
$p \vee t \Leftrightarrow t$	identidade da disjunção
$p \vee c \Leftrightarrow p$	identidade da disjunção
$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	distributiva
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$	distributiva
$p \wedge (p \vee q) \Leftrightarrow p$	absorção
$p \vee (p \wedge q) \Leftrightarrow p$	absorção
$\sim (p \wedge q) \Leftrightarrow \sim p \vee \sim q$	De Morgan
$\sim (p \vee q) \Leftrightarrow \sim p \wedge \sim q$	De Morgan
$p \rightarrow q \Leftrightarrow \sim p \vee q$	substituição fundamental

Estratégias de demonstração

Quando a frase a provar é uma implicação ($a \Rightarrow b$) a estratégia é provar que a proposição $a \rightarrow b$ é uma tautologia. Quando a frase é uma equivalência, deve-se sair de um lado e chegar no outro, tanto faz em que sentido. Note que todas as propriedades que são equivalências permitem essa leitura nos dois sentidos. De resto, o objetivo é aplicar todas as propriedades acima listadas e tudo o que já se conhece da lógica matemática para provar o que se pretende. Não esqueça que muitas vezes NÃO se consegue chegar a nenhum resultado. Isso necessariamente não é um erro. Pode ser que o que se quer provar não possa ser provado.

Eis alguns exemplos:

1. Demonstrar $p \wedge q \Rightarrow p$, regra conhecida como simplificação.

- (a) $p \wedge q \rightarrow p$:- tautologia a provar. Trocando $a \rightarrow b$ por $\sim a \vee b$
- (b) $\sim (p \wedge q) \vee p$:- Aplicando De Morgan

- (c) $(\sim p \vee \sim q) \vee p$:- propriedade associativa da disjunção
- (d) $(\sim p \vee p) \vee \sim q$:- $a \vee \sim a$ é tautologia
- (e) $T \vee \sim q \Leftrightarrow T$ - pela identidade da disjunção. CQD.

2. Provar $c \Rightarrow p$ onde p é uma proposição qualquer e c é uma contradição (tudo F). Para provar esta implicação, deve-se provar que a proposição $c \rightarrow p$ é uma tautologia.

- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____

3. Provar $p \Rightarrow t$. Como acima, deve-se provar que a proposição $p \rightarrow t$ é uma tautologia.

- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____

4. Demonstrar $p \Rightarrow p \vee q$, conhecida como adição

- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____

5. Demonstrar a regra conhecida como modus ponens. Em símbolos $(p \rightarrow q) \wedge p \Rightarrow q$

- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____
- (f) _____ :- _____
- (g) _____ :- _____

Para você fazer

Demonstre as seguintes frases lógicas, indicando passo a passo qual foi a regra, propriedade ou equivalência utilizada:

1. $p \rightarrow q \Rightarrow p \wedge r \rightarrow q$. (17)
- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____
- (f) _____ :- _____
- (g) _____ :- _____
- (h) _____ :- _____

2. $p \rightarrow q \Leftrightarrow p \vee q \rightarrow q$. (27)
- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____
- (f) _____ :- _____
- (g) _____ :- _____
- (h) _____ :- _____

Redução no número de conectivos

A seguir algumas demonstrações de como se pode reduzir o número de conectivos nas proposições:

$\wedge, \rightarrow$ e $\leftrightarrow$ exprimem-se em termos de $\sim$ e de $\vee$	
$p \wedge q$	$\Leftrightarrow \sim (\sim p \vee \sim q)$
$p \rightarrow q$	$\Leftrightarrow \sim p \vee q$
$p \leftrightarrow q$	$\Leftrightarrow \sim (\sim (\sim p \vee q) \vee \sim (\sim q \vee p))$
$\vee, \rightarrow$ e $\leftrightarrow$ exprimem-se em termos de $\sim$ e de $\wedge$	
$p \vee q$	$\Leftrightarrow \sim (\sim p \wedge \sim q)$
$p \rightarrow q$	$\Leftrightarrow \sim (p \wedge \sim q)$
$p \leftrightarrow q$	$\Leftrightarrow \sim (p \wedge \sim q) \wedge \sim (\sim p \wedge q)$
$\wedge, \vee$ e $\leftrightarrow$ exprimem-se em termos de $\sim$ e $\rightarrow$	
$p \wedge q$	$\Leftrightarrow \sim (p \rightarrow \sim q)$
$p \vee q$	$\Leftrightarrow \sim p \rightarrow q$
$p \leftrightarrow q$	$\Leftrightarrow \sim ((p \rightarrow \sim q) \rightarrow \sim (q \rightarrow p))$

Forma normal das proposições

Uma proposição está na forma normal se contém apenas $\sim, \wedge$ e $\vee$.

Para botar uma proposição na forma normal, substitui-se $a \rightarrow b$ por $\sim a \vee b$ e substitui-se $a \leftrightarrow b$ por $(\sim a \vee b) \wedge (a \vee \sim b)$.

Há duas formas normais, a conjuntiva (FNC) e a disjuntiva (FND). A FNC obedece às seguintes condições:

1. Contém apenas $\sim, \wedge$ e $\vee$.
2. $\sim$ só atua em letras proposicionais (ou seja não existe $\sim\sim$ e nem $\sim(\dots)$.
3. $\vee$ não atua sobre $\wedge$ (ou seja não há $a \vee (b \wedge \dots)$.
- A FND, obedece:
1. Contém apenas $\sim, \wedge$ e $\vee$.
2. $\sim$ só atua em letras proposicionais (ou seja não existe $\sim\sim$ e nem $\sim(\dots)$.
3. $\wedge$ não atua sobre $\vee$ (ou seja não há $a \wedge (b \vee \dots)$.

Princípio da dualidade

Se P só contém $\sim, \wedge$ e $\vee$, e se trocarmos em P cada $\vee$ por $\wedge$ e cada $\wedge$ por $\vee$ (um pelo outro), obtem-se a proposição dual de P , chamada P_1 . E daqui, surge o princípio da dualidade que diz: Se P e Q são equivalentes e se só contém $\sim, \wedge$ e $\vee$, então, suas duais P_1 e Q_1 são também equivalentes.

Proposições associadas a uma condicional

Dada uma condicional $p \rightarrow q$, definem-se 3 proposições associadas,

recíproca $q \rightarrow p$

contrária $\sim p \rightarrow \sim q$

contrapositiva $\sim q \rightarrow \sim p$

Suas tabelas verdade são:

p	q	$p \rightarrow q$	$q \rightarrow p$	$\sim p \rightarrow \sim q$	$\sim q \rightarrow \sim p$
V	V	V	V	V	V
V	F	F	V	V	F
F	V	V	F	F	V
F	F	V	V	V	V

As colunas 3-6 mostram que $p \rightarrow q$ e sua contrapositiva $\sim q \rightarrow \sim p$ são equivalentes, isto é $p \rightarrow q \Leftrightarrow \sim q \rightarrow \sim p$.

As colunas 4-5 mostram que a recíproca $q \rightarrow p$ e a contrária $\sim p \rightarrow \sim q$ da condicional $p \rightarrow q$ são equivalentes, ou seja, $q \rightarrow p \Leftrightarrow \sim p \rightarrow \sim q$.

As mesmas tabelas verdade também demonstram que as condicionais $p \rightarrow q$ e a sua recíproca $q \rightarrow p$ e a sua contrária $\sim p \rightarrow \sim q$ não são equivalentes.

Veja-se um exemplo: Se p significa que o triângulo T é equilátero, e q significa que T é isósceles, tem-se $p \rightarrow q$: Se T é equilátero, então T é isósceles, o que é verdadeiro. Já a recíproca $q \rightarrow p$ Se T é isósceles então T é equilátero, o que é falso.

Exemplo Seja determinar: a) a contrapositiva da contrapositiva de $p \rightarrow q$; b) a contrapositiva da recíproca de $p \rightarrow q$; c) a contrapositiva da contrária de $p \rightarrow q$.

A contrapositiva de $p \rightarrow q$ é $\sim q \rightarrow \sim p$. E a contrapositiva de $\sim q \rightarrow \sim p$ é $\sim \sim p \rightarrow \sim \sim q$ que é equivalente a $p \rightarrow q$ ($\sim \sim p \rightarrow \sim \sim q \Leftrightarrow p \rightarrow q$). A recíproca de $p \rightarrow q$ é $q \rightarrow p$. E a contrapositiva de $q \rightarrow p$ é $\sim p \rightarrow \sim q$. A contrária de $p \rightarrow q$ é $\sim p \rightarrow \sim q$. E a contrapositiva de $\sim p \rightarrow \sim q$ é $\sim \sim q \rightarrow \sim \sim p$ que é equivalente a $q \rightarrow p$. ($\sim \sim q \rightarrow \sim \sim p \Leftrightarrow q \rightarrow p$).

Observe que a recíproca e a contrária são cada uma a contrapositiva da outra e que a condicional e a contrapositiva são cada uma a contrapositiva da outra.

Álgebra das proposições - 2.parte

Agora a disciplina de lógica começa a ficar mais emocionante e cheia de drama e paixão. Neste momento, começa-se a provar coisas, construindo o enorme edifício da lógica matemática.

Antes algumas propriedades

$p \wedge p \Leftrightarrow p$	idempotência da conjunção
$p \wedge q \Leftrightarrow q \wedge p$	comutativa da conjunção
$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	associativa da conjunção
$p \wedge t \Leftrightarrow p$	identidade da conjunção
$p \wedge c \Leftrightarrow c$	identidade da conjunção
$p \vee p \Leftrightarrow p$	idempotência da disjunção
$p \vee q \Leftrightarrow q \vee p$	comutativa da disjunção
$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$	associativa da disjunção
$p \vee t \Leftrightarrow t$	identidade da disjunção
$p \vee c \Leftrightarrow p$	identidade da disjunção
$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	distributiva
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$	distributiva
$p \wedge (p \vee q) \Leftrightarrow p$	absorção
$p \vee (p \wedge q) \Leftrightarrow p$	absorção
$\sim (p \wedge q) \Leftrightarrow \sim p \vee \sim q$	De Morgan
$\sim (p \vee q) \Leftrightarrow \sim p \wedge \sim q$	De Morgan
$p \rightarrow q \Leftrightarrow \sim p \vee q$	substituição fundamental

Estratégias de demonstração

Quando a frase a provar é uma implicação ($a \Rightarrow b$) a estratégia é provar que a proposição $a \rightarrow b$ é uma tautologia. Quando a frase é uma equivalência, deve-se sair de um lado e chegar no outro, tanto faz em que sentido. Note que todas as propriedades que são equivalências permitem essa leitura nos dois sentidos. De resto, o objetivo é aplicar todas as propriedades acima listadas e tudo o que já se conhece da lógica matemática para provar o que se pretende. Não esqueça que muitas vezes NÃO se consegue chegar a nenhum resultado. Isso necessariamente não é um erro. Pode ser que o que se quer provar não possa ser provado.

Eis alguns exemplos:

1. Demonstrar a implicação $(p \rightarrow q) \wedge \sim q \Rightarrow \sim p$, conhecida como modus tollens.
- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____
- (f) _____ :- _____
- (g) _____ :- _____
2. Demonstrar a implicação $(p \vee q) \wedge \sim p \Rightarrow q$, conhecida como silogismo disjuntivo.
- (a) _____ :- _____

- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____

3. Demonstre a implicação $p \wedge q \Rightarrow p \vee q$. Vais provar que a proposição $p \wedge q \rightarrow p \vee q$ é uma tautologia.
- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____

4. Demonstrar a equivalência $p \rightarrow q \Leftrightarrow p \wedge \sim q \rightarrow c$, também conhecida como redução ao absurdo. Neste caso, a estratégia de demonstração implica em sair do segundo membro e mediante operações lógicas válidas, chegar no primeiro.
- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____
- (f) _____ :- _____
- (g) _____ :- _____

Para você fazer

Demonstre as seguintes frases lógicas, indicando passo a passo qual foi a regra, propriedade ou equivalência utilizada:

1. $(p \rightarrow r) \wedge (q \rightarrow r) \Leftrightarrow p \vee q \rightarrow r$. (61)
- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____
- (f) _____ :- _____
- (g) _____ :- _____
- (h) _____ :- _____
2. $(p \rightarrow r) \vee (q \rightarrow s) \Leftrightarrow p \wedge q \rightarrow r \vee s$.(7)
- (a) _____ :- _____
- (b) _____ :- _____
- (c) _____ :- _____
- (d) _____ :- _____
- (e) _____ :- _____
- (f) _____ :- _____
- (g) _____ :- _____
- (h) _____ :- _____

O método dedutivo é uma alternativa mais operacional (embora menos fácil) para a demonstração de proposições do que a construção de tabelas verdade.

Sempre é possível representar qualquer proposição usando apenas os símbolos $\sim$, $\wedge$ e $\vee$, quando então a proposição diz-se estar na **forma normal**. Para passar à FN, deve-se trocar $a \rightarrow b$ por $\sim a \vee b$ e deve-se trocar $a \leftrightarrow b$ por $(\sim a \vee b) \wedge (\sim b \vee a)$.

Lembrando mais, que a forma normal conjuntiva de uma proposição é quando ela possui apenas $\sim$, $\wedge$ e $\vee$, e nenhum $\rightarrow$ atua sobre parênteses (apenas está dentro deles). Em outras palavras a proposição está em FNC quando todos seus componentes estão ligados apenas por $\wedge$.

Argumento

Um argumento é toda afirmação de que uma dada sequência de proposições $P_1, P_2, \dots, P_n$ tem como consequência a proposição Q . As proposições $P_1, P_2, \dots, P_n$ são chamadas premissas do argumento e Q é chamada conclusão do argumento. O argumento é indicado por $P_1, P_2, \dots, P_n \vdash Q$.

O símbolo $\vdash$ é lido como $P_1, \dots, P_n$ acarretam Q , ou Q decorre de $P_1, \dots, P_n$, ou Q se deduz de $P_1, \dots, P_n$, ou ainda Q se infere de $P_1, \dots, P_n$. Um argumento que consiste apenas de duas premissas e uma conclusão é chamado de **silogismo**.

Argumentos fundamentais

Daqui pode-se criar uma lista de argumentos fundamentais, a saber:

Nome	fórmula (leia $\vdash$ como acarreta...)
Adição	$p \vdash p \vee q$ ou $p \vdash q \vee p$
Simplificação	$p \wedge q \vdash p$ ou $p \wedge q \vdash q$
Conjunção	$p, q \vdash p \wedge q$ ou $p, q \vdash q \wedge p$
Absorção	$p \rightarrow q \vdash p \rightarrow (p \wedge q)$
Modus Ponens	$p \rightarrow q, p \vdash q$
Modus Tolens	$p \rightarrow q, \sim q \vdash \sim p$
Silogismo Disjuntivo	$p \vee q, \sim p \vdash q$ ou $p \vee q, \sim q \vdash p$
Silogismo Hipotético	$p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$
Dilema Construtivo	$p \rightarrow q, r \rightarrow s, p \vee r \vdash q \vee s$
Dilema Destrutivo	$p \rightarrow q, r \rightarrow s, \sim q \vee \sim s \vdash \sim p \vee \sim r$

Exemplo :

Verificar se o argumento $p \rightarrow q, p \wedge r \vdash q$ é ou não é válido.

- (1) $p \rightarrow q$
(2) $p \wedge r$

- (3) p , de 2 pelo argumento da simplificação
(4) q , de 1 e 3 pelo argumento do modus ponens
e está demonstrada a validade do argumento.

Outro exemplo :

Verificar se $p \wedge q, p \vee r \rightarrow s \vdash p \wedge s$ é ou não é válido.

- (1) $p \wedge q$
(2) $p \vee r \rightarrow s$

- (3) p de 1 pela regra da simplificação
(4) $p \vee r$ de 3 pela regra da adição
(5) s de 2 e 4 pela regra do Modus Ponens
(6) $p \wedge s$, de 3 e 5 pela regra da conjunção
e está demonstrada a validade do argumento.

Mais um :

Verificar se $p \rightarrow (q \rightarrow r), p \rightarrow q, p \vdash r$ é ou não é válido.

- (1) $p \rightarrow (q \rightarrow r)$
(2) $p \rightarrow q$
(3) p

- (4) $q \rightarrow r$ de 1 e 3 pela regra do Modus Ponens
(5) q de 2 e 3 pela regra do Modus Ponens
(6) r , de 4 e 5 pela regra do Modus Ponens
e está demonstrada a validade do argumento.

Demonstrar $p \rightarrow q, p \wedge q \rightarrow r, \sim (p \wedge r) \vdash \sim p$

- (1) $p \rightarrow q$
(2) $p \wedge q \rightarrow r$
(3) $\sim (p \wedge r)$

- (4) $p \rightarrow p \wedge q$ de 1 pela absorção
(5) $p \rightarrow r$, de 2 e 4 pelo silogismo hipotético
(6) $p \rightarrow p \wedge r$, de 5 pela absorção
(7) $\sim p$, de 3 e 6 por modus tolens.

Demonstrar $p \vee q \rightarrow r, r \vee q \rightarrow (p \rightarrow (s \leftrightarrow t)), p \wedge s \vdash s \leftrightarrow t$

- (1) $p \vee q \rightarrow r$,
(2) $r \vee q \rightarrow (p \rightarrow (s \leftrightarrow t))$
(3) $p \wedge s$

- (4) p de 3 por simplificação
(5) $p \vee q$, de 4 por adição
(6) r , de 1 e 5 por Modus ponens
(7) $r \vee q$, de 6 por adição
(8) $p \rightarrow (s \leftrightarrow t)$ de 2 e 7 por modus ponens
(9) $s \leftrightarrow t$, de 4 e 8 por modus ponens

Demonstrar $p \rightarrow \sim q, \sim p \rightarrow (r \rightarrow \sim q), (\sim s \vee \sim r) \rightarrow \sim \sim q, \sim s \vdash \sim r$

- (1) $p \rightarrow \sim q$
(2) $\sim p \rightarrow (r \rightarrow \sim q)$
(3) $(\sim s \vee \sim r) \rightarrow \sim \sim q$
(4) $\sim s$

- (5) $\sim s \vee \sim r$, de 4 por adição
(6) $\sim \sim q$, de 3 e 5 por modus ponens
(7) $\sim p$, de 1 e 6 por modus tolens
(8) $r \rightarrow \sim q$, de 2 e 7 por modus ponens
(9) $\sim r$, de 6 e 8 por modus tolens.

Demonstrar $p \wedge q \rightarrow r, r \rightarrow s, t \rightarrow \sim u, t, \sim s \vee u \vdash \sim (p \wedge q)$

- (1) $p \wedge q \rightarrow r$
(2) $r \rightarrow s$
(3) $t \rightarrow \sim u$
(4) t
(5) $\sim s \vee u$

- (6) $\sim u$, de 3 e 4 por modus ponens
(7) $\sim s$, de 5 e 6 por silogismo disjuntivo
(8) $\sim r$, de 2 e 7 por modus tolens
(9) $\sim (p \wedge q)$, de 1 e 8 por modus tolens.

Demonstrar $p \rightarrow q, q \rightarrow r, s \rightarrow t, p \vee s \vdash r \vee t$

- (1) $p \rightarrow q$
(2) $q \rightarrow r$
(3) $s \rightarrow t$
(4) $p \vee s$

- (5) $p \rightarrow r$, de 1 e 2 por silogismo hipotético
(6) $r \vee t$, de 3, 4 e 5 por dilema construtivo.

Demonstrar $p \rightarrow q, \sim r \rightarrow (s \rightarrow t), r \vee (p \vee s), \sim r \vdash q \vee t$

- (1) $p \rightarrow q$
(2) $\sim r \rightarrow (s \rightarrow t)$
(3) $r \vee (p \vee s)$
(4) $\sim r$

- (5) $s \rightarrow t$, de 2 e 4 por modus ponens
(6) $p \vee s$, de 3 e 4 por silogismo disjuntivo
(7) $q \vee t$ de 1, 5 e 6 por dilema construtivo.

 Para você fazer

1. $p \rightarrow q, (p \rightarrow r) \rightarrow s \vee q, p \wedge q \rightarrow r, \sim s \vdash q$ (110)

- (a) _____
(b) _____
(c) _____
(d) _____
(e) _____
(f) _____
(g) _____
(h) _____

2. $x = y \rightarrow x = z, x \neq y \rightarrow x < z, x \not< z \vee y > z, y \neq z \wedge x \neq z \vdash y > z$. (817)

- (a) _____
(b) _____
(c) _____
(d) _____
(e) _____
(f) _____
(g) _____
(h) _____

Argumentos fundamentais

Pode-se criar uma lista de argumentos fundamen-
tais, a saber:

Nome	fórmula (leia $\vdash$ como acarreta...)
Adição	$p \vdash p \vee q$ ou $p \vdash q \vee p$
Simplificação	$p \wedge q \vdash p$ ou $p \wedge q \vdash q$
Conjunção	$p, q \vdash p \wedge q$ ou $p, q \vdash q \wedge p$
Absorção	$p \rightarrow q \vdash p \rightarrow (p \wedge q)$
Modus Ponens	$p \rightarrow q, p \vdash q$
Modus Tolens	$p \rightarrow q, \sim q \vdash \sim p$
Silogismo Dis- juntivo	$p \vee q, \sim p \vdash q$ ou $p \vee q, \sim q \vdash p$
Silogismo Hipo- tético	$p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$
Dilema Cons- trutivo	$p \rightarrow q, r \rightarrow s, p \vee r \vdash q \vee s$
Dilema Destru- tivo	$p \rightarrow q, r \rightarrow s, \sim q \vee \sim s \vdash \sim p \vee \sim r$

Demonstrações

A. $q \rightarrow p, \sim x, q \vee x \vdash p$

- 1) $q \rightarrow p$
- 2) $\sim x$
- 3) $q \vee x$

- 4) q por silogismo disjuntivo, de 2 3
- 5) p por modus ponens, de 1 4

B. $y \rightarrow p, y \wedge q \vdash p$

- 1) $y \rightarrow p$
- 2) $y \wedge q$

- 3) y por simplificacao, de 2
- 4) p por modus ponens, de 1 3

C. $p \vee w, w \rightarrow x, x \rightarrow z, \sim z \vdash p$

- 1) $p \vee w$
- 2) $w \rightarrow x$
- 3) $x \rightarrow z$
- 4) $\sim z$

- 5) $\sim x$ por modus tolens, de 3 4
- 6) $\sim w$ por modus tolens, de 2 5
- 7) p por silogismo disjuntivo, de 6 1

D. $u \wedge y, u \rightarrow z, z \rightarrow p \vdash p$

- 1) $u \wedge y$
- 2) $u \rightarrow z$
- 3) $z \rightarrow p$

- 4) u $\rightarrow$ p por silogismo hipotético, de 2 3
- 5) u por simplificação, de 1
- 6) p por modus ponens, de 4 5

E. $z \rightarrow p, r \rightarrow q, s \rightarrow r, t \vee s, t \rightarrow w, w \rightarrow z \vdash$

- $p \vee q$
- 1) $z \rightarrow p$
- 2) $r \rightarrow q$
- 3) $s \rightarrow r$
- 4) $t \vee s$
- 5) $t \rightarrow w$
- 6) $w \rightarrow z$

- 7) t $\rightarrow$ z por silogismo hipotético, de 5 6
- 8) z $\vee$ r por dilema construtivo, de 7 3 4
- 9) p $\vee$ q por dilema construtivo, de 1 2 8

F. $\sim x, p \rightarrow x, w \rightarrow s, s \rightarrow q, y \rightarrow w, u \rightarrow$

- $p, y \vee u \vdash q$
- 1) $\sim x$
- 2) $p \rightarrow x$
- 3) $w \rightarrow s$
- 4) $s \rightarrow q$
- 5) $y \rightarrow w$
- 6) $u \rightarrow p$
- 7) $y \vee u$

- 8) w $\vee$ p por dilema construtivo, de 5 6 7
- 9) w $\rightarrow$ q por silogismo hipotético, de 3 4
- 10) q $\vee$ x por dilema construtivo, de 9 2 8
- 11) q por silogismo disjuntivo, de 1 10

G. $z \vee u, u \rightarrow x, \sim x, z \rightarrow q, q \rightarrow s \vdash s$

- 1) $z \vee u$
- 2) $u \rightarrow x$
- 3) $\sim x$
- 4) $z \rightarrow q$
- 5) $q \rightarrow s$

- 6) z $\rightarrow$ s por silogismo hipotético, de 4 5
- 7) $\sim u$ por modus tolens, de 2 3
- 8) z por silogismo disjuntivo, de 7 1
- 9) s por modus ponens, de 6 8

H. $\sim w, t \rightarrow z, x \rightarrow w, t \vee x, z \rightarrow y, y \rightarrow s \vdash s$

- 1) $\sim w$
- 2) $t \rightarrow z$
- 3) $x \rightarrow w$
- 4) $t \vee x$
- 5) $z \rightarrow y$
- 6) $y \rightarrow s$

- 7) z $\rightarrow$ s por silogismo hipotético, de 5 6
- 8) z $\vee$ w por dilema construtivo, de 2 3 4
- 9) z por silogismo disjuntivo, de 1 8
- 10) s por modus ponens, de 7 9

I. $q \rightarrow z, \sim z, v \rightarrow p, t \rightarrow r, r \rightarrow q, w \rightarrow v, u \rightarrow$

- $t, w \vee u \vdash p$
- 1) $q \rightarrow z$
- 2) $\sim z$
- 3) $v \rightarrow p$
- 4) $t \rightarrow r$
- 5) $r \rightarrow q$
- 6) $w \rightarrow v$
- 7) $u \rightarrow t$
- 8) $w \vee u$

- 9) v $\vee$ t por dilema construtivo, de 6 7 8
- 10) t $\rightarrow$ q por silogismo hipotético, de 4 5
- 11) p $\vee$ q por dilema construtivo, de 3 10 9
- 12) $\sim q$ por modus tolens, de 1 2
- 13) p por silogismo disjuntivo, de 12 11

🔗 Para você fazer

- 1.
 $\sim u, z \rightarrow p, w \rightarrow u, z \vee w \vdash p$

- 1) $\sim u$
- 2) $z \rightarrow p$
- 3) $w \rightarrow u$
- 4) $z \vee w$

- 2.
 $q \vee s, s \rightarrow w, \sim w \vdash q$

- 1) $q \vee s$
- 2) $s \rightarrow w$
- 3) $\sim w$

- 3.
 $y, y \rightarrow t, t \rightarrow r \vdash r$

- 1) y
- 2) $y \rightarrow t$
- 3) $t \rightarrow r$

- 4.
 $\sim v, u \rightarrow v, t \vee u, t \rightarrow x, x \rightarrow s \vdash s$

- 1) $\sim v$
- 2) $u \rightarrow v$
- 3) $t \vee u$
- 4) $t \rightarrow x$
- 5) $x \rightarrow s$

- 5.

$v \rightarrow t, v \vee q, q \rightarrow u, \sim u \vdash t$

- 1) $v \rightarrow t$
- 2) $v \vee q$
- 3) $q \rightarrow u$
- 4) $\sim u$

- 6. $u \vee y, y \rightarrow x, x \rightarrow q, \sim q \vdash u$

- 1) $u \vee y$
- 2) $y \rightarrow x$
- 3) $x \rightarrow q$
- 4) $\sim q$

- 7.
 $\sim r, w \rightarrow v, p \rightarrow r, x \rightarrow w, z \rightarrow p, x \vee z \vdash v$

- 1) $\sim r$
- 2) $w \rightarrow v$
- 3) $p \rightarrow r$
- 4) $x \rightarrow w$
- 5) $z \rightarrow p$
- 6) $x \vee z$

- 8.
 $r \rightarrow w, t \rightarrow y, r \vee t, y \rightarrow p, \sim p \vdash w$

- 1) $r \rightarrow w$
- 2) $t \rightarrow y$
- 3) $r \vee t$
- 4) $y \rightarrow p$
- 5) $\sim p$

- 9.
 $t \rightarrow q, q \rightarrow x, w \rightarrow t, w \vdash x$

- 1) $t \rightarrow q$
- 2) $q \rightarrow x$
- 3) $w \rightarrow t$
- 4) w

- 10.
O décimo exercício vai ser proposto (e resolvido) por você. Proponha um argumento e prove-o no verso.



Argumentos fundamentais

Pode-se criar uma lista de argumentos fundamen-
tais, a saber:

Nome	fórmula (leia ⊢ como acarreta...)
Adição	$p ⊢ p ∨ q$ ou $p ⊢ q ∨ p$
Simplificação	$p ∧ q ⊢ p$ ou $p ∧ q ⊢ q$
Conjunção	$p, q ⊢ p ∧ q$ ou $p, q ⊢ q ∧ p$
Absorção	$p → q ⊢ p → (p ∧ q)$
Modus Ponens	$p → q, p ⊢ q$
Modus Tolens	$p → q, ∼ q ⊢ ∼ p$
Silogismo Dis- juntivo	$p ∨ q, ∼ p ⊢ q$ ou $p ∨ q, ∼ q ⊢ p$
Silogismo Hipo- tético	$p → q, q → r ⊢ p → r$
Dilema Cons- trutivo	$p → q, r → s, p ∨ r ⊢ q ∨ s$
Dilema Destru- tivo	$p → q, r → s, ∼ q ∨ ∼ s ⊢ ∼ p ∨ ∼ r$

Demonstrações

A. $q → p, ∼ x, q ∨ x ⊢ p$
1) $q → p$
2) $∼ x$
3) $q ∨ x$

4) q por silogismo disjuntivo, de 2 3
5) p por modus ponens, de 1 4

B. $y → p, y ∧ q ⊢ p$
1) $y → p$
2) $y ∧ q$

3) y por simplificacao, de 2
4) p por modus ponens, de 1 3

C. se freemat é uma ótima alternativa ao matlab
então a política anda muito estridente, se a po-
lítica anda muito estridente então o Perú foi a
sede da colonia espanhola, freemat é uma ótima
alternativa ao matlab e o grampeador está repleto
de grampos acarreta o Perú foi a sede da colonia
espanhola
u = freemat é uma ótima alternativa ao matlab
s = a política anda muito estridente
q = o Perú foi a sede da colonia espanhola
t = o grampeador está repleto de grampos

1. $u → s,$
2. $s → q,$
3. $u ∧ t$

4. u, de 3 por simplificação
5. s, de 4 e 1 por modus ponens
6. q de 5 e 2 por modus ponens

D. o altofalante produz ruido estridente ou a secre-
tária é sempre muito eficiente, não a praça osório
é muito bonita, se a secretária é sempre muito
eficiente então o telefone ficou mudo, se o telefone
ficou mudo então a praça osório é muito bonita,
se o altofalante produz ruido estridente então as
flores são muito olorosas, se as flores são muito
olorosas então o Chile é um país muito comprido
acarreta o Chile é um país muito comprido
x = o altofalante produz ruido estridente
v = secretária é sempre muito eficiente
t = a praça osório é muito bonita
y = o telefone ficou mudo
r = as flores são muito olorosas
q = o Chile é um país muito comprido

1. $x ∨ v,$
2. $∼ t,$
3. $v → y,$
4. $y → t,$
5. $x → r,$
6. $r → q$

7. $∼ y,$ de 4 e 2 por modus tollens
8. $∼ v,$ de 7 e 3 por modus tollens
9. x, de 8 e 1 por silogismo disjuntivo
10. r de 9 e 5 por modus ponens
11. q de 10 e 6 por modus ponens

E. não a lapiseira exige grafite de boa qualidade,

se a caneta vermelha terá muito trabalho então
a secretária é sempre muito eficiente, se freemat
é uma ótima alternativa ao matlab então gatos
são animais amistosos e companheiros, se gatos
são animais amistosos e companheiros então a
lapiseira exige grafite de boa qualidade, se beetho-
ven sempre manda ver legal então a política anda
muito estridente, se morangos são ótima sobremesa
então freemat é uma ótima alternativa ao matlab,
beethoven sempre manda ver legal ou morangos
são ótima sobremesa, se a política anda muito
estridente então o cachorro mordeu o professor,
se o cachorro mordeu o professor então a caneta
vermelha terá muito trabalho acarreta a secretária
é sempre muito eficiente.

q = a lapiseira exige grafite de boa qualidade
p = a caneta vermelha terá muito trabalho
r = a secretária é sempre muito eficiente
v = freemat é uma ótima alternativa ao matlab
w = gatos são animais amistosos e companheiros
t = beethoven sempre manda ver legal
x = a política anda muito estridente
y = morangos são ótima sobremesa
s = o cachorro mordeu o professor

1. $∼ q,$
2. $p → r,$
3. $w → v,$
4. $v → q,$
5. $t → x,$
6. $y → w,$
7. $t ∨ y,$
8. $x → s,$
9. $s → p$

10. $∼ v$ de 1 e 4 por modus tollens
11. $∼ w$ de 10 e 3 por modus tollens
12. $∼ y$ de 11 e 6 por modus tollens
13. t de 12 e 7 por silogismo disjuntivo
14. x de 13 e 5 por modus ponens
15. s de 14 e 8 por modus ponens
16. p de 15 e 9 por modus ponens
17. r de 16 e 2 por modus ponens

Para você fazer

1.
 $r → p, ∼ z, r ∨ z ⊢ p$

1) $r → p$
2) $∼ z$
3) $r ∨ z$

2.
 $x → q, y → x, y ⊢ q$

1) $x → q$
2) $y → x$
3) y

3.
 $r ∨ x, x → z, ∼ z ⊢ r$

1) $r ∨ x$
2) $x → z$
3) $∼ z$

4.
 $u → s, ∼ p, t → u, y → p, t ∨ y ⊢ s$

1) $u → s$
2) $∼ p$
3) $t → u$
4) $y → p$
5) $t ∨ y$

As 6 próximas questões devem ser respondi-
das no verso ou em uma folha anexa

5. se a caneta vermelha terá muito trabalho então a
xícara vermelha está cheia de café, a caneta verme-
lha terá muito trabalho ou pitágoras fundou uma
sociedade secreta, se pitágoras fundou uma socie-
dade secreta então a música chama-se as mocinhas
da cidade, se a música chama-se as mocinhas da
cidade então fermat foi um grande matemático, se
fermat foi um grande matemático então a prova fi-
cou para o período da tarde, não a prova ficou para
o período da tarde acarreta a xícara vermelha está
cheia de café

6. se a borracha serve para apagar erros então gatos
são animais amistosos e companheiros, se gatos são
animais amistosos e companheiros então precisam-
se muitas folhas de papel, se o grampeador está
repleto de grampos então a borracha serve para
apagar erros, o grampeador está repleto de gram-
pos acarreta precisam-se muitas folhas de papel

7. não o telefone ficou mudo, se beethoven sempre
manda ver legal então o telefone ficou mudo, a
praça osório é muito bonita ou beethoven sempre
manda ver legal, se a caneta vermelha terá muito
trabalho então a borracha serve para apagar erros,
se a praça osório é muito bonita então morangos
são ótima sobremesa, se morangos são ótima sobre-
mesa então a caneta vermelha terá muito trabalho
acarreta a borracha serve para apagar erros

8. não Londres é a cidade de Sherlock Holmes, se a casa
caiu então a impressora parou de funcionar, a casa
caiu ou pitágoras fundou uma sociedade secreta,
se pitágoras fundou uma sociedade secreta então a
borracha serve para apagar erros, se a borracha
serve para apagar erros então Londres é a cidade
de Sherlock Holmes acarreta a impressora parou de
funcionar

9. cães são ótima companhia e a ordem é que vigora a
lei do silêncio, se cães são ótima companhia então
as flores são muito olorosas, se as flores são muito
olorosas então a praça osório é muito bonita acar-
reta a praça osório é muito bonita

10. se Carlos desistiu de sua empreitada então cães são
ótima companhia, Carlos desistiu de sua emprei-
tada ou pitágoras fundou uma sociedade secreta,
se pitágoras fundou uma sociedade secreta então a
impressora parou de funcionar, se a impressora
parou de funcionar então o Chile é um país muito
comprido, se o Chile é um país muito comprido en-
tão o cachorro mordeu o professor, não o cachorro
mordeu o professor acarreta cães são ótima compa-
nhia



- 1 - /

Demonstração mediante tabelas verdade Embora muito trabalhoso, continua sendo possível provar argumentos usando tabelas verdade. É um procedimento automático (não exige aplicação de inteligência). Em resumo, Dado um argumento, por exemplo $P_1, P_2, \dots, P_n \vdash Q$ cumpre constatar se há pelo menos um caso de Q ser falso enquanto todos os P_i são verdadeiros.

- Para isso, deve-se proceder assim:
- ★ Construir uma tabela verdade e identificar as colunas das premissas (P_i) e da conclusão (Q).
 - ★ Marcar as linhas em que todas as premissas P_i são verdadeiras
 - ★ Nessas linhas, se a coluna Q for verdadeira, o argumento está provado
 - ★ Se alguma dessas linhas Q for falsa, o argumento é um sofisma.

Condicional associada Uma outra maneira que pode ser usada para efetuar esta análise é considerar a **condicional associada** ao argumento, que se obtém trocando as vírgulas no antecedente por $\wedge$ e o sinal de $\vdash$ por um $\rightarrow$. Depois disto, deve-se ver se a expressão resultante é uma tautologia. Se for, o argumento é válido e está demonstrado. Se não for, é um sofisma.

Validade mediante regras de inferência e equivalências Já se estudou nas aulas passadas como demonstrar argumentos usando apenas as 10 regras de inferência, cujo resumo está aqui:

Nome	fórmula (leia $\vdash$ como acarreta...)
Adição	$p \vdash p \vee q$ ou $p \vdash q \vee p$
Simplificação	$p \wedge q \vdash p$ ou $p \wedge q \vdash q$
Conjunção	$p, q \vdash p \wedge q$ ou $p, q \vdash q \wedge p$
Absorção	$p \rightarrow q \vdash p \rightarrow (p \wedge q)$
Modus Ponens	$p \rightarrow q, p \vdash q$
Modus Tolens	$p \rightarrow q, \sim q \vdash \sim p$
Silogismo Disjuntivo	$p \vee q, \sim p \vdash q$ ou $p \vee q, \sim q \vdash p$
Silogismo Hipotético	$p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$
Dilema Construtivo	$p \rightarrow q, r \rightarrow s, p \vee r \vdash q \vee s$
Dilema Destrutivo	$p \rightarrow q, r \rightarrow s, \sim q \vee \sim s \vdash \sim p \vee \sim r$

Agora a isto vai-se juntar a décima primeira regra, chamada **Regra da substituição** que diz uma proposição P ou apenas uma parte de P pode ser substituída por uma proposição equivalente e a proposição Q que assim se obtém é equivalente a P .

idempotência da conjunção	$p \wedge p \Leftrightarrow p$
comutativa da conjunção	$p \wedge q \Leftrightarrow q \wedge p$
associativa da conjunção	$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$
idempotência da disjunção	$p \vee p \Leftrightarrow p$
comutativa da disjunção	$p \vee q \Leftrightarrow q \vee p$
associativa da disjunção	$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
distributiva	$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
distributiva	$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
absorção	$p \wedge (p \vee q) \Leftrightarrow p$
absorção	$p \vee (p \wedge q) \Leftrightarrow p$
De Morgan	$\sim (p \wedge q) \Leftrightarrow \sim p \vee \sim q$
De Morgan	$\sim (p \vee q) \Leftrightarrow \sim p \wedge \sim q$
Condicional	$p \rightarrow q \Leftrightarrow \sim p \vee q$
Dupla negação	$p \Leftrightarrow \sim \sim p$

Exemplos A. saio para a balada e não estou elegante, se não uso calça boca de sino então estou elegante acarreta saio para a balada e uso calça

boca de sino.
Solução:
a = saio para a balada; b = estou elegante;
c = uso calça boca de sino
 $a \wedge \sim b, (\sim c \rightarrow b) \vdash a \wedge c$
1) $a \wedge \sim b$
2) $\sim c \rightarrow b$
3) $\sim b$ de 1 por $(p \wedge q \vdash p)$
4) $\sim \sim c$ de 2 e 3 por M.T.
5) c de 4 por $(\sim \sim p \vdash p)$
6) a de 1 por $(p \wedge q \vdash p)$
7) $a \wedge c$ de 5 e 6. Logo 1.

B. se estudo então deixo de sair, se compro lapiseira então tenho fome, não tenho fome, não estudo acarreta não deixo de sair.
a = estudo; b = deixo de sair
c = compro lapiseira; d = tenho fome
 $a \rightarrow b, c \rightarrow d, \sim d, \sim a \vdash \sim b$

Não há o que fazer, logo 0.
C. se cheiro flores então não sou companheiro, cheiro flores, se não sou companheiro então ajudo os outros acarreta ajudo os outros.
a = cheiro flores; b = sou companheiro
c = ajudo os outros
 $a \rightarrow \sim b, a, \sim b \rightarrow c \vdash c$
1) $a \rightarrow \sim b$
2) a
3) $\sim b \rightarrow c$
4) $\sim b$ de 1 e 2 por M.P.
5) c de 3 e 4 por M.P. Logo 1

D. compro livros ou gosto de sair, se gosto de sair então preciso beber, se compro livros então tenho boa vontade, não tenho boa vontade acarreta preciso beber e(compro livros ou gosto de sair)
a = compro livros
b = gosto de sair
c = preciso beber
d = tenho boa vontade
 $a \vee b, b \rightarrow c, (a \rightarrow d), \sim d \vdash c \wedge (a \vee b)$
1) $a \vee b$
2) $b \rightarrow c$
3) $a \rightarrow d$
4) $\sim d$
5) $\sim a$ por M.T. de 3 e 4
6) b por $(p \vee q, \sim p \vdash q)$ de 1 e 5
7) c por M.P. de 2 e 6
8) $c \wedge (a \vee b)$ de 7 e 1. Logo 1.

E. se chove então faz calor, não chove acarreta não faz calor
a = chove
b = faz calor
 $a \rightarrow b, \sim a \vdash \sim b$
Não há o que fazer, logo 0.

🔗 Para você fazer

Use qualquer um dos métodos de demonstração estudados para determinar se os argumentos a seguir são ou não válidos.

- se gosto de Beethoven então gosto de Bach, se curto Xitaozinho então sou surdo, não sou surdo, não gosto de Beethoven acarreta não gosto de Bach
- se sou dentista então sou mascate, se sou analista então gosto de comprar, se sou analista então sou mascate, não sou dentista acarreta sou analista
- se sou coxabranca então sou atleticano, se sou paranista então não sou atleticano acarreta se sou paranista então não sou coxabranca
- se saio para a balada então estou elegante, se saio para a balada então uso calça boca de sino, não uso calça boca de sino acarreta estou elegante
- não uso calça boca de sino, se saio para a balada então estou elegante, se saio para a balada então uso calça boca de sino acarreta uso calça boca de sino

- se estudo então deixo de sair, compro lapiseira ou não deixo de sair, não compro lapiseira acarreta não estudo
- se cheiro flores então sou companheiro ou ajudo os outros, não sou companheiro acarreta se cheiro flores então ajudo os outros
- se gosto de amarelo então sou comunista, se vivo no mundo da lua então pulo da cachoeira, não pulo da cachoeira, não gosto de amarelo acarreta não sou comunista
- se sou dentista então sou mascate, gosto de comprar, se sou analista então sou o melhor, sou mascate, não sou o melhor acarreta não sou dentista
- leio livros e não gosto de sair, se não quero beber então gosto de sair acarreta leio livros e quero beber
- se chove então faz calor, se uso camiseta então não faz calor acarreta se uso camiseta então não chove
- se saio para a balada então estou elegante, estou elegante acarreta saio para a balada
- se gosto de Beethoven então gosto de Bach, não curto Xitaozinho, se gosto de Beethoven então curto Xitaozinho acarreta curto Xitaozinho
- saio para a balada ou estou elegante, se estou elegante então uso calça boca de sino, se saio para a balada então está chovendo, não está chovendo acarreta uso calça boca de sino e(saio para a balada ou estou elegante)
- se compro livros então preciso beber, tenho boa vontade, se gosto de sair então vendo papel, preciso beber, não vendo papel acarreta não tenho boa vontade
- se gosto de amarelo então sou comunista, não vivo no mundo da lua, se gosto de amarelo então vivo no mundo da lua acarreta vivo no mundo da lua
- se cheiro flores então não sou companheiro, se ajudo os outros então cheiro flores, sou companheiro acarreta não ajudo os outros
- se sou coxabranca então sou atleticano ou sou paranista, não sou atleticano acarreta se sou coxabranca então sou paranista
- se leio livros então quero beber, tenho preguiça, se gosto de sair então compro papel, quero beber, não compro papel acarreta não tenho preguiça
- se sou dentista então sou analista, se sou mascate então não sou analista acarreta se sou mascate então não sou dentista

Responda 1 se o argumento correspondente for VÁLIDO e responda 0 se for INVÁLIDO.

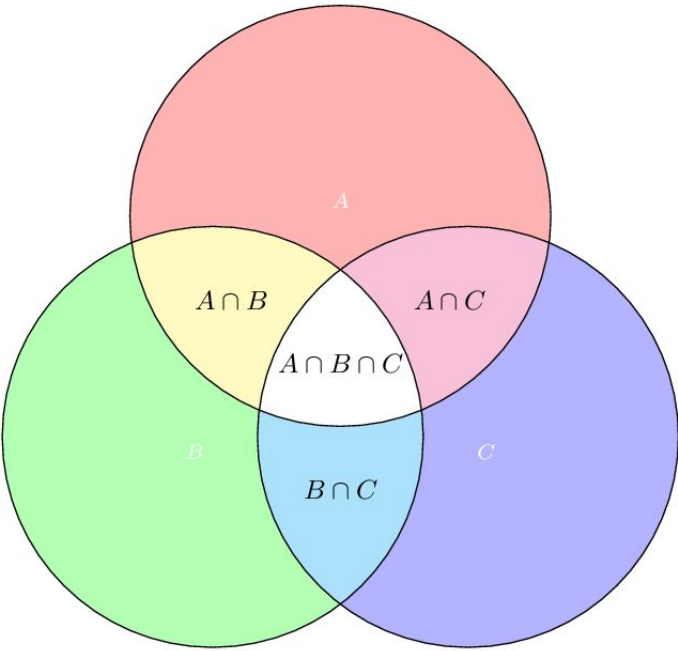
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20



Lógica e Diagramas de Venn

Nos exercícios a seguir, vão-se definir sempre 3 conjuntos, a saber: A, B e C. Depois, serão definidos 10 conjuntos derivados daqueles três originais, sempre segundo as regras a seguir

- M** {m/m ∈ A ∧ m ∈ B ∧ m ∈ C}
- N** {n/n ∈ A ∧ n ∈ B ∧ n ∉ C}
- P** {p/p ∈ A ∧ p ∉ B}
- Q** {q/q ∈ C ∧ ∼ (q ∈ A ∨ q ∈ B)}
- R** {r/r ∈ A ∧ (r ∈ B ∨ r ∈ C)}
- S** {s/(s ∈ B ∧ (∼ ((s ∈ A) ∨ (s ∈ C)))) ∨ ((s ∈ A ∧ s ∈ C) ∧ (∼ (s ∈ A ∧ s ∈ B ∧ s ∈ C)))}
- T** {t/t ∈ A ∨ t ∈ B ∨ t ∈ C}
- U** {u/((u ∈ A ∧ u ∈ B) ∧ u ∉ C) ∨ ((u ∈ A ∧ u ∈ C) ∧ u ∉ B) ∨ ((u ∈ B ∧ u ∈ C) ∧ u ∉ A)}
- V** {v/(v ∈ A ∧ v ∈ B) ∨ (v ∈ A ∧ v ∈ C) ∨ (v ∈ B ∧ v ∈ C)}
- W** {w/(w ∈ A ∨ w ∈ B ∨ w ∈ C) ∧ (∼ ((w ∈ A ∧ w ∈ B) ∨ (w ∈ A ∧ w ∈ c) ∨ (w ∈ B ∧ w ∈ C)))}



👉 Para você fazer

Desenhe o diagrama de Venn para estes 3 conjuntos e informe a cardinalidade (o número de elementos de) dos seguintes conjuntos:

exercício	conj								
1	A	12	7	1	4	3	6	2	8
	B	10	2	1	4	13	11	8	7
	C	7	2	12	9	13	8	15	1
2	A	7	4	11	6	1	5	14	15
	B	13	11	10	7	9	4	5	14
	C	15	10	12	11	5	14	9	7
3	A	4	2	10	7	12	5	3	11
	B	11	6	14	13	7	10	9	4
	C	5	14	3	11	8	9	6	1

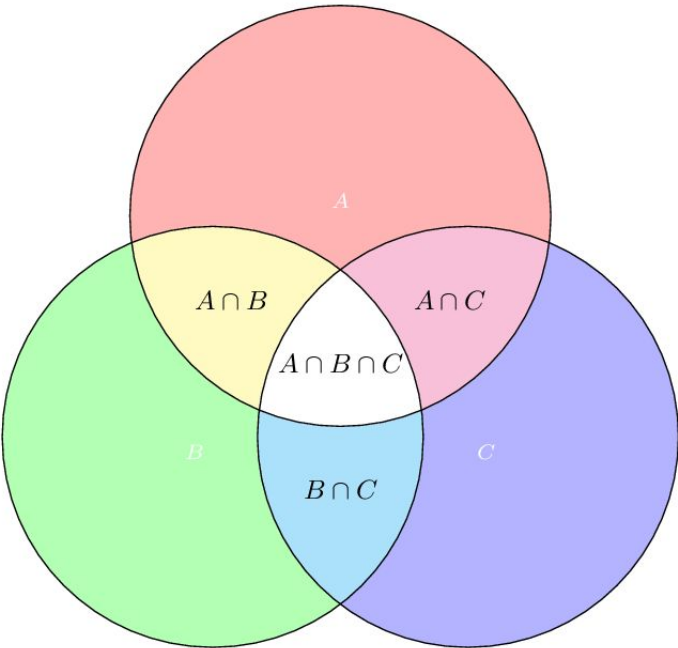
Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

exercício	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
SOMA										

Teoria de Conjuntos e Diagramas de Venn

Nos exercícios a seguir, vão-se definir sempre 3 conjuntos, a saber: A, B e C. Depois, serão definidos 10 conjuntos derivados daqueles três originais, sempre segundo as regras a seguir

$M = A \cap B \cap C$
 $N = A \cap B \cap C^c$
 $P = A \cap B^c$
 $Q = C \cap (A \cup B)^c$
 $R = A \cap (B \cup C)$
 $S = (B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
 $T = A \cup B \cup C$
 $U = (A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
 $V = (A \cap B) \cup (A \cap C) \cup (B \cap C)$
 $W = (A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$



 Para você fazer

Desenhe o diagrama de Venn para estes 3 conjuntos e informe a cardinalidade (o número de elementos de) dos seguintes conjuntos:

exercício	conj								
1	A	5	7	6	1	2	10	15	13
	B	6	15	10	5	4	1	14	3
	C	12	4	14	5	6	9	13	2
2	A	9	10	2	12	6	3	1	4
	B	13	12	1	9	10	14	4	7
	C	9	8	3	10	11	15	6	2
3	A	11	13	12	15	6	2	5	9
	B	15	11	2	13	14	6	3	4
	C	3	7	1	8	11	12	9	15

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

exercício	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
SOMA										

Expressões lógicas

Para cada um dos exercícios a seguir, escreva o enunciado pedido em Lógica simbólica ou em lógica de predicados, conforme o caso. Utilize a seguinte tabela de símbolos:

$\forall$	Para todo
$\exists$	Existe pelo menos um
$\exists!$	Existe um e só um
$\wedge$	E
$\vee$	OU
$\rightarrow$	Implica
$\sim$	NÃO

Exemplos

- Ladrão que rouba ladrão, tem cem anos de perdão.

- Supondo definido o predicado Filho(x,y) defina o predicado Avo(x,y).

- Bateu, levou.

- Em terra de cego, quem tem um olho é rei.

- Quem age obtém resultado.

- Para bom entendedor, meia palavra basta.

- Morreu de repente: ou foi vento pelas costas ou sogra pela frente.

- $\forall x \text{ Nasce}(x, \text{EspiritoSanto}) \rightarrow \text{Capixaba}(x)$

- $\forall x \text{ Nasce}(x, \text{Rio}) \vee \text{GostaCar}(x) \rightarrow \text{Capixaba}(x)$

- $\forall x \exists y \text{ GrandeHomem}(x) \wedge \text{GrandeMulher}(y) \rightarrow \text{Atrás}(x, y)$

- $\forall x \text{ Aula}(\text{frances}, x) \vee \text{ Aula}(\text{ingles}, x) \rightarrow \text{Passou}(x)$

- $\text{Cunhado}(x, y) \rightarrow \text{Irmão}(y, z) \wedge \text{Cônjuge}(x, z)$

- $\forall x \text{ VisitaSan}(x) \rightarrow \text{Fala}(x, \text{Prov}). \forall x \sim \text{Fala}(x, \text{Prov}) \rightarrow \text{NãoF}(x)$

- $\text{Chuva} \wedge \text{Sol} \rightarrow \text{CasamentoEspanhol}. \text{Sol} \wedge \text{Chuva} \rightarrow \text{CasamentoViuva}$

- Para você fazer

De Português para Lógica

Converta a frase em português para sua representação em lógica de predicados.

- Se a montanha não vem a Maomé, Maomé vai à montanha.

- Supondo definidos os predicados Filho(x,y) e Cônjuge(x,y), defina o predicado Nora(x,y).

- Porque está no artigo 13 da Constituição, o Português é a língua oficial do País.

- Tancredo vence no Colégio eleitoral se e somente se o Exército estiver de acordo com a sua eleição.

- Onde há fumaça, há fogo.

- Quem nasce no Rio Grande do Sul ou na Argentina é gaúcho.

- Nenhuma pessoa compra um seguro caro

- A melhor pontuação em tupi-guarani é sempre mais alta que a melhor pontuação em francês.

- Uma maçã caiu na cabeça de Newton. Newton foi um gênio.

- Somente um aluno teve aulas de tupi-guarani em 2003.

De Lógica para Português

Converta agora de lógica de predicados para a Língua Portuguesa

- $\forall x \text{ Gato}(x) \wedge \text{Enoite} \rightarrow \text{Pardo}(x)$

- $\forall x [\text{ContraC}(x) \rightarrow \text{Queima}(x)] \vee [\text{FavorC}(x) \rightarrow \text{Queima}(x)]$

- $\forall x \text{ MoraCG}(x) \vee \text{Dinh}(x, \text{muito}) \rightarrow \text{MedoS}(x)$

- $\forall x \text{ Pobre}(x) \wedge \text{Come}(x, \text{carne}) \rightarrow \text{Morde}(x, \text{língua})$

- $\text{Rola}(\text{Tomate}) \wedge \text{Rola}(\text{Laranja}) \wedge \sim \text{Rola}(\text{Batata})$

- $\forall x \text{ MoraCP}(x) \wedge \text{Dinh}(x, \text{pouco}) \rightarrow \sim \text{MedoS}(x)$

- $\forall x \text{ Obvia}(x) \vee \sim \text{Inteligível}(x) \rightarrow \text{Inútil}(x)$

- $\forall x \text{ Mequetrefe}(x) \rightarrow \text{Patife}(x) \vee \text{JN}(x) \vee [\text{Patife}(x) \wedge \text{JN}(x)]$

- $\forall x \forall y \text{ Bicudo}(x) \wedge \text{Bicudo}(y) \rightarrow \sim \text{Bica}(x, y)$

- $\forall x \text{ Inteligente}(x) \vee \text{Destemido}(x) \rightarrow \text{CompraSeguroCaro}(x)$



- 1 - /

Exemplos da folha 102

1. Ladrão que rouba ladrão, tem cem anos de perdão.
 $\forall x \forall y \text{Ladrão}(x) \wedge \text{Ladrão}(y) \wedge \text{Rouba}(x, y) \rightarrow \text{CemAP}(x)$
2. Supondo definido o predicado Filho(x,y) defina o predicado Avo(x,y).
 $\text{Avo}(x, y) \rightarrow \text{Filho}(y, z) \wedge \text{Filho}(z, x)$
3. Bateu, levou.
 $\forall x \text{Bateu}(x) \rightarrow \text{Levou}(x)$
4. Em terra de cego, quem tem um olho é rei.
 $\forall x \text{Está}(x, \text{emterradecego}) \wedge \text{Tem1olho}(x) \rightarrow \text{Rei}(x)$
5. Quem age obtem resultado.
 $\forall x \text{Age}(x) \rightarrow \text{ObtémR}(x)$
6. Para bom entendedor, meia palavra basta.
 $\forall x \text{BomEntendedor}(x) \rightarrow \text{Basta}(x, \text{meiapalavra})$
7. Morreu de repente: ou foi vento pelas costas ou sogra pela frente.
 $\forall x \text{Morreu}(x) \rightarrow \text{VentoC}(x) \vee \text{SograF}(x)$
8. Quem nasce no Espírito Santo é capixaba.
 $\forall x \text{Nasce}(x, \text{EspiritoSanto}) \rightarrow \text{Capixaba}(x)$
9. Quem nasce no Rio ou gosta de carnaval, é carioca.
 $\forall x \text{Nasce}(x, \text{Rio}) \vee \text{GostaCar}(x) \rightarrow \text{Carioca}(x)$
10. Atrás de um grande homem. há sempre uma grande mulher.
 $\forall x \exists y \text{GrandeHomem}(x) \wedge \text{GrandeMulher}(y) \rightarrow \text{Atrás}(x, y)$
11. Todos os alunos que tiveram aulas de francês ou de inglês, passaram de ano.
 $\forall x \text{Aula}(\text{frances}, x) \vee \text{Aula}(\text{ingles}, x) \rightarrow \text{Passou}(x)$
12. Supondo definidos os predicados Irmão(x,y) e Cônjuge(x,y) defina Cunhado (x,y)
 $\text{Cunhado}(x, y) \rightarrow \text{Irmão}(y, z) \wedge \text{Cônjuge}(x, z)$
13. Quem visita Santiago fala provençal. Se não fala, não pode ficar lá.
 $\forall x \text{VisitaSan}(x) \rightarrow \text{Fala}(x, \text{Prov}). \forall x \sim \text{Fala}(x, \text{Prov}) \rightarrow \text{NãoF}(x)$
14. Chuva e sol, casamento de espanhol. Sol e chuva, casamento de viúva.
 $\text{Chuva} \wedge \text{Sol} \rightarrow \text{CasamentoEspanhol}. \text{Sol} \wedge \text{Chuva} \rightarrow \text{CasamentoViuva}$

Existe um agente que só vende seguros às pessoas que não têm seguro
 $\exists x \forall y \text{Nãotem}(y) \rightarrow \text{VendeSeguro}(x, y)$

Pelo menos uma pessoa ganhará na loteria. Quem ganha na loteria é rico .
 $\exists x \text{GanhaL}(x). \forall x \text{GanhaL}(x) \rightarrow \text{Rico}(x)$

A noite, todos os gatos são pardos.
 $\forall x \text{Gato}(x) \wedge \text{Enoite} \rightarrow \text{Pardo}(x)$

O árabe teria dito: se são contrários ao Corão devem ser queimados . Se são favoráveis, são supérfluos e devem ser queimados.
 $\forall x [\text{ContraC}(x) \rightarrow \text{Queima}(x)] \vee [\text{FavorC}(x) \rightarrow \text{Queima}(x)]$

Quem mora em cidades grandes ou tem muito dinheiro, tem muito medo de sequestros.
 $\forall x \text{MoraCG}(x) \vee \text{Dinh}(x, \text{muito}) \rightarrow \text{MedoS}(x)$

O pobre só come carne quando morde a língua.
 $\forall x \text{Pobre}(x) \wedge \text{Come}(x, \text{carne}) \rightarrow \text{Morde}(x, \text{língua})$

Tomates rolam. Laranjas rolam. Batatas não rolam.
 $\text{Rola}(\text{Tomate}) \wedge \text{Rola}(\text{Laranja}) \wedge \sim \text{Rola}(\text{Batata})$

Existe um barbeiro na aldeia que faz a barba a todas as pessoas que não se barbeiam.
 $\exists x \forall y \text{Barbeia}(x, y) \rightarrow \sim \text{Barbeia}(y, y)$

Quem ama os animais não os machuca. Estrangeiros não amam os animais.
 $\forall x \text{Ama}(x, \text{animais}) \rightarrow \sim \text{Machuca}(x, \text{animais})$

Quem mora em cidades pequenas e tem pouco dinheiro, não tem medo de sequestros.
 $\forall x \text{MoraCP}(x) \wedge \text{Dinh}(x, \text{pouco}) \rightarrow \sim \text{MedoS}(x)$

Toda pessoa que compra um seguro é inteligente
 $\forall x \text{Compraseguro}(x) \rightarrow \text{Inteligente}(x)$

Cavalos, vacas e porcos são mamíferos. Mamíferos são vertebrados
 $\text{Mam}(\text{Cav}) \wedge \text{Mam}(\text{Vac}) \wedge \text{Mam}(\text{Por}). \forall x \text{Mam}(x) \rightarrow \text{Vert}(x)$

Palavras óbvias ou ininteligíveis são inúteis.
 $\forall x \text{Obvia}(x) \vee \sim \text{Inteligível}(x) \rightarrow \text{Inútil}(x)$

Quem é mequetrefe é um patife ou um João-ninguém. Ou ambas as coisas .
 $\forall x \text{Mequetrefe}(x) \rightarrow \text{Patife}(x) \vee \text{JN}(x) \vee [\text{Patife}(x) \wedge \text{JN}(x)]$

Quem vende armas aos nossos inimigos é nosso inimigo.
 $\forall x \forall y \text{VendeArma}(x, y) \wedge \text{Inimigomeu}(y) \rightarrow \text{Inimigomeu}(x)$

Dois bicudos não se beijam.

$\forall x \forall y \text{Bicudo}(x) \wedge \text{Bicudo}(y) \rightarrow \sim \text{Bica}(x, y)$

Pessoas inteligentes ou destemidas, compram seguros caros
 $\forall x \text{Inteligente}(x) \vee \text{Destemido}(x) \rightarrow \text{CompraSeguroCaro}(x)$

Pessoas que gostam de sorvete, vestem azul ou vestem branco.
 $\forall x \text{GostaS}(x) \rightarrow \text{Veste}(x, \text{azul}) \vee \text{Veste}(x, \text{branco})$

Cão não come cão.
 $\forall x \forall y \text{Cão}(x) \wedge \text{Cão}(y) \rightarrow \sim \text{Come}(x, y)$

Supondo definido o predicado Pai(x,y) defina o predicado Avo(x,y).
 $\text{Avo}(x, y) \rightarrow \text{Pai}(x, z) \wedge \text{Pai}(z, y)$

Amigos, amigos, negócios à parte.
 $\forall x \text{Amigo}(x) \rightarrow [\text{FazNegocios}(x) \vee \sim \text{FazNegocios}(x)]$

Todo mundo que ama os animais é amado por alguém.
 $\forall x \text{Ama}(x, \text{animais}) \rightarrow \exists y \text{Ama}(y, x)$

Deus ajuda a quem cedo madruga.
 $\forall x \text{CedoM}(x) \rightarrow \text{DeusAj}(x)$

Supondo definidos os predicados Filho(x,y) e Cônjuge(x,y), defina o predicado Nora(x,y).
 $\text{Nora}(x, y) \rightarrow \text{Filho}(x, z) \wedge \text{Cônjuge}(z, y)$

Uma esfera tem área e volume. Quem tem volume ocupa 3 dimensões.
 $\text{Area}(\text{Esfera}) \wedge \text{Volume}(\text{Esfera}). \forall x \text{Volume}(x) \rightarrow \text{Dimensão}(x, 3)$

A cavalo dado não se olha o dente.
 $\forall x \text{DadoC}(x) \rightarrow \text{NaoODC}(x)$

Quando mil pessoas dizem a mesma coisa, pode ser a voz de Deus, ou pode ser uma grande besteira.
 $\forall x \text{MilPessoasDizem}(x) \rightarrow \text{DeusDiz}(x) \vee \text{UmaBesteira}(x)$

Quem percebe age.
 $\forall x \text{Percebe}(x) \rightarrow \text{Age}(x).$

Supondo definidos os predicados Pai(x,y), Mãe(x,y) Cônjuge(x,y), defina o predicado Genro (x,y).
 $\text{Genro}(a, b) \rightarrow \text{Cônjuge}(b, c) \wedge [\text{Pai}(c, a) \vee \text{Mãe}(c, a)]$

Supondo os predicados Pai(x,y), Irmão(x,y), defina o predicado Tio(x,y).
 $\text{Tio}(a, b) \rightarrow \text{Pai}(a, c) \wedge \text{Irmão}(c, b)$

Qualquer pessoa que conheça mais de 700 palavras de um idioma, já é considerada falante neste idioma.
 $\forall x \forall y \text{ConheceM700}(x, y) \rightarrow \text{Falante}(x, y)$

Nós só compramos armas a nossos amigos.
 $\text{CompraArma}(\text{Nós}, x) \rightarrow \text{Amigomeu}(x)$

Quem nasce no Estado de São Paulo é paulista.
 $\forall x \text{Nasce}(x, \text{SP-Estado}) \rightarrow \text{Paulista}(x)$

Onde quer que esteja a língua romana, aí está Roma.
 $\forall x \text{Está}(\text{línguaR}, x) \rightarrow \text{Está}(\text{Roma}, x)$

Supondo definido o predicado Filho(x,y) defina o predicado Irmão(x,y).
 $\text{Irmão}(x, y) \rightarrow \text{Filho}(z, x) \wedge \text{Filho}(z, y)$

Supondo definidos os predicados Filho(x,y) e Cônjuge(x,y), defina o predicado Genro (x,y).
 $\text{Genro}(x, y) \rightarrow \text{Filho}(x, z) \wedge \text{Cônjuge}(z, y)$

Nem esta pintura é barroca, nem esta música é impressionista.
 $\sim [\text{Barroca}(\text{Pintura}) \wedge \text{Impressionista}(\text{Música})]$

Supondo definido o predicado Pai(x,y) defina o predicado Irmão(x,y).
 $\text{Irmão}(x, y) \rightarrow \text{Pai}(x, z) \wedge \text{Pai}(y, z)$

Capitu e Escobar foram amantes mesmo ou tudo não passou de paranóia de Bentinho?
 $\text{Amante}(\text{Capitu}, \text{Escobar}) \vee [\sim \text{Amante}(\text{Capitu}, \text{Escobar}) \wedge \text{Paranóia}(\text{Bentinho})]$

Deus dá nozes a quem não tem dentes.
 $\forall x \text{NãoTemDentes}(x) \rightarrow \text{DeusDaNozes}(x)$

Qualquer um que mata gatos é amado por ninguém.
 $\forall x \forall y \text{Matagatos}(x) \rightarrow \sim \text{Ama}(y, x)$

Alguém feriu o cachorro Bilu.
 $\exists x \text{Feriu}(x, \text{Bilu})$

Uma maçã caiu na cabeça de Newton. Newton foi um gênio.
 $\text{Caiu}(\text{maçã}, \text{Newton}) \wedge \text{Gênio}(\text{Newton})$

Somente um aluno teve aulas de tupi-guarani em 2003.
 $\exists! x \text{Aula}(\text{tupi}, x)$

Uma pessoa nascida fora da Inglaterra, com o pai nascido na Inglaterra, é

inglês por descendência.

$\forall x \text{ NascF}(x) \wedge \text{Pai}(x, y) \wedge \text{NascI}(y) \rightarrow \text{IngD}(y)$

Se a montanha não vem a Maomé, Maomé vai à montanha.

$\forall x \sim [\text{Vem}(x, \text{Maomé})] \wedge \text{Montanha}(x) \rightarrow \text{Vem}(\text{Maomé}, x)$

Um nascido na Inglaterra, que tem o pai cidadão inglês ou residente na Inglaterra é um cidadão inglês de nascença.

$\forall x \text{ NascidoIng}(x) \wedge \text{Pai}(x, y) \wedge (\text{NascidoIng}(y) \vee \text{Residente}(y)) \rightarrow \text{CidadaoIngNasc}(x)$

Quem nasce no Rio Grande do Sul ou na Argentina é gaúcho.

$\forall x \text{ Nasce}(x, \text{RS}) \vee \text{Nasce}(x, \text{Arg}) \rightarrow \text{Gaúcho}(x)$

O campeão da fórmula Indy é o único que tem o direito a usar o capacete branco

$\exists! x \text{ Campeão}(x) \rightarrow \text{CapaceteBranco}(x)$

Tomates são redondos. Laranjas são redondas. Quem é redondo, rola.

$\text{Redondo}(\text{Tomate}) \wedge \text{Redondo}(\text{Laranja}). \forall x \text{ Redondo}(x) \rightarrow \text{Rola}(x)$

O céu envia a carne, mas o diabo envia as cozinheiras.

$\text{CeUE}(\text{carne}) \wedge \text{DiaboE}(\text{cozinheiras})$

Alguns alunos tiveram aulas de francês em 2003. Outros tiveram aula de inglês no mesmo ano.

$\exists x \text{ Aulas2003}(\text{Frances}, x) \text{ land } \exists y \text{ Aulas2003}(\text{Ingles}, y)$

A César o que é de César.

$\forall x \text{ EdeC}(x) \rightarrow \text{DarC}(x)$

A melhor pontuação em tupi-guarani é sempre mais alta que a melhor pontuação em francês.

$x = \text{Melhor}(\text{tupi}) \text{ land } y = \text{Melhor}(\text{frances}) \rightarrow x > y$

Nenhuma pessoa compra um seguro caro

$\forall x \sim \text{CompraSeguroCaro}(x)$

Quem chegasse a Alexandria, era revistado.

$\forall x \text{ ChegaA}(x) \rightarrow \text{Revista}(x)$

Porque está no artigo 13 da Constituição, o Português é a língua oficial do País.

$\text{Artigo13Const}(\text{Brasil}) \rightarrow \text{LínguaOf}(\text{Brasil}, \text{Português})$

Janio Quadros renunciou. Quem renuncia ou é maluco, nunca mais se elege.

$\text{Renunciou}(\text{JQ}). \forall x \text{ Renunciou}(x) \vee \text{Maluco}(x) \rightarrow \sim \text{Elege}(x)$

Onde há fumaça, há fogo.

$\forall x \text{ HáFumaça}(x) \rightarrow \text{HáFogo}(X)$

Tancredo vence no Colégio eleitoral se e somente se o Exército estiver de acordo com a sua eleição.

$\text{Vence}(\text{Tancredo}) \Leftrightarrow \text{AcordoE}(\text{exército})$

Na Grécia, quando alguém não é entendido, afirma: "estou falando chinês"

$\forall x \forall y \text{ Está}(x, \text{Grécia}) \wedge \sim \text{Entende}(x, y) \rightarrow \text{FalandoChines}(x)$

A fila do lado sempre anda mais rápido.

$\forall x \text{ Filaalado}(x) \rightarrow \text{Andamais}(x)$

Um raio nunca cai duas vezes no mesmo lugar.

$\forall x \forall y \text{ Raio}(x) \wedge \text{Raio}(y) \rightarrow \sim \text{Local}(x, y)$

Ajoelhou, tem que rezar.

$\forall x \text{ Ajoelhou}(x) \rightarrow \text{Rezar}(x)$

Supondo os predicados Filho(x,y), Irmão(x,y), defina o predicado Tio(x,y).

$\text{Tio}(x, y) \rightarrow \text{Filho}(z, x) \wedge \text{Irmão}(z, y)$

Alegria de pobre dura pouco.

$\forall x \text{ Pobre}(x) \rightarrow \text{Alegria}(x, \text{pouco})$

Vocês que são brancos, que se entendam.

$\forall x \forall y \text{ Branco}(x) \wedge \text{Branco}(y) \rightarrow \text{Entedamse}(xy)$

Quem é gênio e é inglês faz grandes coisas.

$\forall x \text{ Gênio}(x) \wedge \text{Inglês}(x) \rightarrow \text{FazGC}(x)$

As más notícias chegam depressa.

$\forall x \text{ MáNotícia}(x) \rightarrow \text{ChegaDepressa}(x)$

Supondo definidos os predicados Ascendente(x,y) e Cônjuge(x,y) defina o predicado Nora(x,y).

$\text{Nora}(x, y) \rightarrow \text{Ascendente}(z, x) \wedge \text{Cônjuge}(z, y)$

Demonstração automática de teoremas

Como se viu a Lógica permite uma descrição bastante precisa do mundo ou pelo menos de um mundo em particular. O tema de estudo agora é como usar a lógica para recuperar e produzir conhecimento novo sobre este mundo. Vai-se pegar alguns conceitos emprestados da matemática. As premissas iniciais sobre o mundo vão receber o nome de axiomas. Este nome será usado tanto para fatos como para regras, desde que ambos sejam dados. A partir deles, ou baseado neles, vão ser feitas perguntas, que poderão ser respondidas como SIM ou NÃO. Tais perguntas vão receber o nome de TEOREMAS e vai-se estudar aqui o processo pelo qual a Lógica nos habilita a demonstrar um teorema. O significado disto é afirmar, ao final, que o teorema é válido ou seja que a resposta à pergunta que ele faz é SIM. Este exercício está preparando o caminho para o estudo da linguagem PROLOG, já que esta faz exatamente a mesma coisa que este exercício, só que de maneira automática. Grosso modo, o processo sempre é o mesmo e composto das seguintes etapas:

preparação

- 1. Reescrever as informações contidas no enunciado na forma de Lógica de Predicados. Utilizar apenas cláusulas de Horn (apenas uma cláusula após a implicação).
- 2. Eliminar $\rightarrow$, trocando-a por $\sim A \vee B$.
- 3. Reduzir o escopo de $\sim$, usando
 - (a) $\sim\sim A$ equivale a A
 - (b) $\sim (A \wedge B)$ equivale a $\sim A \vee \sim B$
 - (c) $\sim \forall x P(x)$ equivale a $\sim P(x)$
 - (d) $\sim \exists x P(x)$ equivale a $\forall x \sim P(x)$
- 4. Renomear as variáveis e passar os quantificadores para a esquerda
- 5. Retirar os quantificadores
- 6. Transformar as expressões em conjunção de disjuntos

demonstração

- 1. Acrescentar o que se quer provar NEGADO.
- 2. Localizar pares de afirmações contraditórias.
- 3. Se for possível efetuar substituições de modo que as afirmações contraditórias se refiram ao mesmo objeto, substituí-las e criar um novo teorema, onde as afirmações contraditórias são cortadas (simplificadas)
- 4. Prosseguir nesse ciclo até obter a cláusula NIHIL. Se isto foi possível o teorema original está demonstrado.
- 5. Se não for possível, não é possível demonstrar o teorema original.

A propósito vai-se estabelecer uma regra de escrita que será bem útil no PROLOG: variáveis serão escritas em maiúsculo e constantes em minúsculo.

Exemplo simples Seja o seguinte exemplo simples e manjado: *Todo homem é mortal. Sócrates é homem.* É possível afirmar que Sócrates é mortal ?

- 1. Transformando em predicados
 - 1. $\forall X, homem(X) \rightarrow mortal(X)$
 - 2. $homem(s)$.
- 2. Transformando o axioma 1, e já extraindo o quantificador $\forall$
 - 1. $\sim homem(X) \vee mortal(X)$.
- 3. Incluir a dúvida, negada
 - 3. $\sim mortal(s)$.
- 4. Os três axiomas agora são:
 - 1. $\sim homem(X) \vee mortal(X)$
 - 2. $homem(s)$
 - 3. $\sim mortal(s)$.
- 5. Examinando os axiomas verifica-se que substituindo X/s em 1 e juntando com 3, tem-se: $\sim homem(s) \vee mortal(s) \oplus \sim mortal(s)$
- 6. A existência de a e $\sim a$ permite a simplificação e o teorema 4 é
 - 4. $\sim homem(s)$.
- 7. Examinando agora o teorema 4 e o axioma 2, tem-se
 - $homem(s) \oplus \sim homem(s)$.
- 8. Desta simplificação não resta nada, ou melhor, resta NIHIL
- 9. Então está provado o teorema original *Sócrates é mortal*.

Mais um *Todo careca é feliz. Todo feliz é gordo. Zé é careca ou Zé está doente. Zé está são. Pode-se afirmar que Zé é gordo ?*
 $\forall X, careca(X) \rightarrow feliz(X)$
 $\forall Y, feliz(Y) \rightarrow gordo(Y)$
 $careca(z) \vee doente(z)$
 $são(z)$
Faltou usar uma informação do senso comum
 $são(K) \rightarrow \sim doente(K)$
Incluindo a fórmula negada,
 $\sim gordo(z)$
Bem resumido, e já devidamente operado:
1. $\sim careca(X) \vee feliz(X)$
2. $\sim feliz(Y) \vee gordo(Y)$
3. $\sim são(K) \vee \sim doente(K)$
4. $careca(z) \vee doente(z)$
5. $são(z)$
6. $\sim gordo(z)$
Substituindo Y/z e juntando 2 e 6, fica
7. $\sim feliz(z)$
Substituindo X/z e juntando 1 e 7, fica:
8. $\sim careca(z)$
Juntando 8 e 4 (sem substituição), fica
9. $doente(z)$
Juntando 3 e 9, fica
10. $\sim são(z)$
Juntando 10 e 5, fica
11. NIHIL
Está provada a afirmação de que Zé é gordo.

🔗 Para você fazer

A seguir, 10 textos em português, cada um deles com uma pergunta ao final. Responda 1 se a resposta for SIM e 0 se a resposta for não. Para seu governo a probabilidade da resposta ser sim é de 50 %. Idem para ser não.

- 1. quem determina as regras, tem que estudar muito/quem estabelece a pena, precisa ser justo/marcos não tem que estudar muito/marcos não estabelece a pena/é possível provar que marcos não determina as regras ?
- 2. quem gosta de Bach, tem bom gosto/quem gosta de programar, se sente uma máquina/ lucas gosta de programar ou gosta de Bach/lucas não gosta de programar/é possível provar que lucas tem bom gosto ?
- 3. quem ouve Beethoven, sabe o que é bom/quem come chucrute, tem bom gosto/sandro come chucrute ou ouve Beethoven/sandro não ouve Beethoven/é possível provar que sandro sabe o que é bom ?
- 4. quem gosta de Bach, tem bom gosto/quem tem bom gosto, sabe o que procurar/mario não sabe o que procurar/é possível provar que mario não gosta de Bach ?
- 5. quincas é corrupto e quincas ouve Beethoven/quem é corrupto, rouba a sociedade/ quem ouve Beethoven, sabe o que é bom/é possível provar que quincas rouba a sociedade ou quincas sabe o que é bom ?
- 6. deodoro é esperto e deodoro gosta de frio/quem é esperto, compra roupas novas/ quem gosta de frio, bebe vinho/é possível provar que deodoro compra roupas novas ou deodoro bebe vinho ?
- 7. pedro gosta de cachoeiras e pedro é bem humorado/quem gosta de cachoeiras, adora o verão/quem é bem humorado, leva a vida com leveza/é possível provar que pedro adora o verão ou pedro leva a vida com leveza ?
- 8. quem é corrupto, rouba a sociedade/quem come jujuba, usa roupa vermelha/ benedito come jujuba ou é corrupto/benedito não come jujuba/é possível provar que benedito rouba a sociedade ?
- 9. quem gosta de Bach, tem bom gosto/quem é gordo, gosta de passear/deodoro é gordo ou gosta de Bach/deodoro não gosta de Bach/é possível provar que deodoro tem bom gosto ?
- 10. quem gosta de Bach, tem bom gosto/quem sente frio, precisa roupas quentes/ sandro não tem bom gosto/sandro não sente frio/é possível provar que sandro não gosta de Bach ?

Não esqueça: 1 na resposta significa SIM e para responder NÃO, escreva 0.

1	2	3	4	5	6	7	8	9	10



Demonstração automática de teoremas

Para realizar este exercício, você deve:

1. Reescrever as informações contidas no enunciado na forma de Lógica de Predicados.
2. Eliminar $\rightarrow$, trocando-a por $\sim A \vee B$.
3. Reduzir o escopo de $\sim$, usando

(a) $\sim\sim A$ equivale a A
(b) $\sim (A \wedge B)$ equivale a $\sim A \vee \sim B$
4. Renomear as variáveis e passar os quantificadores para a esquerda
5. Retirar os quantificadores
6. Transformar as expressões em conjunção de disjuntos
7. Acrescentar o que se quer provar NEGADO.

Exemplo

Neste exemplo, seguir-se-á a convenção do PROLOG: variáveis em minúsculo e constantes em MAIÚSCULO. Seja o seguinte enunciado

1. Existem muitas frutas de diversas cores e sabores. A laranja é uma fruta. Todas as frutas são vegetais. Vegetais se dividem em duas categorias, os que fazem e os que não fazem mal à saúde. Os vegetais que não fazem mal à saúde são comestíveis. As laranjas são amarelas. Vegetais amarelos e comestíveis são gostosos.

Pergunta-se: Com esta base de informações é possível afirmar ser a laranja gostosa ?

Resolução:

Fatos:

1. Fruta(l)

2. $\forall X$ Fruta(X) $\rightarrow$ Vegetal(X)

3. $\forall X$ Vegetal(X) $\wedge$ Nãofazmal(X) $\rightarrow$ Comestível(X)

4. Amarelo(l)

5. $\forall X$ Vegetal(X) $\wedge$ Amarelo(X) $\wedge$ Comestível(X) $\rightarrow$ Gostoso(X)

Resolvendo:

6. $\sim$ Gostoso(l)

Juntando a 6 e a 5, e substituindo X/L , fica:

7. $\sim$ Gostoso(l) $\oplus \sim$ Vegetal(l) $\vee \sim$ Amarelo(l) $\vee \sim$ Comestível(l) $\vee$ Gostoso(l) e simplificando:

7. $\sim$ Vegetal(l) $\vee \sim$ Amarelo(l) $\vee \sim$ Comestível(l)

Juntando a 7 e a 4:

8. $\sim$ Vegetal(l) $\vee \sim$ Amarelo(l) $\vee \sim$ Comestível(l) $\oplus$ Amarelo(l) e simplificando:

8. $\sim$ Vegetal(l) $\vee \sim$ Comestível(l):

Juntando a 8 com a 2 e substituindo X/L , fica:

9. $\sim$ Vegetal(l) $\vee \sim$ Comestível(l) $\oplus \sim$ Fruta(l) $\vee$ Vegetal(l), simplificando:

9. $\sim$ Comestível(l) $\vee \sim$ Fruta(l)

Juntando a 9 e a 1,

10. $\sim$ Comestível(l) $\vee \sim$ Fruta(l) $\oplus$ Fruta(l), simplificando, fica:

11. $\sim$ Comestível(l)

Ao não haver mais nada a fazer, conclui-se que não é possível afirmar que a laranja é gostosa.

```
%exercicio 1
fruta(laranja).
vegetal(X):-fruta(X).
comestivel(X):-vegetal(X),naofazmal(X).
amarelo(laranja).
gostoso(X):-vegetal(X),amarelo(X),comestivel(X).
```

Ao rodar o programa acima no SWIprolog, com a pergunta "gostoso(laranja)", o compilador diz que naofazmal(X) está indefinido. Incluindo a cláusula "naofazmal(X):-comestivel(X).", o programa entra em loop. Daqui se conclui que não é possível provar o solicitado.

2. João gosta de todos os tipos de alimentos. maçãs são alimentos. galinha é um alimento. Qualquer coisa que qualquer um coma e não morra, é alimento. José come amendoim e ainda está vivo. Maria come tudo o que José come. pergunta: João gosta de amendoim ?

Fatos:

0. Comer e gostar são sinônimos

1. $\forall X$ Alimento(X) $\rightarrow$ Come,(João, X)

2. Alimento(maçã)

3. Alimento(galinha)

4. $\forall X \forall Y$ Come(Y, X) $\wedge \sim$ Morto(Y) $\rightarrow$ Alimento(X)

5. $\forall X$ Vivo(X) $\rightarrow \sim$ Morto(X)

6. Come(José,amendoim) $\wedge$ Vivo(José), separando, fica:

6a. Come(José,amendoim)

6b. Vivo(José)

7. $\forall X$ Come(José, X) $\rightarrow$ Come(Maria, X)

Resolvendo:

1. $\sim$ Alimento(X_1) $\vee$ Come,(João, X_1)

2. Alimento(maçã)

3. Alimento(galinha)

4. $\sim$ Come(y, X_2) $\vee$ Morto(Y) $\vee$ Alimento(X_2)

5. $\sim$ Vivo(X_3) $\vee \sim$ Morto(X_3)

6a. Come(José,amendoim)

6b. Vivo(José)

7. $\sim$ Come(José, X_4) $\vee$ Come(Maria, X_4)

8. $\sim$ Come(João,amendoim)

Juntando 1 e 8, e substituindo X_1 /amendoim, fica:

9. $\sim$ Alimento(a) $\vee$ Come,(João,a) $\oplus \sim$ Come(João,a) e simplificando

9. $\sim$ Alimento(a)

Juntando a 9 com a 4, e substituindo Y /José e X_2 /amendoim, fica:

10. $\sim$ Alimento(a) $\oplus \sim$ Come(José,a) $\vee$ Morto(José) $\vee$ Alimento(a) e simplificando

10. $\sim$ Come(José,a) $\vee$ Morto(José)

Juntando a 10 com a 5 e substituindo X_3 /José, fica:

11. $\sim$ Come(José,a) $\vee$ Morto(José) $\oplus \sim$ Vivo(José) $\vee \sim$ Morto(José) e simplificando

11. $\sim$ Come(José,a) $\vee \sim$ Vivo(José)

Juntando a 11 com a 6b, fica

12. $\sim$ Come(José,a) $\vee \sim$ Vivo(José) $\oplus$ Vivo(José), simplificando:

12. $\sim$ Come(José,a)

Juntando a 12 com 6a, fica

13. $\sim$ Come(José,a) $\oplus$ Come(José,a), que juntas se anulam e fica

14. NIL.

submetendo o programa

```
%exercicio 2
come(joao,X):-alimento(X).
alimento(maca).
alimento(galinha).
alimento(X):-come(Y,X),not(morre(Y)).
morre(X):-not(vivo(X)).
come(jose,amendoim).
vivo(jose).
come(maria,X):-come(jose,X).
ao prolog e perguntando: "come(jose,amendoim)"
ele responde true.
```

3. Os membros do clube de truco Seis Papudo são José, Sara, Bento e Elisa. José é casado com Sara. Bento é irmão de Elisa. O cônjuge de toda pessoa casada que é do clube também pertence ao Seis Papudo. A última reunião do clube foi na casa do José. Pergunta: A última reunião foi na casa de Sara ?

Afirmações:

1. Membro(José)

2. Membro(sara)

3. Membro(bento)

4. Membro(elisa)

5. Casado(Jose,sara)

6. $\forall X, Y$ Casado(X, Y) $\rightarrow$ Casado(Y, X)

7. $\forall X, Y$ Membro (X) $\wedge$ Casado(X, Y) $\rightarrow$ Membro(Y)

8. Irmão(bento,elisa)

9. Irmão(elisa,bento)

10. $\forall X, Y$ Casado(X, Y) $\wedge$ Reuniãofoi(X) $\rightarrow$ Reuniãofoi(Y)

11. Reuniãofoi(Jose)

Arrumando:

1. Membro(José)

2. Membro(sara)

3. Membro(bento)

4. Membro(elisa)

5. Casado(Jose,sara)

6. $\sim$ Casado(X_1, Y_1) $\vee$ Casado(Y_1, X_1)

7. $\sim$ Membro (X_2) $\vee \sim$ Casado(X_2, Y_2) $\vee$

Membro(Y_2)

8. Irmão(bento,elisa)

9. Irmão(elisa,bento)

10. $\sim$ Casado(X_3, Y_3) $\vee \sim$ Reuniãofoi(X_3) $\vee$ Reuniãofoi(Y_3)

11. Reuniãofoi(José)

Buscando a prova

12. $\sim$ Reuniãofoi(sara)

Juntando a 12 com a 10 e substituindo X_3 /José e Y_3 /sara, fica

13. $\sim$ Reuniãofoi(sara) $\oplus \sim$ Casado(José,sara) $\vee \sim$ Reuniãofoi(José) $\vee$ Reuniãofoi(sara), simplificando fica

13. $\sim$ Casado(José,sara) $\vee \sim$ Reuniãofoi(José)

Juntando a 13 com a 11, fica

14. $\sim$ Casado(José,sara) $\vee \sim$ Reuniãofoi(José) $\oplus$ Reuniãofoi(José), simplificando

14. $\sim$ Casado(José,sara)

Juntando a 14 com a 5, fica

15. $\sim$ Casado(José,sara) $\oplus$ Casado(Jose,sara), simplificando

16. NIL

```
%exercicio 3
membro(jose).
membro(sara).
membro(bento).
membro(elisa).
casado(jose,sara).
casado(X,Y):-casado(Y,X).
membro(X):-membro(Y),casado(X,Y).
irmao(bento,elisa).
irmao(elisa,bento).
reuniao(jose).
reuniao(Y):-reuniao(X),casado(X,Y).
```

à pergunta "reuniao(sara)." o prolog respondeu true.

🔗 Para você fazer

Nos exercícios a seguir, verifique se a pergunta feita pode ou não pode ser afirmada, baseado no enunciado e nas regras da lógica.

Enunciado 1

Juca gosta de fazer apenas matérias fáceis. As matérias da engenharia da computação são difíceis. Todas as matérias do curso de Bruxaria são fáceis. A matéria de Encantamentos e Poções é do curso de Bruxaria. Estrutura de dados é matéria da Engenharia da Computação. Pergunta: possível afirmar que Juca gostaria de fazer Encantamentos e Poções ?

Enunciado 2

Juca gosta de fazer apenas matérias fáceis. As matérias da engenharia da computação são difíceis. Todas as matérias do curso de Bruxaria são fáceis. A matéria de Encantamentos e Poções é do curso de Bruxaria. Estrutura de dados é matéria da Engenharia da Computação. Pergunta: possível afirmar que Juca não gostaria de fazer Estrutura de Dados ?

Resposta

Faça uma marca nos enunciados que PUDERAM ser provados:

1	2



Lógica e Teoria dos Conjuntos

Grande progresso é conseguido ao se estudar uma ciência quando se podem fazer analogias e estabelecer pontes entre diversos ramos dessa ciência. Tais analogias facilitam o entendimento e lançam luzes sobre algo desconhecido. Há inúmeros exemplos conhecidos, um dos últimos é da última década do século 20 quando o matemático Andrew Wiles demonstrou o último Teorema de Fermat ($x^n + y^n \neq z^n, \forall n > 2$ com x, y e z inteiros) usando a conjectura Taniyama-Shimura. Esta conjectura (agora teorema) faz uma analogia entre equações elípticas e formas modulares. Ela acabou se revelando chave na demonstração.

Com a palavra Simon Singh, em o último teorema de Fermat

O valor das pontes matemáticas é enorme. Elas permitem que as comunidades de matemáticos que viviam em ilhas separadas troquem idéias e explorem suas criações. A matemática consiste em ilhas de conhecimento em um mar de ignorância. Por exemplo, existe uma ilha ocupada pelos geométricas, que estudam as formas e existe a ilha dos probabilistas que estudam o risco e o acaso. Existem dúzias de tais ilhas, cada uma com a sua linguagem única, incompreensível para os habitantes das outras ilhas.

O objetivo deste exercício é construir uma analogia deste tipo entre a lógica proposicional e a teoria dos conjuntos.

As idéias básicas da Teoria dos Conjuntos foram desenvolvidas pelo matemático alemão Georg Cantor (1845-1918) por volta de 1875. A primeira dificuldade é definir conjunto. Não é possível fazer isso em termos matemáticos, razão pela qual ela fica sendo uma noção não definível: um conjunto é uma coleção de coisas. Por exemplo, o conjunto dos naturais: $\{1, 2, 3, \dots\}$. Note que as reticências ao final da lista indicam que o conjunto é infinito. Por exemplo, o conjunto das vogais no Português é $\{a, e, i, o, u\}$.

O número de elementos em um conjunto é conhecido como a *cardinalidade* do conjunto. Assim, a cardinalidade dos naturais é ∞ enquanto que a cardinalidade das vogais é 5.

Define-se um conjunto chamado conjunto vazio que é aquele cuja cardinalidade é zero. Ele é representado por ϕ .

A relação de pertinência de um elemento a um conjunto é estabelecida pela operação de *pertença* cujo símbolo é $\in$. (Ou seu oposto $\notin$).

Por exemplo, $a \in \{a, b, c, d, e, f, g\}$. $E, u \notin \{a, b, c, d, e, f, g\}$.

Dois conjuntos, digamos A e B são iguais quando duas condições são verdadeiras:
* todo elemento de A também é elemento de B e
* todo elemento de B também é elemento de A .

Assim, pode-se afirmar que:
 $\{a, b, c, d\}$ é igual a $\{a, b, c, d\}$ ou
 $\{a, b, c, d\}$ é igual a $\{b, d, a, c\}$ ou
 $\{a, b, c, d\}$ é igual a $\{b, a, d, c\}$

Razão pela qual costumam-se desprezar as repetições de elementos ao descrever um conjunto.

Nesta teoria define-se o conjunto Universo (identificado por U) que é o conjunto de todos os possíveis elementos. Naturalmente o conjunto U muda convenientemente a cada problema tratado. Esta idéia foi levada adiante pelo lógico John Venn (1834-1923) que desenvolveu os Diagramas de Venn, nos quais o conjunto U é a totalidade do desenho.

Com este conceito é possível definir o conjunto *complemento* de um conjunto dado. Assim, o complemento de um conjunto A é o conjunto de todos os elementos de U que não pertencem a A .

Representa-se o complemento de A por A' e escreve-se $A' = \{x|x \in U \text{ e } x \notin A\}$.

Define-se um subconjunto S de A quando todo elemento de S pertence a A . Escreve-se $S \subseteq A$ neste caso. Note que esta definição é verdadeira quando $S = A$. Se quisermos eliminar esta possibilidade, usa-se o símbolo $S \subset A$ e denomina-se S como subconjunto próprio de A . Em símbolos $S \subset A$, se $S \subseteq A$ e $S \neq A$.

Uma relação importante na nossa profissão, que vai aparecer muiiiiitas vezes daqui para a frente

é: O número de subconjuntos de um conjunto com n elementos é 2^n . O número de subconjuntos próprios de um conjunto com n elementos é $2^n - 1$.

Operações de conjuntos A primeira é a operação de intersecção de dois conjuntos, representada por $\cap$, e assim definida: $A \cap B = \{x|x \in A \text{ e } x \in B\}$. Note que a intersecção de dois conjunto é também um conjunto. Quando a intersecção de dois conjuntos é ϕ pode-se afirmar que eles não tem elemento comum.

Outra operação é a união de dois conjuntos, denotada por $\cup$ e assim definida: $A \cup B = \{x|x \in A \text{ ou } x \in B\}$. Quando o conjunto vazio é unido a outro conjunto o resultado é igual ao outro conjunto.

Uma terceira é a diferença entre dois conjuntos, denotada por $-$. $A - B = \{x|x \in A \text{ e } x \notin B\}$.

Uma quarta é o produto cartesiano de dois conjuntos, digamos A e B que se escreve como $A \times B$ e é $A \times B = \{(a, b)|a \in A \text{ e } b \in B\}$.

Aqui surgem as Leis de De Morgan, a saber:
 $(A \cap B)' = A' \cup B'$ e
 $(A \cup B)' = A' \cap B'$

Finalmente, as operações de $\cap$ e $\cup$ obedecem às propriedades comutativa, associativa, distributiva e de elemento identidade. Neste caso, estas operações formam o que na teoria algébrica se conhece como anél.

Lógica

A analogia que se pode fazer aqui é entre as operações lógicas e as operações entre conjuntos. Assim, suponhamos a existência de dois conjuntos, A e B . Ao definir o conjunto dos elementos x que formam a intersecção destes conjuntos, $x \in A \cap B$, pode-se dizer que o elemento x é aquele que torna verdadeira a proposição $x \in A \wedge x \in B$.

Da mesma maneira ao definir a união, $x \in A \cup B$ tem-se que x torna verdadeira a proposição $x \in A \vee x \in B$.

Idem para a operação de complemento, $x \in A'$ se $\sim x \in A$.

Em resumo:

conjuntos	lógica
$A \cap B$	$a \wedge b$
$A \cup B$	$a \vee b$
A'	$\sim a$

Exemplo Feito

No exercício a seguir, vão ser definidos 4 conjuntos. Cada um deles é uma Progressão Aritmética. A saber:

$A = PA, a_1 = 2, r_a = 5$ e $n_a = 7$.
 $B = PA, b_1 = -3, r_b = -2$ e $n_b = 9$.
 $C = PA, c_1 = -2, r_c = -3$ e $n_c = 6$.
 $D = PA, d_1 = 3, r_d = 4$ e $n_d = 4$.

Com isto, as PAs são:
 $A = \{2, 7, 12, 17, 22, 27, 32\}$
 $B = \{-3, -5, -7, -9, -11, -13, -15, -17, -19\}$
 $C = \{-2, -5, -8, -11, -14, -17\}$
 $D = \{3, 7, 11, 15\}$

As perguntas a responder são as mesmas do seu exercício e são:

a) o sexto elemento do primeiro conjunto ($A \cup B$); b) a soma dos elementos de $A \cap (B \cup C)$; c) a cardinalidade de $B \cap C'$ e d) o maior elemento de $C \cup D \cup A$.

Os conjuntos são:

$P = \{2, 7, 12, 17, 22, 27, 32, -3, -5, -7, -9, -11, -13, -15, -17 \text{ e } -19\}$, que devidamente ordenado em ordem crescente fica: $\{-19, -17, -15, -13, -11, -9, -7, -5, -3, 2, 7, 12, 17, 22, 27, 32\}$. E, o sexto elemento é -9 .

$Q =$ vazio, logo a soma dos elementos que o compõe é zero.

$R = \{-3, -7, -9, -13, -15 \text{ e } -19\}$ cuja cardinalidade é 6.

$S = \{-2, -5, -8, -11, -14, -17, 3, 7, 11, 15, 2, 12, 17, 22, 27 \text{ e } 32\}$, cujo maior elemento é 32.

Com isso, as respostas do exemplo são: $\{-9, 0, 6 \text{ e } 32\}$.

Para você fazer

1. No exercício a seguir, vão ser definidos 4 conjuntos. Cada um deles é uma Progressão Aritmética. A saber: A é uma P.A. onde $a_1 = -3, r_a = -4, n_a = 4$.
 B idem, onde $b_1 = -3, r_b = 2, n_b = 9$.
 C idem, onde $c_1 = 3, r_c = 5, n_c = 5$.
 D idem, onde $d_1 = 3, r_d = -2, n_d = 7$.

Para resolver o exercício, você deve construir 4 conjuntos, P, Q, R e S , a partir de

conj	formação	em lógica
P	$A \cup B$	$p \in A \vee p \in B$
Q	$A \cap (B \cup C)$	$q \in A \wedge (q \in B \vee q \in C)$
R	$B \cap C'$	$r \in B \wedge (\sim r \in C)$
S	$C \cup D \cup A$	$(s \in C) \vee (s \in D) \vee (s \in A)$

Tendo calculado os quatro conjuntos, responda às perguntas

1	Ordene em ordem crescente o conjunto P e informe o sexto elemento	
2	Some os valores do conjunto Q	
3	Ache a cardinalidade do conjunto R	
4	O maior elemento do conjunto S é	

2. No exercício a seguir, vão ser definidos 4 conjuntos. Cada um deles é uma Progressão Aritmética. A saber: A é uma P.A. onde $a_1 = 2, r_a = 2, n_a = 8$.
 B idem, onde $b_1 = 2, r_b = 6, n_b = 5$.
 C idem, onde $c_1 = 5, r_c = 3, n_c = 5$.
 D idem, onde $d_1 = 3, r_d = 3, n_d = 9$.

Para resolver o exercício, você deve construir os mesmos 4 conjuntos do exercício anterior.

Tendo calculado os quatro conjuntos, responda às perguntas

5	Ordene em ordem crescente o conjunto P e informe o sexto elemento	
6	Some os valores do conjunto Q	
7	Ache a cardinalidade do conjunto R	
8	O maior elemento do conjunto S é	

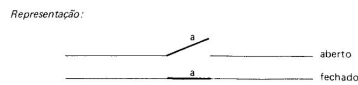


Álgebra de Boole

Em 1854, o inglês George Boole escreveu *An investigation of the laws of thought*, e nele apresentou um sistema matemático de análise lógica conhecido como álgebra booleana. Era o início da *matematização* da lógica. Até então, a lógica pertencera ao reino da filosofia, e agora ela iniciava sua caminhada em direção à matemática, cujo auge seriam os livros de Bertrand Russell e Whitehead, *The Principles of Mathematics*, no qual ambos tentam provar que matemática = lógica. Não conseguiram e levaram o tiro de misericórdia quando por volta de 1930, o tcheco Kurt Goedel publicou o seu teorema da incompletude.

Em 1938, o americano Claude Shannon usou as idéias de Boole para construir circuitos de telefonia com relés, praticamente criando o campo da eletrônica digital. Este ramo da eletrônica emprega em seus sistemas um pequeno grupo de circuitos básicos padronizados conhecidos como portas lógicas.

Começemos com o conceito de interruptor:

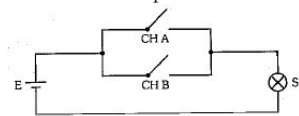


Se considerarmos a corrente elétrica fluindo pelo circuito, pode-se associar a passagem da corrente a um valor lógico (tipicamente V), ou mais adequadamente ao valor 1. Opostamente, a não passagem da corrente, pode ser associada a F, ou neste caso ao valor 0.

Acompanhe-se a seguinte transposição:

fenômeno	lógica clás.	álg.booleana
inter. aberto	F	0
inter. fechado	V	1

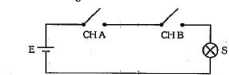
Imagine-se agora 2 interruptores ligados em paralelo. Nesta a corrente vai passar se pelo menos um dos interruptores (ou até os 2) estiverem fechados. Acompanhe



Tal fato é representado na álgebra de boole pela expressão $a + b$ sendo a e b os interruptores que estão ligados em paralelo. Salta aos olhos que está-se falando na proposição lógica $a \vee b$. Se se pretender escrever uma tabela verdade para este circuito, vai aparecer aquela nossa velha conhecida:

a	b	$a + b$
1	1	1
1	0	1
0	1	1
0	0	0

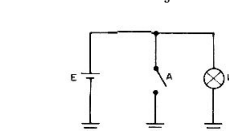
Imagine-se agora 2 interruptores ligados em série, um depois do outro. Neste caso a corrente só vai passar pelo circuito se ambos estiverem fechados. Veja o desenho:



Na álgebra de boole, este fato é representado por $a.b$, sendo a e b os interruptores. Não é coincidência, que na lógica este é o circuito $\wedge$, que representa a expressão $a \wedge b$. Olhemos a tabela verdade:

a	b	$a.b$
1	1	1
1	0	0
0	1	0
0	0	0

Agora, vai-se apresentar o circuito que implementa a função não. A corrente passa quando o circuito está aberto e não passa quando o circuito está fechado.. Veja

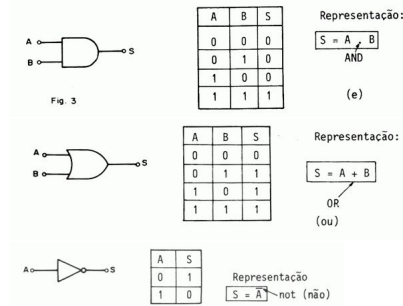


Note que a operação não da lógica é representada na álgebra de boole como $\bar{a}$ para o circuito a .

Note-se que alguns livros, com dificuldade em representar $\bar{a}$ contentam-se em usar a representação a' para a negação. Sua tabela verdade

a	$\bar{a}$ ou a'
1	0
0	1

Em termos de representação em portas lógicas, as convenções são:



Álgebra de Boole

A teoria de boole está fundamentada em 7 postulados a saber:

Postulado	descrição
1	$x=0$ ou $x=1$
2	$0.0=0$
3	$1.1=1$
4	$0+0=0$
5	$1+1=1$
6	$1.0=0.1=0$
7	$1+0=0+1=1$

Baseado nestes postulados, diversos teoremas podem ser apresentados:

N.	Lei	báse	dual
1	Comutativa	$A+B = B+A$	$A.B = B.A$
2	Associativa	$(A+B)+C = A+(B+C)$	$(A.B).C = A.(B.C)$
3	Distributiva	$A.(B+C) = A.B + A.C$	$A+(B.C) = (A+B).(A+C)$
4	Identidade	$A+A = A$	$A.A = A$
5	Negação	$\bar{\bar{A}} = A$	$\bar{\bar{A}} = A$
6	Redundância	$A+A.B = A$	$A.(A+B) = A$
7.1	Elemento neutro	$0+A = A$	$1.A = A$
7.2	Elemento absorvente	$1+A = 1$	$0.A = 0$
8	operação com oposto	$\bar{A}+A = 1$	$\bar{A}.A = 0$
9		$A+\bar{A}.B = A+B$	$A.(\bar{A}+B) = A.B$
10	De Morgan	$\overline{A+B} = \bar{A}.\bar{B}$	$\overline{A.B} = \bar{A}+\bar{B}$

Os teoremas 1,2 e 3 mostram que as leis básicas da álgebra se aplicam à álgebra binária. A lei da negação só se aplica à álgebra booleana. A lei da redundância pode ser comprovada como:

$A+A.B=A$, colocando o A em evidencia

$A.(1+B)=A$

$A.1=A$

$A=A$, ou seu dual

$A.(A+B)=A$

$A.A+A.B=A$

$A+A.B=A$

$A.(1+B)=A$

$A.1=A$

$A=A$

Os teoremas 7.1, 7.2 e 8 são teoremas da álgebra booleana. O T9 pode ser demonstrado como $A+\bar{A}.B=A+B$, expandindo

$(A+\bar{A}).(A+B)=A+B$

$1.(A+B)=A+B$

$A+B=A+B$

O T10 é o nosso já conhecido De Morgan.

Note-se que estas manipulações algébricas são muito indicadas quando se pretende simplificar um circuito, ou seja construir um circuito que tenha o mesmo comportamento mas que tenha menos componentes (já que isto significa menos custos e menos complexidade). Outra maneira de obter tal simplificação é através dos mapas de Veitch-Karnaugh.

Para você fazer

- No primeiro exercício você vai receber 3 expressões lógicas e deve desenhar o circuito elétrico que vai atendê-lo. O número de entradas varia em cada caso (conforme a expressão), mas a saída só tem dois valores (0 ou 1). Desenhe os circuitos em uma folha anexa e coloque nela seu nome e o código 1.

J+M

$(A.L)+M$

$(G.L)+(B.F)$

- No segundo exercício, você também vai receber 3 expressões lógicas. Construa as tabelas verdade para cada expressão. Entregue o resultado na mesma folha acima.

$(J.K)'$

$(C+D).L$

$((D.F)') . ((J+K)')$

- No terceiro exercício, você vai receber a descrição de um problema e deve pensar em um circuito elétrico que o resolva. Especial atenção às entradas (quantas e quais) e às saídas (quantas e quais, também). Desenhe a tabela verdade para o problema. Se quiser, desenhe o circuito também. Em uma obra, existe uma betoneira super inteligente que atende a 3 andares próximos. Ela opera segundo a regra: o andar mais baixo sempre é atendido primeiro.



- 1 - /

Lista de pedidos 106

1. Existe uma cidade distante que se chama VIVENDOPERIGOSAMENTE. Suponha um semáforo de 2 ruas que se cruzam, sujeito às seguintes regras: Não havendo carro em A, nem havendo carro em B, ambas as luzes vermelhas devem ficar acesas. Havendo carro em A e não em B, verde para A e vermelho para B. Havendo carro em B e não havendo em A, acenda verde para B e vermelho para A. Havendo carro em ambos, acenda o verde para ambos. Viu-se porquê do nome da cidade ?
2. Em uma obra, existe uma betoneira super inteligente que atende a 3 andares próximos. Ela opera segundo a regra: o andar mais baixo sempre é atendido primeiro.
3. Uma escola modernosa bolou uma sala de aula bacana. Nela há 3 interruptores: na porta de entrada; na mesa do professor e na sala do diretor. Se dois interruptores estiverem ligados, as luzes devem estar acesas. Se o diretor ligar o seu, as luzes devem ser acesas. Em qualquer outro caso, as luzes devem ficar apagadas.
4. Um barzinho da moda inovou na forma de chamar o garçon. O barzinho só tem 2 mesas, e em cada mesa dois botões: um botão para homens e outro para mulheres. Se apenas uma mesa (qualquer botão) for apertado o garçon deve atendê-la. Se ambas mesas forem acionadas, vale a regra Ambos homens: atenda a mesa 1; homem e mulher: atenda a mesa da mulher. Ambas mulheres: atenda a mesa 2. Se 3 botões são acionados atenda a mesa que mais botões apertou. Se os 4 botões forem apertados atenda a mesa 2. Finalmente, se nenhum botão é apertado atenda a mesa 1.
5. Uma universidade à frente do seu tempo determinou uma estratégia interessante para reservar e usar os laboratórios de informática. Há 4 professores em cada turno: (primeiro ano, segundo ano,...) Só há um laboratório. Se apenas um professor quiser usar o laboratório será dele. Se dois ou mais quiserem usar, o laboratório o laboratório será do menor ano que o pediu, EXCETO se for o profe do quarto ano. Neste caso, o laboratório é dele.
6. Uma cidade contruiu um estádio para uso de 3 times que há lá. Cada time tem uma cor: o time A tem a cor azul, o time B tem a cor branca e o time C tem a cor laranja. Você deve iluminar adequadamente o estádio sujeito às regras: Se A, B ou C jogarem contra outro time, a cor deve ser a de A, B ou C. Se eles jogarem entre sí, vale a ordem alfabética reversa: Primeiro a cor de C, depois a de B e depois a de A.
7. Um elevador pode ser chamado de 3 andares, mas não do andar onde ele está atualmente. Se só um andar chamar, deve atendê-lo. Se dois andares chamarem deve atender ao mais próximo, e se houver empate deve atender ao mais alto.
8. Tres diretores em uma empresa compartilham os serviços de uma secretária: o presidente, o vice e o diretor de logística. Cada um deles tem um campainha para chamá-la. Indique para a secretária qual diretor deve atender, baseado nas regras: Se só um diretor chamar, deve ser atendido. Se mais de um chamar, deve-se observar a hierarquia.
9. Uma cidade chamada S AZAR, está sujeita a inundações, raios e e huaycos (um termo quechua que indica que o morro vem abaixo carregando água e barro e arrastando tudo na sua passagem). Há 4 alarmes: I, R, H e F. Este último chamado F sugere FUJA CORRENDO LOGO !. Havendo apenas um risco, deve soar o o alarme correspondente. Havendo dois riscos deve se observar a prioridade: H, I e R. Finalmente, havendo os 3 riscos deve tocar o alarme F.

Outras lógicas

Até aqui houve o compromisso epistemológico ¹ de que as proposições são verdadeiras, falsas ou desconhecidas.

Entretanto, esta estratégia é falha pelo menos por 3 razões:

- preguiça: é trabalhoso demais listar o conjunto completo de antecedentes ou consequentes para assegurar regras sem exceção. É muito difícil usar tais regras.
- ignorância teórica: a ciência nem sempre tem a teoria completa para o domínio.
- ignorância prática: ainda que toda a teoria seja conhecida, em um caso prático real nem todos os testes podem ou foram executados.

A principal ferramenta para lidar com a crença é teoria da probabilidade que atribui a cada sentença um grau numérico de crença variando entre 0 e 1.

0 significa que a crença é de que a frase é falsa. 1 significa crença inequívoca de que a frase é verdadeira. Vale diferenciar grau de crença com grau de verdade. A probabilidade de 0,8 não significa que a frase é 80% verdadeira e sim que a frase tem uma crença de 80%. A teoria da probabilidade continua com o compromisso ontológico ² assumido pelo lógica : os fatos são ou não são válidos no mundo. O grau de verdade é assunto da lógica difusa.

Os graus de crença sempre se aplicam a proposições. O elemento básico em tais proposições é a variável aleatória, com um domínio específico que pode ser booleano, discreto ou contínuo.

A probabilidade a priori é o grau de crença para a proposição na ausência de outras informações. Representada como $P(a)$. Às vezes queremos nos referir às probabilidades de todos os valores de uma variável aleatória. A expressão é $P(a)$.

A probabilidade condicional é $P(a|b)$ onde a e b são proposições quaisquer. Ela é lida como a probabilidade de a dado que sabemos b . Passa-se de uma a outra através da fórmula

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

Probabilidades tem 3 axiomas

1. Toda probabilidade está entre 0 e 1
2. probabilidades verdadeiras são 1 e falsas são 0
3. A probabilidade de uma disjunção é

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

Lógica Probabilística

- A probabilidade é a maneira correta de raciocinar sobre a incerteza
- A incerteza surge como consequência da preguiça e da ignorância. Ela é inevitável em mundos complexos, dinâmicos ou inacessíveis.
- a incerteza significa que muitas das simplificações que são possíveis no caso da inferência dedutiva não são mais válidas
- As probabilidades expressam a inabilidade do agente para alcançar uma decisão definida com relação à verdade de uma sentença. As probabilidades resumem a crença de um agente.
- Declarações básicas de probabilidade incluem probabilidades a priori e probabilidades condicionais sobre proposições simples e complexas.

- A regra de Bayes permite que probabilidades desconhecidas sejam calculadas a partir de probabilidades condicionais conhecidas em geral no sentido causal. Normalmente a aplicação da regra de Bayes com muitas peças de evidência resultará nos mesmos problemas de ampliação da escala que encontramos na distribuição conjunta total.

Lógica Nebulosa, ou fuzzy

A lógica é a ciência que tem por objetivo as leis do raciocínio. A lógica nebulosa estuda os princípios formais do raciocínio aproximado.

Na lógica clássica (aristotética) há dois valores verdade possível: verdadeiro e falso. São as lógicas bivalentes. No modelo bayesiano e em outros modelos probabilísticos também há bivalência: cada fato é verdadeiro ou falso, mas muitas vezes não fica claro qual deles vale. A probabilidade informa a plausibilidade de uma dada proposição ser verdadeira.

Surgiu uma lógica polivalente para raciocinar sobre o princípio da incerteza: os valores eram V, F e indeterminado. Uma extensão desta lógica é $1=V, 0=F$ e $n, 0 < n < 1$ para representar graus de verdade.

Variáveis linguísticas são um conceito como altura, que pode ser alto, médio e baixo. A variável linguística altura pode ser definida como universo do discurso desde 0,6m a 2,4m.

A lógica nebulosa é usada para raciocinar sobre conjuntos nebulosos. Os conjuntos tradicionais são conhecidos como nítidos (*crisp sets*). Um conjunto nítido é descrito pelos elementos que fazem parte dele. Um valor está ou não está em um conjunto nítido.

Já na área nebulosa, vamos considerar o conjunto de pessoas altas. Bill com 2,1m certamente está nele. John, de 1,2m certamente não está. Mas, e Jane que tem 1,78m ? O conjunto nebuloso de pessoas altas inclui Bill, Jane e até mesmo John. Cada um é membro do conjunto até certo ponto e não é até certo ponto também.

Lá atrás definiu-se a lei do terceiro excluído, perfeitamente válida na lógica aristotética. Aqui na lógica nebulosa, ele não é verdadeiro. Também a regra $A \vee \sim A = V$ e a $A \wedge \sim A = F$ na lógica nebulosa não valem.

Um conjunto nebuloso é definido pela sua função de pertinência. Por exemplo um conjunto Bebê (B), pode ser definido $P_b(x) = 1 - \frac{x}{5}$ para $x \leq 2$ e 0 para $x > 2$ onde x é a idade em anos. Não há nada de especial sobre estas funções. Elas foram uma escolha subjetiva do autor.

Para representar um conjunto nebuloso em um computador usa-se uma lista de pares. Cada par representa um valor e o valor nebuloso de pertinência para aquele valor. Para definir B acima poder-se-ia ter $B = \{(0, 1), (2, 0)\}$. Isto pode ser considerado como representando coordenadas x e y de dois pontos na linha que representa a função de pertinência do conjunto.

A teoria de conjunto tradicional (devida a Cantor) usa operadores que se aplicam a conjuntos A e B. Tem-se: $\neg A$ complemento de A, $A \cap B$ interseção de A e B, $A \cup B$, união de A e B. Podemos considerá-los como relacionados aos operadores $\sim, \wedge$ e $\vee$. Como esperados os operadores de conjunto são comutativos, associativos e distributivos e também obedecem à lei de Morgan.

Pode-se definir operadores semelhantes para conjuntos nebulosos. O complemento do conjunto nebuloso A é $P_{\neg A}(x) = 1 - P_A(x)$. Usando esta definição ao conjunto de bebês (não bebês, na verdade) ficaria: $\neg B = \{(0, 0), (2, 1)\}$.

A interseção de dois conjuntos nebulosos B e C é o mínimo das funções nebulosas para os conjuntos. Ou $P_{A \cap B}(x) = \min(P_A(x), P_B(x))$. Por exemplo, se C (criança) for definido como $C = \{(1, 0), (7, 1), (8, 1), (14, 0)\}$, a interseção de B com C fazemos:

Antes é preciso equalizar B e C para os mesmos valores, fica: $B = \{(0, 1), (1, 0.5), (2, 0), (7, 0), (8, 0), (14, 0)$ e $C = \{(0, 0), (1, 0), (2, 0.16), (7, 1), (8, 1), (14, 0)$ a agora pode-se encontrar a interseção: $B \cap C = \{(0, 0), (1, 0), (2, 0), (7, 0), (8, 0), (14, 0)\}$. Mas isto não funcionou. Precisamos definir o conjunto usando valores que definirão corretamente as faixas: então $B \cap C = \{(1, 0), (1.75, 1.25), (2, 0)\}$. 1.75

foi usado como valor para x e foi determinado pelo cálculo do valor de x para o qual $P_B(x) = P_C(x)$.

A interseção nebulosa significa que seus elementos pertencem aos dois conjuntos.

A união de dois conjuntos A e B é definida como $P_{A \cup B}(x) = \max(P_A(x), P_B(x))$. A união nebulosa de crianças e bebês fica $B \cup C = \{(0, 1), (1.75, 0.25), (7, 1), (8, 1), (14, 1)\}$.

Há ainda o operador nebuloso inclusão. Na teoria tradicional, se A contiver B isto significa que todos os elementos de B estarão em A. E, $A \cup B = A$ e $A \cap B = B$. Neste caso, B é subconjunto de A e se escreve $A \subset B$.

Na lógica nebulosa a inclusão é $B \subset A$ sse $\forall x(P_B(x) \leq P_A(x))$.

Em lógica nebulosa usam-se modificadores como muito, razoavelmente, extremamente ou pouco. Pode-se aplicar o modificador elevando a função de pertinência ao conjunto a uma potência apropriada, usando a tabela:

muito	eleva a 2
razoavelmente	1,3
pouco	0.5
extremamente	4

Lógica nebulosa

Uma lógica que se aplica a variáveis nebulosas. É não monotônica, no sentido de que se um novo fato for conhecido ele poderá contradizer conclusões previamente derivadas.

As funções MAX e MIN são usadas para calcular a disjunção ou conjunção de duas variáveis nebulosas. Uma variável nebulosa valendo 0.5 pode ser pouco verdadeiro ou tanto verdadeiro quanto falso.

As leis lógicas agora são: $A \vee B \equiv \max(A, B)$ e $A \wedge B \equiv \min(A, B)$ e $\sim A \equiv 1 - A$.

Obviamente não se pode construir uma tabela verdade para uma expressão lógica nebulosa: as entradas seriam infinitas. Mas, pode-se fazê-lo para um conjunto finito de valores lógicos, por exemplo:

A	B	$A \vee B$ (max)
0	0	0
0	0.5	0.5
0	1	1
0.5	0	0.5
0.5	0.5	0.5
0.5	1	1
1	0	1
1	0.5	1
1	1	1

A implicação ($\rightarrow$), que lá na lógica clássica era $A \rightarrow B \equiv \sim A \vee B$, aqui é $A \rightarrow B \equiv \max(\sim A, B)$.

Dai os livros definem uma Implicação de Godel, como $A \rightarrow B \equiv (A \leq B) \vee B$. Definem também a regra do modus ponens.

Para você fazer

Faça uma avaliação comparativa entre as 3 lógicas estudadas (clássica, probabilística e nebulosa), quanto aos seguintes aspectos:

1. adequação aos problemas do mundo real
2. utilidade
3. facilidade de manuseio das ferramentas da lógica considerada
4. grau de sucesso



¹A epistemologia é um ramo da filosofia que estuda a natureza, a origem e a validade do conhecimento. Ela é frequentemente definida como a teoria do conhecimento

²a ontologia estuda a classificação das entidades

Indexação e indireção

Observação: Origem dos índices é 0 (ZERO)

Vetores e “for” O comando *for* é muito adequado para lidar com vetores, já que neste caso, sempre se conhecem os limites inferior e superior da variação. Acompanhe nos exemplos:

Exemplo 1:

```
int V[10];
int I;
for (I=0;I<10;I++) {
    V[I] = I * 10; }
cout<< V[2]+V[3]+V[4];
```

Resposta: _____

Exemplo 2:

```
float V[30];
int I;
for (I=0;I<30;I++) {
    V[I] = I+1; }
for (I=2;I<30;I++) {
    V[I] = V[I-1]+V[I]; }
cout<< (V[19]+V[20]+V[21]);
```

Resposta: _____

Exemplo 3:

```
int V[20];
float W[10];
int I;
for(I=0;I<20;I++) {
    V[I] = 21-I; }
for (I=0;I<19;I++) {
    W[1+((I-1) div 2)] = V[I]+V[I+1]+I; }
cout<<w[4]<< ' '<<w[5]<< ' '<<w[6]<< ' '<<w[7]<<endl;
cout<< W[5]+W[6]
```

Resposta: _____

Outro exercício

Sejam os vetores:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	3	7	1	4	5	2	5	3	20	14	12	9	5	17	4	10
T	3	5	5	2	2	5	1	5	2	9	12	8	8	3	18	9
Z	3	2	6	7	6	4	8	8	10	17	7	5	5	20	6	1

E, sejam ainda $A = 4$, $E = -5$, $H = 7$, $K = 9$ e $M = -9$

Acompanhe os seguintes exemplos

1. $Z[14 + 1]$ 1
2. $Z[14 - 1]$ 20
3. $T[9 - 2]$ 5
4. $Z[Z[3 + A]]$ 10
5. $V[V[5]]$ 1
6. $T[T[Z[10 + 1] + K] - A] + 4$ 22
7. $T[16 - 2]$ 18
8. $Z[T[8]] - 5$ 1
9. $Z[Z[T[6 + 1] + H]] + 4$ 8
10. $V[3 + 1] - 6$ -1
11. $T[V[3 + 2]] + 3$ 8
12. $Z[T[T[8 + 1] + 2]] + 8$ 18
13. $V[6 + H] - M$ 26
14. $Z[V[7] + 1] + 7$ 13
15. $Z[V[Z[13 + E]] + M] - A$ 3
16. $V[8 + 2]$ 12
17. $Z[V[12 - K] + 2] + E$ 3
18. $V[Z[Z[3]] - 1] + 8$ 11
19. $Z[11 - 1] - K$ -2
20. $T[T[8 - 2] + 1] + E$ 0
21. $T[V[V[12] - 2] - 1] - K$ -7
22. $V[4 - 1] - M$ 13
23. $V[V[4 - M] - 2] - 9$ 1
24. $T[Z[T[4 - E] - 1] + M]$ 5

Para você fazer

Sejam os vetores:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	6	4	7	7	7	4	6	5	14	4	4	7	5	15	14	19
T	5	0	3	1	7	6	3	1	13	16	17	10	14	7	15	4
Z	1	1	3	6	6	5	5	7	0	10	12	11	10	10	6	6

E, sejam ainda $A = -2$; $E = 6$; $H = 0$; $K = -8$; $M = 1$ Resolva,

exercício 1.

```
V [ 1 + 2 ] _____+
Z [ 10 ] - 3 _____+
Z [ 10 + 1 ] + H _____+
T [ Z [ 12 ] ] _____+
Z [ V [ 10 - 1 ] ] _____+
Z [ Z [ V [ 10 ] - 1 ] + 1 ] _____+
Z [ 5 ] + A _____+
V [ V [ 5 - 2 ] - 2 ] _____+_____
```

exercício 2.

```
V [ V [ T [ 4 ] + 2 ] - 2 ] + 1 _____+
Z [ 10 + 2 ] _____+
V [ T [ 7 + 2 ] - 2 ] + E _____+
T [ Z [ T [ 8 - 2 ] ] + 2 ] + 4 _____+
Z [ 10 + 2 ] + E _____+
T [ T [ 4 ] ] _____+
V [ T [ T [ 16 - 2 ] ] ] _____+
V [ 1 ] - 0 _____+_____
```

exercício 3.

```
V [ T [ 4 - 2 ] ] - K _____+
T [ V [ V [ 12 + 2 ] ] ] - 7 _____+
V [ 2 - 1 ] _____+
Z [ T [ 8 - 1 ] + 2 ] + H _____+
T [ V [ V [ 2 ] - 2 ] + 1 ] _____+
Z [ 4 - 1 ] _____+
V [ T [ 3 + 1 ] ] _____+
Z [ V [ Z [ 3 ] + 1 ] ] _____+_____
```

exercício 4.

```
V [ 6 - 2 ] _____+
Z [ Z [ 1 + 1 ] - 2 ] - 3 _____+
T [ V [ V [ 6 ] - 2 ] + 1 ] + 9 _____+
V [ 7 + 1 ] - K _____+
Z [ T [ 4 + 2 ] ] - 7 _____+
T [ V [ T [ 13 ] - 1 ] ] _____+
Z [ 3 ] + 1 _____+
V [ Z [ 7 + 1 ] ] _____+_____
```

exercício 5.

```
T [ Z [ Z [ 6 + 1 ] ] - 2 ] + 8 _____+
T [ 11 ] + A _____+
T [ V [ 9 + 2 ] ] + 8 _____+
V [ T [ Z [ 14 ] - 1 ] - 2 ] _____+
V [ 0 + 1 ] _____+
V [ Z [ 12 - 2 ] - 1 ] - 8 _____+
T [ Z [ V [ 3 + 1 ] - 2 ] - 2 ] _____+
V [ 14 - 1 ] - 4 _____+_____
```

exercício 6.

```
T [ T [ 7 ] + 1 ] _____+
V [ Z [ V [ 1 ] + 1 ] ] + E _____+
V [ 14 - 2 ] + H _____+
Z [ T [ 6 ] - 1 ] + M _____+
Z [ T [ V [ 0 ] ] - 2 ] - A _____+
T [ 5 - 1 ] - K _____+
V [ V [ 1 - 1 ] + 1 ] _____+
T [ V [ V [ 7 ] ] + 1 ] + A _____+_____
```

Respostas

1 2 3 4 5 6

=====



Indexação e indireção

Vetores e “para” O comando *para* é muito adequado para lidar com vetores, já que neste caso, sempre se conhecem os limites inferior e superior da variação. Acompanhe nos exemplos:

1: V:vetor[1..10] de inteiro
 2: I:inteiro
 3: **para** I de 1 a 10 **faça**
 4: V[I] ← I × 10
 5: **fim para**
 6: escreva (V[3]+V[4]+V[5])

A resposta é _____

Exemplo 2:

1: V:vetor[1..30] de inteiro
 2: I:inteiro
 3: **para** I de 1 a 30 **faça**
 4: V[I] ← I
 5: **fim para**
 6: **para** I de 3 a 30 **faça**
 7: V[I] ← V[I-1]+V[I]
 8: **fim para**
 9: escreva (V[20]+V[21]+V[22])

A resposta é _____

Exemplo 3:

1: V:vetor[1..20] de inteiro
 2: W:vetor[1..10] de real
 3: I:inteiro
 4: **para** I de 1 a 20 **faça**
 5: V[I] ← 21-I
 6: **fim para**
 7: **para** I de 1 a 19 **faça**
 8: W[I+((I-1) div 2)] ← V[I]+V[I+1]+I
 9: **fim para**
 10: escreva (W[5]+W[6])

A resposta é _____

Outro exercício Sejam os vetores:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V	3	7	1	4	5	2	5	3	20	14	12	9	5	17	4	10
T	3	5	5	2	2	5	1	5	2	9	12	8	8	3	18	9
Z	3	2	6	7	6	4	8	8	10	17	7	5	5	20	6	1

E, sejam ainda $A = 4$, $E = -5$, $H = 7$, $K = 9$ e $M = -9$

Acompanhe os seguintes exemplos

1. $Z[14 + 1]$ 6
2. $Z[14 - 1]$ 5
3. $T[9 - 2]$ 1
4. $Z[Z[3 + A]]$ 8
5. $V[V[5]]$ 5
6. $T[T[Z[10 + 1] + K] + A] + 4$ 12
7. $T[16 - 2]$ 3
8. $Z[T[8]] - 5$ 1
9. $Z[Z[T[6 + 1] + H]] + 4$ 12
10. $V[3 + 1] - 6$ -2
11. $T[V[3 + 2]] + 3$ 5
12. $Z[T[T[8 + 1] + 2]] + 8$ 10
13. $V[6 + H] - M$ 14
14. $Z[V[7] + 1] + 7$ 11
15. $Z[V[Z[13 + E]] - M] - A$ 1
16. $V[8 + 2]$ 14
17. $Z[V[12 - K] + 2] + E$ 1
18. $V[Z[Z[3]] - 1] + 8$ 9
19. $Z[11 - 1] - K$ 8
20. $T[T[8 - 2] + 1] + E$ 0
21. $T[V[V[12] - 2] - 1] - K$ -7
22. $V[4 - 1] - M$ 10
23. $V[V[4 - M] + 2] - 9$ -4
24. $T[Z[T[4 - E] - 1] - M]$ 8

Para você fazer

Sejam os vetores:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V	8	7	7	7	3	8	6	3	4	15	12	12	19	2	4	9
T	2	8	1	1	2	7	4	8	9	20	11	18	15	2	15	2
Z	2	8	8	8	2	2	1	7	19	19	18	16	12	19	10	17

E, sejam ainda $A = 3$; $E = 5$; $H = -4$; $K = -4$; $M = -2$. Resolva,

exercício 1.

$T[8 - 2] - H$ _____+
 $T[9 + 1] + 3$ _____+
 $V[9 - 1] - A$ _____+
 $T[V[4 - 2] - 1]$ _____+
 $V[Z[3] + 1]$ _____+
 $Z[T[T[1 + 1] + 1] - 1] - 3$ _____+
 $Z[2]$ _____+
 $T[T[8 - 2] + 2] + M$ _____=_____

exercício 2.

$Z[V[T[11]]] + E$ _____+
 $Z[3 - 2] - 7$ _____+
 $V[T[5 + 1] - 1]$ _____+
 $T[V[V[12 - 1]] + 2] - A$ _____+
 $T[12 + 1] + 7$ _____+
 $Z[T[12 - 1] - 2] - 2$ _____+
 $V[T[V[9] - 2]] - 1$ _____+
 $V[12] + K$ _____=_____

exercício 3.

$T[V[14] + 2] - A$ _____+
 $V[V[V[6]] + 1] + M$ _____+
 $T[1]$ _____+
 $Z[Z[3 - 1]] + 6$ _____+
 $V[Z[V[14] + 2]]$ _____+
 $Z[12]$ _____+
 $T[V[10 - 2]]$ _____+
 $Z[Z[V[2] + 1] - 1]$ _____=_____

exercício 4.

$T[6] - 7$ _____+
 $V[V[9 - 1] + 2] + 8$ _____+
 $Z[Z[T[1]] - 1] + 8$ _____+
 $T[7 - 2] - M$ _____+
 $T[V[8]]$ _____+
 $T[T[Z[2] - 2]] + K$ _____+
 $V[2 + 1] - E$ _____+
 $Z[V[16 - 2]] - H$ _____=_____

exercício 5.

$Z[T[T[11 - 2] + 2]]$ _____+
 $T[7 + 2]$ _____+
 $T[T[8] + 2] + E$ _____+
 $T[Z[T[14]] - 1]$ _____+
 $Z[11]$ _____+
 $T[T[10 + 1] + 2]$ _____+
 $Z[V[V[5 - 2] + 2]]$ _____+
 $T[13 - 1]$ _____=_____

exercício 6.

$Z[T[7]] - 9$ _____+
 $Z[Z[V[13 + 1]]] - E$ _____+
 $T[8] - 4$ _____+
 $T[T[7 + 1]]$ _____+
 $V[Z[T[7 - 1]] + 2]$ _____+
 $V[1] + H$ _____+
 $V[V[11] + 2]$ _____+
 $V[V[V[3] - 2] - 1] - 8$ _____=_____

Respostas

1 2 3 4 5 6



- 1 - /

Cadeia de Caracteres como vetores A família de linguagens descendente do C (C++, Java, Clipper, Harbour etc) considera a cadeia de caracteres (também conhecida como *string*) como sendo um vetor de caracteres. Apenas para conhecimento, outras linguagens, como por exemplo LISP entendem a palavra como um átomo (indecomponível, portanto). As cadeias no C são limitadas pelo terminador “/0”. Já o PASCAL por exemplo, usa a primeira posição da cadeia como descritor de comprimento, limitando-a a 255 bytes.

Em pseudo-código, a definição é cadeia <nome>[tamanho]. Por exemplo:cadeia NOMERUA[30]. Usando-se o nome da variável isolado, está-se referindo ao vetor inteiro. Se a referência vier acompanhada de um colchete com uma expressão dentro, a referência é a um caracter isolado dentro da cadeia. Por exemplo NOMERUA pode ser “Rua Quinze de Novembro” enquanto NOMERUA[1] é “R”.

Infelizmente, estes exercícios não podem ser implementados em VisualG, pois este trata a cadeia como um item elementar, não deixa tratá-lo como vetor. O acesso aos caracteres componentes é possível mas não usando índices.

Contagem de vogais Dada uma cadeia, imprimir os 5 totalizadores de letras vogais maiúsculas “A”, “E”, “I”, “O” e “U”.

Qual a letra mais freqüente? Dada uma cadeia, informar qual a letra maiúscula mais freqüente ? Considere-se que a letra “A” ocupa a 65ª posição no código ASCII. A função “ASC” abaixo usada, recebe um caracter e devolve a posição dele dentro do código ASCII.

Upper e Lower Estas funções, presentes em boa parte das linguagens de programação, convertem uma frase toda ela para maiúscula (upper) ou para minúscula (lower). Em VG chamam-se “maiusc” e “minusc”. Atente-se que no código ASCII, as letras minúsculas ocorrem na mesma ordem das maiúsculas e exatamente 32 posições à direita no código. Por exemplo, vale sempre a regra (‘a’ é apenas um exemplo, a regra vale para todas as letras)

‘a’ = carac(asc(‘A’)+32)

Dado um nome próprio, abreviá-lo Para economizar espaço de armazenamento, esta função poderia diminuir o tamanho de nomes próprios mantendo íntegros apenas o primeiro e o último nomes e trocando os intermediários pela letra inicial seguida de um ponto.

Achar sub-cadeias dentro de uma cadeia Este é um algoritmo bem importante em Ciência da computação Uma possibilidade de solução é o chamado algoritmo de “Força Bruta” e que pode ser assim descrito:

Este algoritmo é o pior (mais demorado) que existe. Outras possibilidades como o algoritmo Knuth-Morris-Pratt ou o Boyer-Moore serão vistas no segundo ano. Apenas como comentário este algoritmo ganhou importância na manipulação do DNA, já que este é uma enorme cadeia de 4 letras (G, A, C, T).

Justificar um texto O verbo justificar neste contexto significa alinhar um pedaço de cadeia em dois limites arbitrários, à esquerda e à direita. Este alinhamento é feito distribuindo espaços em branco entre as palavras que compõe a frase. Existem diversas estratégias:

- Calculo da quantidade de espaços a incluir, digamos k .
- Incluir k espaços imediatamente antes da última palavra, deslocando-a de maneira a que seu último caractere coincida com o final da linha (***).
- Incluir 2 espaços ao invés de 1 entre cada palavra começando de trás para a frente, até totalizar k espaços
- Incluir n espaços antes da última, $n - 1$ antes da penúltima, $n - 2$ antes da ante-penúltima e assim por diante. n é escolhido de maneira a totalizar k .
- etc

Uma possível solução:

Achar a maior palavra em uma frase Dada uma frase, qual o comprimento da maior palavra e qual é ela ? Considere, para ajudar no algoritmo, que a frase sempre se encerra por um ponto final. Atente também para o truque usado ao devolver dois valores distintos em um único inteiro.

Índice de similaridade de cadeias: DNA Dados dois fragmentos de DNA, quão similares eles são ? A pesquisa em biologia molecular precisa de ferramentas que suportem buscas em seqüências de DNA por homologia (similaridade). Suponha que é necessário seqüenciar um novo gen no homem e descobriu-se que ele é similar ao gene da hemoglobina em ratos. Sai daqui que este novo gene também produz hemoglobina e que qualquer diferença encontrada é o resultado de mutações genéticas no processo evolutivo.

Para ser mais preciso, seja P um padrão a pesquisar e T um texto sobre o mesmo alfabeto. A distância de edição entre P e T é o menor número de trocas suficientes para transformar um subconjunto de T em P, através das operações de substituição, inclusão e exclusão. Este algoritmo está descrito no exercício 535 e no livro do Skiena.

Palíndromo Dada uma frase alfanumérica, ela é um palíndromo ? Exemplos de palíndromos: roma me tem amor; socorram me subi no ôni-bus em marrocos; saíram o tio e oito marias; ovo, rotor, radar; me vê se a panela da moça é de aço madalena paes e vem; a diva da vida; luz azul; em francês: esope reste ici et se repose.

Expressão parentizada Dada uma cadeia envolvendo dígitos, operações e parênteses, estes últimos estão corretos ?

Análise Estilística Dado um texto, fazer uma análise estilística. A idéia é identificar e individualizar cada uma das palavras utilizadas. Cada palavra deve ser associada a um contador e cada vez que a palavra aparecer, o contador deve ser incrementado. Em geral, costuma-se depurar as palavras a partir de uma STOP WORDS LIST, que identifica as palavras que podem ser desprezadas (por exemplo: A, O, AS, OS, ...).

Ao final, pode-se obter:

Quantidade de contadores dividido pela quantidade total de palavras.

Quantas palavras diferentes perfazem 50% do total.

Qual a porcentagem das palavras totais que representam a soma dos n maiores contadores.

Existe uma palavra cujas ocorrências sejam maiores do que 1% do total ? Além destas medidas, outras podem ser obtidas a partir da Lei de Heaps (Ziv04, pág. 305). Ela diz que o vocabulário de um texto em linguagem natural contendo n palavras tem tamanho $V = Kn^\beta$, onde K e β dependem do texto (ou do autor). K geralmente está entre 10 e 100, e β entre 0 e 1, mais propriamente entre 0,4 e 0,6. Assim, o vocabulário cresce na proporção (aproximada) da raiz quadrada do tamanho do texto.

Bate Byte, n. 35, 07/1994, diz, (por P.Kantek): ...Diante disto, comecei a elaborar um estudo particular para saber quais as palavras que eu mais freqüentemente uso, com o objetivo de elaborar um dicionário pessoal que aumente a minha produtividade durante a tarefa de digitação.

Comecei o levantamento na minha dissertação de mestrado, e os resultados a que cheguei estão a seguir descritos. Antes de analisá-los, entretanto, valem duas considerações: A primeira é de que trata-se de um texto específico, tratando sobre um aspecto único da computação. O segundo é que todo texto reflete um pouco do seu autor. Diante das duas questões, deve-se tomar muito cuidado ao generalizar os resultados.

A dissertação trata sobre o tema ”downsizing”, tem 163 páginas A4, com letra tamanho 10. É formada por 68.363 palavras. Dessas, apenas 42.704 têm três ou mais bytes e apenas 7.746 são diferentes.

As mais freqüentes:

que = 1234 ocorrências, sistemas = 702, para = 534, uma = 532, comandar = 439, não = 322, por = 294, mais = 274, como = 259, ser = 239, dar = 191, informática = 179, tem = 173, são = 167, esta = 130, suas = 130, este 122, pelo = 112, entre = 112, software = 111 e muito = 108.

Proposta:

👉 Para você fazer

Para ganhar a nota correspondente a esta folha, você deve implementar um dos algoritmos acima estudados em uma linguagem de programação qualquer VisualG, Java, C, Pascal, C++, SmallTalk, Clipper, APL, Basic, Lisp, Fortran, Assembler, ... e entregar a listagem grampeada a esta folha. A correção estará sujeita às regras de

- Ineditismo: copias de trabalhos (desde que assim identificados) darão origem a nota zero a ambos
- Funcionamento: o programa entregue terá que funcionar e resolver o problema proposto.
- Garantia: o professor poderá solicitar a execução deste programa no laboratório diante de uma caso inédito.



Vetores, matrizes e registros

Os 5 tipos básicos das linguagens comuns (inteiro, real, caracter, cadeia e lógico) nem sempre são suficientes para quem precisa fazer alguma computação mais sofisticada.

Usualmente, o que se tem é algum agrupamento de variáveis distintas que passam a ser processadas juntas de alguma maneira. A este agrupamento dá-se o nome de ARRANJO, e de cara pode-se afirmar existirem 2 grandes tipos de arranjo.

Arranjo homogêneo (vetor) obtido pela repetição de uma variável (ou um arranjo) mais simples. Também é chamado de vetor e neste, a individualização dos componentes se dá pelo seu número dentro do arranjo.

Por exemplo, podemos ter um conjunto de 10 inteiros, assim definido:
A:vetor[1..10] de inteiro. Neste caso, existem 10 variáveis, todas batizadas com o mesmo nome “A”. O primeiro é identificado por A[1], o segundo por A[2] e assim por diante. O último é o A[10].

Embora C (e Java) não aceitem, nós aqui vamos combinar que quando se quiser referenciar o conjunto (os 10 valores) basta escrever o seu nome, sem o colchete ao lado. Por exemplo, ao escrever **leia A** se estará dizendo que o programa deve ler os 10 números. Se se quiser ler apenas o primeiro número, deve-se fazer **leia A[1]**.

Isso vale para todas as operações: leitura (**leia A**), inicialização (**A ← 0**), impressão e assim por diante.

Outra questão é qual é o número do primeiro elemento. C e Java exigem o início da numeração em zero, o Pascal deixa o usuário escolher enquanto outras linguagens (Cobol e Basic, por exemplo) iniciam em 1. Nós aqui vamos padronizar o UM, porque é a maneira que os seres humanos usam para contar. Quando precisarmos começar a numeração em zero, haverá sempre um aviso explícito.

Exemplos de definição e uso de vetores:

Uma data, formada por dia, mês e ano. A definição pode ser **DATA:vetor[1..3] de inteiro**. Aqui, o dia é DATA[1], o mês é DATA[2] e o ano é DATA[3]. Veja-se que as 3 variáveis são inteiras.

O conjunto dos 13 salários líquidos de um empregado. **SALAR:vetor[1..13] de real.**

A quantidade de alunos matriculados nas 4 séries deste curso. **QTD_ALU:vetor[1..4] de inteiro.**

Cadeia e vetor de caracteres Como é muito comum e necessário usar vetor de caracteres, definiu-se um tipo fundamental chamado *cadeia* que vem a ser exatamente isto. Assim, definir **X:vetor[1..30] de caracter**, pode ser simplificado para **X:cadeia[30]**. É a mesma coisa.

Arranjo heterogêneo (registro) Aqui agrupam-se duas ou mais variáveis de tipo original diferente. Por exemplo, uma placa de veículo em nosso país é formada por 3 letras e 4 dígitos. Para definir este arranjo, não se pode usar um vetor, pois o primeiro campo (as letras) são estruturalmente diferentes do segundo campo (os dígitos).

Para definir arranjos heterogêneos, o caminho é um pouco mais demorado. Antes de tudo, há que se definir uma nova estrutura. Faz-se isso com o comando **estrutura**, que é encerrado pelo terminador **fim estrutura**. Acompanhe o exemplo da definição de uma placa de veículo.

```
estrutura PLACA
LETRAS : cadeia[3]
NUMEROS : inteiro
fim estrutura
```

Depois de ter definido a nova estrutura (tipo), há que se definir uma variável que tenha esta estrutura (tipo). Faz-se isto, escrevendo **MEUCARRO:PLACA**. Neste segundo comando, PLACA define a estrutura da variável MEUCARRO, que acaba de ser definida.

Este segundo comando pode ser comparado à definição de um tipo básico, senão veja-se: **MEUCARRO:PLACA** comparado com **X:inteiro**. A diferença é que PLACA é um tipo definido pelo usuário, enquanto “inteiro” é um tipo básico, já conhecido por todos.

Quando o usuário emitir o comando **leia MEUCARRO**, o programa deve ler as duas informações: as letras e os dígitos.

A questão surge quando se pretende isolar um dos dois componentes da variável MEUCARRO. Aqui não se usam os índices e sim os nomes. O primeiro campo (as letras) são obtidas escrevendo-se **MEUCARRO.LETRAS**. A propósito, a primeira letra é **MEUCARRO.LETRAS[1]**. O segundo campo é obtido com **MEUCARRO.NUMEROS**.

Usa-se aqui o operador PONTO “.” que serve exatamente para separar os nomes simples formadores dos nomes compostos usados em registros.

Erros Nos arranjos homogêneos, (vetores) um erro comum é: **ABC:vetor[1..30] de inteiro**, seguido de **ABC[42] ← 0**

Perceba-se que pediu-se um índice que cai fora da definição do vetor (que é de 1 a 30). Isto costuma causar um erro de índice que aborta a execução do programa (exceto em C).

Nos arranjos heterogêneos (registros) um erro comum é usar a estrutura para referir-se à variável:

```
estrutura ENDERE
NOMERUA:cadeia[30]
BAIRRO:cadeia[20]
CIDADE:cadeia[15]
fim estrutura
```

NUMERO:inteiro
CEP:inteiro

O erro poderia ser **leia ENDERE**. Note que ENDERE não é uma variável é apenas um tipo.

Vetor de registros Nada impede que você escreva

```
estrutura ENDERE
NOMERUA:cadeia[30]
NUMERO:inteiro
fim estrutura
```

E depois defina um vetor de 10 endereços, assim: **VV:vetor[1..10] de ENDERE**. Note que o vetor é VV, enquanto o tipo de VV é ENDERE. Note também que ele ocorre 10 vezes.

Registro de Vetores Também nada impede que dentro de um registro exista um vetor. Por exemplo

```
estrutura ENDERE
NOMERUA:cadeia[30]
NUMERO:inteiro
SALDOS:vetor[1..5] de inteiro
fim estrutura
X:ENDERE
```

Agora, a obtenção do primeiro saldo é **X.SALDOS[1]**.

Indireção A indireção é a múltipla indexação. Por exemplo suponha o vetor

```
inteiro VV[6]
VV ← 0, 2, 3, 5, 11, 5
Ao escrever VV[4], a resposta é 5
Ao escrever VV[VV[4]], a resposta é 11
```

Matriz é a generalização do conceito de vetor para mais de uma dimensão.

Por exemplo, se quisermos definir uma planilha de 12 linhas por 7 colunas, contendo inteiros, pode-se fazer **PLANILHA:vetor [1..12][1..7] de inteiro**. O acesso aos dados, agora se dá usando-se 2 índices, um para cada dimensão. A obtenção do valor na segunda linha, quarta coluna da planilha é **PLANILHA[2,4]**.

🔗 Para você fazer

- Defina as seguintes variáveis:
Um vetor de 8 inteiros e nome ALFA

Um vetor de 12 reais de nome BETA

Um vetor de 30 caracteres com o nome RUA

- Defina um registro e depois uma variável que o utilize: Um contracheque (nome, mes/ano, salários bruto e líquido). Depois defina a variável **FUNC1** com essa estrutura.

Idem para um cartão de visita (nome, cargo, telefone e email). Defina 3 variáveis de nomes GEREN, SUPV, FUNC.

- Defina um registro que contenha um nome e 8 valores. Defina uma variável com esta estrutura. Depois imprima o primeiro e o ultimo desses valores da variável.

- Defina um registro que tenha uma data e um veículo. Depois defina um vetor que tenha 6 ocorrências desse registro. Defina uma variável chamada AGORA associada a esse registro. Imprima o dia da primeira ocorrência das seis dentro de AGORA.

- Defina uma matriz para guardar as notas das suas disciplinas durante todo o ano letivo.

Manuseio de vetores Os vetores são elementos fundamentais na programação. Em sentido amplo, quase tudo pode ser entendido como um vetor: a web pode ser um vetor onde cada página é um elemento desse vetor (isto sob o ponto de vista do google, por exemplo), um banco de dados pode ser um vetor (cada registro é um elemento), a memória do computador é um vetor (cada byte é um elemento), e assim por diante.

Assim, vamos definir algumas operações fundamentais em vetores, com a generalidade que nos for possível.

inclusão	adicionar um novo elemento ao vetor
exclusão	eliminar um elemento do vetor
ordenação	mudar a ordem dos elementos do vetor
busca	dado um vetor V e uma chave K, descobrir se $K \in V$

Ordenação Ordenar um vetor é trocar as posições de seus elementos de maneira a que eles entrem em alguma ordem (crescente ou decrescente).

Em termos matemáticos, seja X um conjunto composto por i elementos entre os quais se pode estabelecer uma relação de ordem. Dados X_i e X_j com $i \neq j$, sempre pode-se estabelecer $X_i > X_j$ ou $X_i = X_j$ ou $X_i < X_j$, onde valha sempre a propriedade:

ordem crescente	$V[i+i] \geq V[i], \forall i$
ordem decrescente	$V[i+i] \leq V[i], \forall i$
ordem estritamente crescente	$V[i+i] > V[i], \forall i$
ordem estritamente decrescente	$V[i+i] < V[i], \forall i$

O algoritmo de ordenação é o mais extensivamente estudado (ocupa 80% das quase 600 páginas do volume 2 do “The Art of computer programming” do Knuth) e isso acontece porque:

- ★ É um algoritmo caro (demorado).
- ★ É o algoritmo mais usado no universo. (James Martin, sugere que 40% do tempo de todos os computadores da terra é gasto neste algoritmo).

Um algoritmo de ordenação é estável se no caso particular em que se $k[i] = k[j]$, a ordem original dos dados é preservada. É sempre bom que um método de ordenação seja estável.

Veja-se alguns exemplos:

1 4 6 21 60 62	estritamente crescente
1 4 6 6 21 60	crescente
2 2 2 2 2	crescente
1 4 21 19 50 60	desordenado
60 21 6 4 1	estritamente decrescente
60 21 21 6 4 1	decrescente
2 2 2 2 2	decrescente
60 50 19 21 4 1	desordenado
1 2 3 4 5 6	
90 61 15 4 2	
12 16 18 18 90	
12 16 18 17 90	
90 61 15 17 4 2	
17 2 89 7 48 1 90	

A seguir, dois dos algoritmos de ordenação possíveis (Note que são conhecidos mais de 20 algoritmos de ordenação).

Método de seleção direta

Ordenação por seleção direta:

- 1: **Algoritmo** SELEÇÃO(V:vetor[1..n]de inteiro)
- 2: I,J,COR,INUI : inteiro
- 3: $I \leftarrow 1$
- 4: enquanto ($I \leq n$)
- 5: COR $\leftarrow V[I]$
- 6: INUI $\leftarrow I$
- 7: J $\leftarrow I + 1$
- 8: enquanto ($J \leq n$)
- 9: se $V[J] < COR$
- 10: COR $\leftarrow V[J]$
- 11: INUI $\leftarrow J$
- 12: fim{se}
- 13: J $\leftarrow J + 1$
- 14: fim{enquanto}
- 15: TROCA (I, INUI)

- 16: I $\leftarrow I + 1$
 - 17: fim{enquanto}
 - 18: devolva (V)
 - 19: fim algoritmo
- Bolha**
- Ordenação por bolha
- 1: **Algoritmo** BOLHA (V:vetor[1..n]de inteiro)
 - 2: inteiro I, J
 - 3: $I \leftarrow n$
 - 4: enquanto ($I \geq 2$)
 - 5: J $\leftarrow 2$
 - 6: enquanto ($J \leq I$)
 - 7: se ($V[J-1] > V[J]$)
 - 8: TROCA (J, J-1)
 - 9: fim{se}
 - 10: J $\leftarrow J + 1$
 - 11: fim{enquanto}
 - 12: I $\leftarrow I - 1$
 - 13: fim{enquanto}
 - 14: devolva V
 - 15: fim algoritmo

Inclusão Note-se que a maioria das linguagens não permite a mudança dinâmica do tamanho de um vetor. (Por exemplo, em VG, após definir **V : vetor[1..15] de inteiro** não é possível durante o programa mudar o limite superior de 15 para 16 ou outro número qualquer). Assim, se pretendermos fazer inclusões em um vetor, é necessário defini-lo com espaço livre ao final. Para isso funcionar, temos que ter uma variável de controle (um bom nome para ela é ULTIMO), que contém a última ocorrência válida no vetor.

No exemplo do VG, se quisermos 15 elementos válidos, mas com espaço para futuras inclusões, faremos:

V : vetor [1..25] de inteiro
ULTIMO $\leftarrow$ 15

Se a tabela é mantida desordenada, a inclusão pode e deve ser feita no seu final e o algoritmo é simples:

- 1: função INCLUSAO (V:vetor, K:inteiro) : vetor
- 2: ULTIMO $\leftarrow$ ULTIMO + 1
- 3: V[ULTIMO] $\leftarrow$ K

Naturalmente, ao fazer **ULTIMO+1**, há que se ter o cuidado de não ultrapassar o tamanho definido do vetor, sob pena de ganhar um INDEX ERROR.

Se a tabela é mantida ordenada, a inclusão fica um pouco mais complexa, já que:

- ★ Há que se achar o local exato da inclusão
- ★ Empurrar todos os elementos à direita desse local, para a casa ao lado
- ★ Efetuar a inclusão

Acompanhe tudo isso no algoritmo

- 1: função INCLUSAO (V:vetor, K:inteiro) : vetor
- 2: $I \leftarrow 1$
- 3: enquanto V[I] < K
- 4: I $\leftarrow I + 1$
- 5: fim{enquanto}
- 6: J $\leftarrow$ ULTIMO
- 7: enquanto J $\geq$ I
- 8: V[J+1] $\leftarrow$ V[J]
- 9: J $\leftarrow J - 1$
- 10: fim{enquanto}
- 11: V[I] $\leftarrow$ K
- 12: ULTIMO $\leftarrow$ ULTIMO + 1

Outra alternativa (ruim) é incluir o elemento ao final e reordenar a tabela. Porque esta alternativa é ruim ?

Exclusão Excluir um elemento significa deixar de considerá-lo como válido. Existem duas estratégias aqui: excluir fisicamente o elemento, o que sempre implicará em puxar uma posição os elementos que estavam à direita do excluído (já que não é possível deixar buracos no meio do vetor). Note que este cuidado é necessário mesmo que o vetor esteja desordenado. Eis o algoritmo

- 1: função EXCLUSAO (V:vetor, K:inteiro) : vetor
- 2: $I \leftarrow 1$
- 3: enquanto V[I] $\neq$ K
- 4: I $\leftarrow I + 1$
- 5: fim{enquanto}
- 6: enquanto I < ULTIMO
- 7: V[I] $\leftarrow$ V[I+1]
- 8: I $\leftarrow I + 1$
- 9: fim{enquanto}
- 10: ULTIMO $\leftarrow$ ULTIMO - 1

A propósito, note-se que o algoritmo pressupõe neste caso que $K \in V$. Se isto tiver que ser verificado (sempre é bom fazer isso) o algoritmo fica um pouco mais complicado:

- 1: função EXCLUSAO (V:vetor, K:inteiro) : vetor
- 2: $I \leftarrow 1$
- 3: enquanto V[I] $\neq$ K
- 4: I $\leftarrow I + 1$
- 5: se I $\geq$ tamanho(V)
- 6: ...ERRO: K não existe em V
- 7: fim{se}
- 8: fim{enquanto}
- 9: enquanto I < ULTIMO
- 10: V[I] $\leftarrow$ V[I+1]
- 11: I $\leftarrow I + 1$
- 12: fim{enquanto}
- 13: ULTIMO $\leftarrow$ ULTIMO - 1

Imaginou-se no caso acima que a ocorrência de K é única. Se houver mais de uma, e todas essas ocorrências precisam ser excluídas, há que se modificar um pouco o algoritmo acima.

Note o custo computacional de fazer uma única exclusão. Para fugir dele é comum usar outra estratégia. Na exclusão de um elemento, ele é excluído apenas LOGICAMENTE, sendo marcado como inválido, embora fisicamente ele continue lá. Isto poderia ser feito colocando um valor inválido (por exemplo, um número negativo) na posição do elemento a ser excluído. Ao fazer isso, deixa-se de precisar puxar todos os seus vizinhos.

Busca A busca é a mais comum das operações. Hoje vamos ver apenas um algoritmo simples e na próxima aula vamos olhar 3 algoritmos que fazem a busca. É também o momento do curso de começar a exercitar um maravilhoso exercício intelectual: *Dados dois (ou mais) algoritmos que fazem exatamente a mesma coisa, qual deles usar ?*

Não é um exercício inútil. As diferenças de desempenho entre algoritmos equivalente (que fazem a mesma coisa) pode ser de 100 vezes ou mais. Não é incomum este número.

A propósito, vale a pergunta: como se mede o custo de um algoritmo ? Como veremos no segundo ano (no tópico Teoria da Complexidade Algorítmica) em termos de algoritmos é muito verdadeira a frase **tempo é dinheiro**. Só que a medida do tempo é simplificada:

- ★ Não tem sentido usar minutos ou segundos: cada computador, cada S.O. e cada ambiente são únicos.
- ★ Usa-se então a unidade “operação computacional” e convencionase que ela demora 1 unidade.

Busca Linear Para este algoritmo o conjunto de chaves não é mantido ordenado. O algoritmo básico é:

- 1: função BUSCAL(CHAVE:inteiro, V:vetor)
- 2: ONDE, I : inteiro
- 3: ONDE $\leftarrow$ -1
- 4: $I \leftarrow 1$
- 5: enquanto i $\leq$ tamanho(V)
- 6: se CHAVE = V[I]
- 7: ONDE $\leftarrow$ I
- 8: I $\leftarrow$ tamanho(V)
- 9: fim{se}
- 10: I $\leftarrow I + 1$
- 11: fim{enquanto}
- 12: devolva ONDE

 **Para você fazer** Localize na bibliografia do curso o nome de pelo menos 5 algoritmos (além da seleção e do bolha) que servem para ordenar um vetor. Responda a seguir:



Vamos modificar o algoritmo básico para registrar quantas comparações o algoritmo fará (independente de achar ou não achar a CHAVE. Isso é feito para comparar o desempenho do algoritmo frente aos outros dois listados abaixo.

```
1: função BUSCALIN(CHAVE:inteiro, V:vetor)
2: qtd, I : inteiro
3: I ← 1
4: qtd ← 0
5: enquanto i ≤ tamanho(V)
6:   qtd ← qtd + 2
7:   se CHAVE = V[I]
8:     ...ACHOU...
9:     I ← tamanho(V)
10:  fim{se}
11:  I ← I + 1
12: fim{enquanto}
13: devolva qtd
```

```
37 BUSCALIN 62 45 61 78 1 94 30 28 4 31
            33 92 37 2 23 -----> 26
98 BUSCALIN 70 98 81 46 29 12 65 40 72 14
            20 4 82 2 30 -----> 4
3 BUSCALIN 29 66 71 42 48 44 69 35 30 37
            94 68 36 19 21 -----> 30
```

Busca linear ordenada Neste algoritmo, o conjunto de chaves deverá estar ordenado. Eis o algoritmo:

```
1: função BUSCALO (CHAVE:inteiro, V:vetor)
2: ONDE, I : inteiro
3: I ← 1
4: ONDE ← -1
5: enquanto (I ≤ tamanho(TABELA)) ∧ (CHAVE > V[I])
6:   I ← I + 1
7: fim{enquanto}
8: se CHAVE = V[I]
9:   ONDE ← I
10: fim{se}
11: devolva ONDE
```

Versão para contar as comparações:

```
1: função BUSCALINORD (CHAVE:inteiro, V:vetor)
2: I ← 1
3: qtd ← 0
4: enquanto (I ≤ tamanho(TABELA)) ∧ (CHAVE > V[I])
5:   qtd ← qtd + 2
6:   I ← I + 1
7: fim{enquanto}
8: se CHAVE = V[I]
9:   ...ACHOU...
10: fim{se}
11: devolva qtd
```

```
96 BUSCALINORD 5 10 26 34 39 48 56 60 68 70
              73 89 92 93 96 -----> 28
58 BUSCALINORD 1 2 8 10 12 41 42 47 49 53
              58 59 60 62 85 -----> 20
96 BUSCALINORD 2 12 21 26 27 37 43 46 52 54
              65 79 83 84 89 -----> 30
```

Existe uma busca muito mais eficiente, no caso do vetor estar ordenado. É a busca binária, que usa a estratégia de *dividir para conquistar*. Eis o algoritmo:

```
1: função BUSCABINARIA (V:vetor, K:inteiro)
2: inf, sup, meio : inteiro
3: inf ← 1
4: sup ← tamanho(V)
5: enquanto inf ≤ sup
6:   meio ← (inf + sup) ÷ 2
7:   se K = V[meio]
8:     retorne meio
9:   abandone
10:  senão
11:    se K < V[meio]
12:      sup ← meio-1
13:    senão
14:      inf ← meio+1
15:  fim{se}
```

```
16: fim{se}
17: fim{enquanto}
18: retorne -1 // insucesso
```

Busca linear com sentinela A tabela não precisa estar ordenada, e a busca gasta apenas a metade dos testes da busca linear. A sentinela é a chave que se busca que é forçosamente incluída ao final da tabela. Assim, a saída do laço sempre se dará pela igualdade de chaves. Se a chave procurada for encontrada no meio da tabela, ...ACHOU... Se for encontrada no final da tabela, (onde foi explicitamente colocada) é porque ela não se encontrava lá antes do início do algoritmo.

```
1: função BUSCASE (CHAVE:inteiro, V:vetor)
2: ONDE, I : inteiro
3: ONDE ← -1
4: I ← 1
5: V[ULTIMO+1] ← CHAVE {note que ULTIMO não é alterado}
6: enquanto CHAVE ≠ V[I]
7:   I ← I + 1
8: fim{enquanto}
9: se I ≠ (ULTIMO + 1)
10:  ONDE ← I
11: fim{se}
12: devolva ONDE
```

Versão modificada para contar as comparações:

```
1: função BUSCALINSEN (CHAVE:inteiro, V:vetor)
2: I ← 1
3: qtd ← 0
4: V[ULTIMO+1] ← CHAVE {note que ULTIMO não é alterado}
5: enquanto CHAVE ≠ V[I]
6:   qtd ← qtd + 1
7:   I ← I + 1
8: fim{enquanto}
9: se I ≠ (ULTIMO + 1)
10:  ...ACHOU...
11: fim{se}
12: devolva qtd
```

```
38 BUSCALINSEN 24 35 91 18 8 72 10 88 27 54
              28 34 43 3 38 -----> 14
54 BUSCALINSEN 63 17 35 68 27 86 54 53 5 78
              41 58 44 51 25 -----> 6
61 BUSCALINSEN 75 67 51 60 85 38 49 74 73 78
              66 35 24 4 2 -----> 15
```

Alguns dados experimentais Para o teste a seguir, fiz 6 passagens de 1000 chaves. Para cada método, em cada passagem foram 3 tentativas: as primeiras 2 com sucesso e a última para uma chave inexistente. Eis os dados:

passagem	Lin	Lin Ord	Lin Sen
1	1265.3	1063.3	817.3
2	1777.3	749.3	621.7
3	1832.0	1314.0	674.0
4	1244.7	613.3	634.7
5	1200.7	1186.7	607.3
6	948.0	832.7	770.3
média	1378.0	959.9	687.6

Note-se que na média, o terceiro algoritmo (o da sentinela) é cerca do dobro de rápido em relação ao primeiro.

Manuseio em geral

Vamos brincar um pouco com vetores, construindo alguns algoritmos *ad hoc*. Por exemplo, sabendo existirem 1200 alunos em um determinado curso, se quisermos saber quantos alunos há em cada um dos 5 anos do curso, e supondo que cada aluno é assim identificado pela tripla de informações (código do aluno, série em que ele está e código IBGE do município onde ele nasceu)

```
1: COD : inteiro
2: SERIE : inteiro
3: MUN : inteiro
   Como fazer ?
1: QTDAL : vetor[1..5] de inteiro
2: para I de 1 ate 1200
3:   leia(COD, SERIE, MUN)
4:   QTDAL [SERIE] ← QTDAL [SERIE] + 1
5: fim{para}
6: para I de 1 ate 5
7:   (escreval("Alunos da serie ",I,"são ",QTDAL[I])
8: fim{para}
```

Um segundo caso, algo mais complexo, ocorre quando os identificadores da classe não são conhecidos de antemão. Por exemplo, suponhamos querer saber qual o local de nascimento (município) dos 1200 alunos do curso.

Agora, o índice de acesso não é mais conhecido de antemão e precisa ser localizado. Suponhamos existir o campo MUN, como sendo o código numérico do município onde o aluno nasceu. Suponhamos que um limite exagerado (para a quantidade de municípios) seja 200.

Criaremos dois vetores, o primeiro contendo os códigos de município que forem surgindo e o segundo para guardar os contadores. Ambos os vetores serão acessados pelo mesmo índice.

Para cada MUN lido, deve-se pesquisar o vetor MUNICIPIO até que uma das três coisas aconteça: a) Achar um MUNICIPIO = MUN; b) Achar um MUNICIPIO = 0; c) Ultrapassar o limite de I = 200.

A primeira indica que este município já havia aparecido e seu contador deve ser incrementado em uma unidade. A segunda, informa ser a primeira vez que este município está aparecendo e seu contador deve ser 1. A terceira, não deveria ocorrer e se acontecer indicará um erro.

```
Eis o algoritmo
1: MUNICIPIO : vetor[1..200] de inteiro
2: CONTADOR : vetor[1..200] de inteiro
3: para I de 1 ate 1200
4:   leia (MUN)
5:   I ← 1
6:   enquanto MUN ≠ MUNICIPIO[I] ∧ MUNICIPIO[I] ≠ 0
7:     I ← I + 1
8:     se I > 200
9:       ...erro... (200 é pouco)
10:    fim{se}
11:    fim{enquanto}
12:    MUNICIPIO[I] ← MUN[I]
13:    CONTADOR[I] ← CONTADOR[I] + 1
14: fim{para}
```

A questão agora é descobrir quem é o município campeão de alunos, isto é qual o maior valor do vetor CONTADOR acima criado. Deve-se notar que este processamento (achar o maior valor de um vetor) é extraordinariamente comum em programação.

A estratégia é criar duas variáveis, de nome MELHOR e QUAL. Fazendo uma passada no vetor, sempre que CONTADOR[I] for maior a MELHOR, vamos guardar este valor maior na variável MELHOR. Junto vai-se guardar o índice (o endereço) deste MELHOR na variável QUAL.

```
1: // achar o mais freqüente
2: MELHOR ← -∞
3: I ← 1
4: enquanto I ≤ 200 ∧ MUNICIPIO[I] ≠ 0
5:   se CONTADOR[I] > MELHOR
6:     MELHOR ← CONTADOR[I]
7:     QUAL ← I
8:   fim{se}
9:   I ← I + 1
10: fim{enquanto}
11: escreva ("O campeão é o município ", MUNICIPIO[QUAL])
```

 **Para você fazer** A seguir 3 execuções das funções de busca modificadas para medir a quantidade de operações de comparação. O algoritmo a implementar é exatamente igual à versão acima listada. Busca-se descobrir o valor da função **qtd** em cada execução. Os vetores estão em duas linhas por falta de espaço. Na verdade o número antes do nome da função é a chave que se busca e o vetor que segue o nome da função é um conjunto de 15 inteiros.

```
1 BUSCALIN 46 32 61 73 42 71
83 70 69 95 14 38 64 40 56 é =====
12 BUSCALINORD 5 8 9 13 16 39
56 61 63 68 72 73 88 91 98 é =====
68 BUSCALINORD 59 43 80 11 71 35
14 73 52 77 42 30 56 37 88 é =====
```


- 1 - /

Manuseio de tabelas II

Para todos os algoritmos abaixo suponha-se duas variáveis globais:

1: inteiro TABELA[num]
 2: inteiro ULTIMO

Onde *TABELA* contém os dados a serem acessados. *num* é um valor suficiente para conter todos os dados necessários e *ULTIMO* é um valor que aponta para o último valor válido da tabela.

Inclusão de itens em tabelas desordenadas Esta inclusão é fácil, bastando acrescentar o item ao final da área útil da tabela.

1: função INCLUSAONORD (inteiro CHAVE)
 2: ULTIMO ← ULTIMO + 1
 3: **se** ULTIMO > num **então**
 4: ...erro...
 5: **fim se**
 6: TABELA[ULTIMO] ← CHAVE

Qual a quantidade de operações para fazer uma inclusão em tabela não ordenada no caso de uma tabela de

elementos	operações
100	
10.000	
1.000.000	
n	

Exclusão de itens em tabelas desordenadas Há duas estratégias aqui: a primeira é eliminar o item puxando os que ficaram abaixo da exclusão para “tampar o buraco”.

1: função EXCLUSAO1NORD (inteiro CHAVE)
 2: inteiro i ← 1
 3: **enquanto** TABELA[i] ≠ CHAVE **faça**
 4: i++
 5: **se** i > ULTIMO **então**
 6: ...erro...
 7: **fim se**
 8: **fim enquanto**
 9: **enquanto** i < ULTIMO **faça**
 10: TABELA[i] ← TABELA[i+1]
 11: i++
 12: **fim enquanto**
 13: ULTIMO ← ULTIMO - 1

Qual a quantidade de operações para fazer uma exclusão do tipo 1; qual o número médio de operações para achar um item que está na tabela e qual o número médio de operações para concluir que um dado número não está na tabela

elementos	ops para exclusão	ops para achar	ops para concluir que não está
100			
10.000			
1.000.000			
n			

A segunda estratégia é incluir algum “indicador” (*filler*) que informe que o item foi eliminado.

1: função EXCLUSAO2NORD (inteiro CHAVE)
 2: inteiro i ← 1
 3: **enquanto** TABELA[i] ≠ CHAVE **faça**
 4: i++
 5: **se** i > ULTIMO **então**
 6: ...erro...
 7: **fim se**
 8: **fim enquanto**
 9: TABELA[i] ← 9999999

Qual a quantidade de operações para fazer uma exclusão do tipo 2; qual o número médio de operações para achar um item que está na tabela e qual o número médio de operações para concluir que um dado número não está na tabela

elementos	ops para exclusão	ops para achar	ops para concluir que não está
100			
10.000			
1.000.000			
n			

A vantagem desta segunda alternativa é que a exclusão é mais rápida, mas em compensação as buscas tendem a ficar mais demoradas.

Busca binária Conjunto ordenado, ótimo desempenho

1: função BUSCABIN(inteiro CHAVE)
 2: inteiro INIC, METADE, FIM
 3: INIC ← 1
 4: FIM ← ULTIMO
 5: **repita**
 6: METADE ← ⌊ ((INIC + FIM)/2)
 7: **se** CHAVE ≤ TABELA[METADE] **então**
 8: FIM ← METADE - 1
 9: **senão**
 10: INIC ← METADE + 1
 11: **fim se**
 12: **até** TABELA[METADE] = CHAVE ∨ INIC > FIM
 13: **se** TABELA[METADE] = CHAVE **então**
 14: devolva METADE
 15: **senão**
 16: devolva -1
 17: **fim se**

Qual a quantidade de operações para fazer pesquisa e qual o número médio de operações para concluir que um dado número não está na tabela

elementos	ops para pesquisa	ops para concluir que não está
100		
10.000		
1.000.000		
n		

Inclusão em tabela ordenada A contrapartida para poder fazer a busca ordenada é criar e manter a tabela em ordem. Para isso, a inclusão de novos elementos deve ocorrer em seus locais específicos e não no final, como vimos acima.

1: função INCLUSAOOORD (inteiro CHAVE)
 2: ULTIMO ← ULTIMO + 1
 3: **se** ULTIMO > num **então**
 4: ...erro...
 5: **fim se**
 6: inteiro i ← 1
 7: **enquanto** TABELA[i] < CHAVE **faça**
 8: i++
 9: **fim enquanto**
 10: k ← ULTIMO
 11: **enquanto** k > i **faça**
 12: TABELA[k] ← TABELA[k-1]
 13: k--
 14: **fim enquanto**
 15: TABELA[k] ← CHAVE

Qual a quantidade de operações para fazer uma inclusão em tabela ordenada no caso de uma tabela de

elementos	operações
100	
10.000	
1.000.000	
n	



Para você fazer

Escreva a seguir um algoritmo de uma função que Receba uma matriz global, de nome MAT-ENT, cujo conteúdo é:

MAT-ENT é uma matriz de 3 colunas, sendo todas inteiros, formada por

código da transação	valor da transação	código da cidade

O algoritmo deverá criar e imprimir uma segunda matriz, de apenas duas colunas

MAT-SAI é uma matriz de 2 colunas, sendo ambos inteiros, formada por

código da cidade	total dos valores de transações

Nesta matriz de saída, os valores de débito (cód 2) deverão ser totalizados, e quando houver **quebra** de cidade, uma nova linha deve ser introduzida na matriz de saída com o total. Note que a matriz de entrada está ordenada por cidade. (ENUNCIADO 2)

Orientações:

- Anexe a esta folha a listagem deste algoritmo produzida pelo software VISUALG. Anexe também as listagens das matrizes de entrada e de saída.
- Para baixar, www.apoioinformatica.inf.br

Enunciados 119

Enunciado 1

Receba duas matrizes globais, de nomes MAT1 e MAT2, cujos conteúdos são:

MAT1 é uma matriz de 3 colunas, sendo todas inteiros, formada por

código da transação	valor da transação	código da cidade

MAT2 é uma matriz de 2 colunas, sendo ambos inteiros, formada por

código da cidade	estado a que pertence

O algoritmo deverá criar e imprimir uma terceira matriz, de apenas duas colunas

MATSAIDA é uma matriz de 2 colunas, sendo ambos inteiros, formada por

estado onde foi efetuada a transação	valor da transação

Nesta segunda matriz, apenas as transações de débito deverão ser incluídas. (ENUNCIADO 1)

Enunciado 2

Receba uma matriz global, de nome MAT-ENT, cujo conteúdo é:

MAT-ENT é uma matriz de 3 colunas, sendo todas inteiros, formada por

código da transação	valor da transação	código da cidade

O algoritmo deverá criar e imprimir uma segunda matriz, de apenas duas colunas

MAT-SAI é uma matriz de 2 colunas, sendo ambos inteiros, formada por

código da cidade	total dos valores de transações

Nesta matriz de saída, os valores de débito (cód 2) deverão ser totalizados, e quando houver **quebra** de cidade, uma nova linha deve ser introduzida na matriz de saída com o total. Note que a matriz de entrada está ordenada por cidade. (ENUNCIADO 2)

Enunciado 3

Receba uma matriz global, de nome MAT-VENDAS, cujo conteúdo é:

MAT-ENT é uma matriz de 3 colunas, sendo todas inteiros, formada por

número de parcelas	valor da transação	código da cidade

O número de parcelas pode ser de 1 a 6.

O algoritmo deverá criar e imprimir uma segunda matriz, de apenas duas colunas

MAT-SAI é uma matriz de 2 colunas, por 2 linhas sendo ambos inteiros, formada por

número de parcelas	quantidade de vendas

Nesta matriz de saída, deverão ser mostrados o número de parcelas campeão positivo (em quantidade de vendas) e o campeão negativo (idem), associado ao número de vendas de cada um. (ENUNCIADO 3)

Enunciado 4

Receba uma matriz global, de nome MAT-ENT, cujo conteúdo é:

MAT-ENT é uma matriz de 3 colunas, sendo todas inteiros, formada por

código do estado	área do estado	população do estado

O algoritmo deverá imprimir o código do estado de maior densidade populacional e o código do estado de menor densidade populacional, imprimindo também quantas vezes o primeiro é mais populoso do que o segundo. (ENUNCIADO 4)

Prática em conjuntos

Exemplo 1 Seja o texto descritivo: Qualquer caderno é prazeroso ou é cansativo ou ambos. Nenhum caderno cansativo é caderno caro. Todo caderno caro é caderno prazeroso e nenhum caro é cansativo. Há mais (ou igual) cadernos prazerosos do que cadernos caros. Todo caderno útil é caderno cansativo e nenhum útil é prazeroso.
Daqui: Algum caderno cansativo é caderno prazeroso. →V
Algum caderno prazeroso é caderno cansativo.→V

Exemplo 2 Existe uma revenda de iates atuando em nossa cidade. Ali, iates são vendidos com 3 opcionais que valorizam o produto. São eles: âncora, plataforma-saltos e circuito-tv. Em um levantamento de 42 vendas de iates, descobriu-se que: 27 tinham âncora, 27 tinham plataforma-saltos e 21 tinham circuito-tv. Já os que tinham âncora + circuito-tv eram 16. Os que tinham âncora + plataforma-saltos eram 15. Os que tinham plataforma-saltos + circuito-tv eram 11. Finalmente os 3 opcionais estavam em 7.
Pergunta-se: Quantos têm: (a) só circuito-tv?; (b) apenas âncora?; (c) só plataforma-saltos? (d) plataforma-saltos e circuito-tv mas não âncora?; (e) âncora e plataforma-saltos mas não circuito-tv?; (f) Apenas um opcional ?; (g) Pelo menos um opcional ? e (h) Nenhum opcional ? Resposta → 1 3 8 4 8 12 40 2.

Exemplo 3 As pessoas podem ser qualificadas como coxas, paranistas e atleticanas. Neste exemplo, todas as pessoas atleticanas são também coxas. Há pessoas que são só coxas e também há pessoas que são só paranistas. Há pessoas que são paranistas e coxas ao mesmo tempo, mas que não são atleticanas. Finalmente, há pessoas que são as 3 coisas. Não esqueçamos que há pessoas que não são nada. Em um universo de 48 pessoas, sabe-se que 4 são coxas. 8 são atleticanas e coxas. 8 são coxas e paranistas. 12 são apenas paranistas. Finalmente, 10 não são nada.
Pergunta-se quantas são as pessoas que têm as 3 qualidades ? → 6

Exemplo 4 Considere um problema onde há diversas preferências. Gostam só de discurso 9. Gostam só de conversa 4. Gostam só de monólogo 10. Gostam só de bate-boca 6. Gostam de discurso+conversa 10. Gostam de discurso+bate-boca 15. Gostam de discurso+conversa+bate-boca 13. Gostam de monólogo+bate-boca 14. Gostam de monólogo+conversa 8. Gostam de conversa+monólogo+bate-boca 11. Gostam de conversa+bate-boca 8.
Desenhe um diagrama de Venn e descubra: (a) Quantos gostam de discurso ? (b) Quantos gostam de conversa? (c) Quantos gostam de monólogo ? E finalmente (d) Quantos gostam de bate-boca ? → 47 54 43 67.

Exemplo 5 Imagine um grupo de pessoas que têm seus gostos (gosto não se discute). Gostam só de laranja 8. Gostam só de morango 6. Gostam só de banana 9. Gostam só de abacaxi 6. Gostam só de mimosa 9. Gostam só de coco 6. Gostam de laranja + morango 7. Gostam de morango + banana 4. Gostam de banana + abacaxi 5. Gostam de abacaxi + mimosa 5. Gostam de mimosa + coco 5. Gostam de laranja + coco 7.
Desenhe um diagrama de Venn e descubra: (a) Quantos gostam de laranja ? (b) Quantos gostam de morango? (c) Quantos gostam de banana ? (d) Quantos gostam de abacaxi? (e) Quantos gostam de mimosa ? E finalmente (f) Quantos gostam de coco ? → 22 17 18 16 19 18.

🔗 Para você fazer

1 No exercício a seguir, um determinado universo está descrito por algumas frases, a saber: Qualquer dicionário é prazeroso ou é bonito ou ambos. Todo dicionário complexo é dicionário prazeroso e nenhum complexo é bonito. Todo dicionário interessante é dicionário bonito e nenhum interessante é praze Algum dicionário bonito é dicionário prazeroso. Algum dicionário prazeroso é dicionário bonito.

A seguir, são apresentadas 10 frases que podem ser **V** ou **F** baseadas nas afirmações feitas acima. Eis as frases:

- 1. Pode haver um dicionário interessante que é dicionário prazeroso.
- 2. Pode haver um dicionário prazeroso que é dicionário bonito.
- 3. Pode haver um dicionário bonito que é dicionário prazeroso.
- 4. Não há menos dicionários bonitos do que dicionários interessantes.
- 5. Há mais (ou igual) dicionários prazerosos do que dicionários bonitos.
- 6. Nenhum dicionário prazeroso é dicionário interessante.
- 7. Nenhum dicionário bonito é dicionário complexo.
- 8. Nenhum dicionário interessante é dicionário prazeroso.
- 9. Nenhum dicionário complexo é dicionário interessante.
- 10. Algum dicionário interessante é dicionário bonito.

1	2	3	4	5	6	7	8	9	10

2 Agora, você deve calcular alguns valores pedidos.

Existe uma revenda de motocicletas atuando em nossa cidade. Ali, motocicletas são vendidos com 3 opcionais que valorizam o produto. São eles:

estepe, gps e luz-led.
Em um levantamento de 39 vendas de motocicletas, descobriu-se que: 21 tinham estepe, 20 tinham gps e 17 tinham luz-led.
Já os que tinham estepe + luz-led eram 14.
Os que tinham estepe + gps eram 10.
Os que tinham gps + luz-led eram 11.
Finalmente os 3 opcionais estavam em 9.
Pergunta-se: Quantos têm:
(a) só luz-led?; (b) apenas estepe?; (c) só gps?; (d) gps e luz-led mas não estepe?; (e) estepe e gps mas não luz-led?; (f) Apenas um opcional ?; (g) Pelo menos um opcional ? e (h) Nenhum opcional ?

a	b	c	d	e	f	g	h

3 As pessoas podem ser qualificadas como altas, gordas e jovens. Neste exemplo, todas as pessoas jovens são também altas. Há pessoas que são só altas e também há pessoas que são só gordas. Há pessoas que são gordas e altas ao mesmo tempo, mas que não são jovens. Finalmente, há pessoas que são as 3 coisas. Não esqueçamos que há pessoas que não são nada.
Em um universo de 44 pessoas, sabe-se que 6 são altas. 5 são jovens e altas. 5 são altas e gordas. 10 são apenas gordas. Finalmente, 8 não são nada.
Pergunta-se quantas são as pessoas que têm as 3 qualidades ?

4 Considere um problema onde há diversas preferências. Gostam só de laranja 5.
Gostam só de morango 14.
Gostam só de banana 11.
Gostam só de abacaxi 11.
Gostam de laranja+morango 7.
Gostam de laranja+abacaxi 14.
Gostam de laranja+morango+abacaxi 9.
Gostam de banana+abacaxi 12.
Gostam de banana+morango 15.
Gostam de morango+banana+abacaxi 12.
Gostam de morango+abacaxi 7.
Desenhe um diagrama de Venn e descubra:
(a) Quantos gostam de laranja ?
(b) Quantos gostam de morango?
(c) Quantos gostam de banana ?
E finalmente (d) Quantos gostam de abacaxi ?

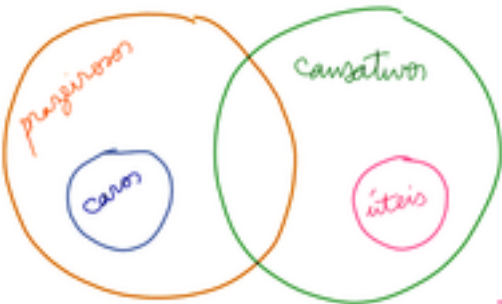
a	b	c	d

5 Imagine um grupo de pessoas que têm seus gostos (gosto não se discute). Gostam só de carro 7.
Gostam só de moto 5.
Gostam só de trem 6.
Gostam só de avião 9.
Gostam só de carona 6.
Gostam só de metrô 4.
Gostam de carro + moto 9.
Gostam de moto + trem 4.
Gostam de trem + avião 4.
Gostam de avião + carona 8.
Gostam de carona + metrô 4.
Gostam de carro + metrô 9.
Desenhe um diagrama de Venn e descubra:
(a) Quantos gostam de carro ?
(b) Quantos gostam de moto?
(c) Quantos gostam de trem ?
(d) Quantos gostam de avião?
(e) Quantos gostam de carona ?
E finalmente (f) Quantos gostam de metrô ?

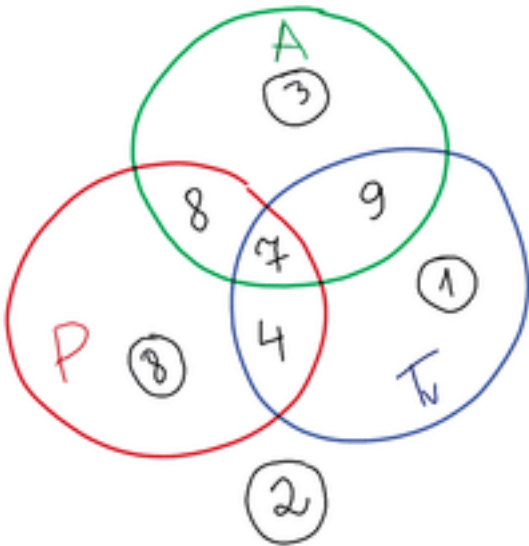
a	b	c	d	e	f



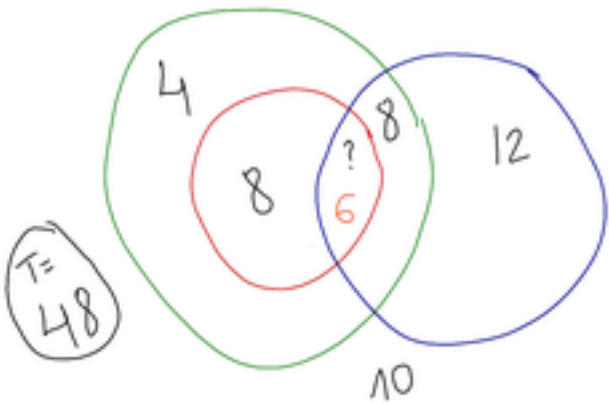
Figuras



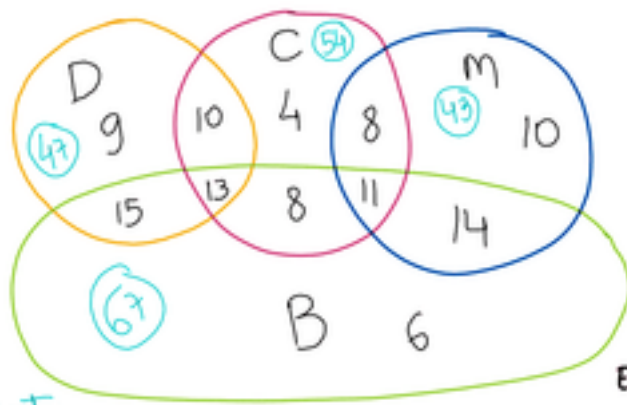
Exercício
1



Exercício
2

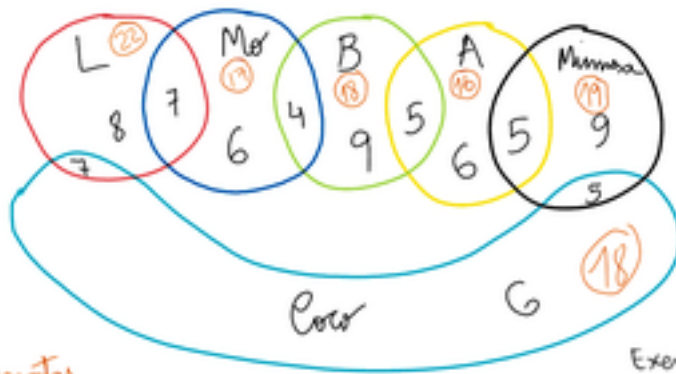


Exercício
3



Respostas

Exercício 4



Respostas

Exercício 5

Prática em Matrizes

1. Multiplicação de matrizes Considere as matrizes A e B conforme definido abaixo. Calcule a matriz produto de $A \times B$, transcreva o resultado encontrado e transcreva também o programa Python que você usou para achar o resultado.

$$A = \begin{pmatrix} 31 & 41 & -44 & 14 & 31 & -2 \\ -31 & 7 & -8 & -38 & -2 & -16 \\ -41 & 20 & 44 & -32 & 23 & 25 \\ 23 & 9 & 40 & -27 & -27 & -26 \\ 18 & 11 & -33 & -37 & 4 & -44 \end{pmatrix}$$

B =			
-40	-31	-9	-14
-38	-3	2	-20
-25	-32	50	33
31	36	50	-14
22	-33	24	7
1	18	-18	-37

$A \times B =$			

Transcreva a seguir, o programa usado:

2. Adição dos múltiplos de n Suponha a matriz D abaixo listada. Escreva um programa em Python que processe a matriz (que deve ser digitada nele) e calcule a soma dos elementos da matriz que são múltiplos de n . Transcreva o programa também. $n = 21$

D =						
74	2	708	382	982	766	594
247	957	660	642	666	546	579
464	118	144	176	859	479	658
837	540	306	427	529	463	818
20	541	400	184	963	292	747
208	721	467	149	729	214	623
533	367	744	19	912	646	771

A soma dos múltiplos de n é

Transcreva a seguir, o programa usado:

3. Matriz maluca Suponha a matriz E abaixo.

E =						
48	54	60	66	72	78	84
95	101	107	113	119	125	131
147	153	159	165	171	177	183
204	210	216	222	228	234	240
266	272	278	284	290	296	302
333	339	345	351	357	363	369
405	411	417	423	429	435	441

Escreva um programa em Python capaz de gerar a matriz E acima de maneira automatizada (i. é, sem digitar os valores).



Manipulações numéricas

A melhor maneira de aprender a programar é lendo código. (Manual do Hacker, disponível na Internet).

Nesta folha, você receberá 5 algoritmos, que podem ou não estar errados.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Abaixo de cada algoritmo estão 3 execuções corretas do mesmo. Não há a garantia de que o algoritmo mostrado é que gerou aquela saída. Lembre-se que o algoritmo pode ter tido a inclusão deliberada de um erro.

Ao final, informe quais algoritmos você considerou corretos.

Algoritmo 1 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva a cadeia 'primo' se ele for PRIMO e 'não primo' senão.

```
1: função A01PRIM (N : inteiro) : cadeia
2: M,Z,T : inteiro
3: M ← 0
4: X ← ⌈(N0.5)
5: para T de 2 a X
6:   Z ← N mod T
7:   se Z = 0
8:     M ← M + 1
9: fim{se}
10: fim{para}
11: se M = 0
12:   retorne 'primo'
13: senão
14:   retorne 'nao primo'
15: fim{se}
16: fimfunção
```

Esta função correta, deu os seguintes resultados:
A01PRIM(14) = NAOPRIMO ; A01PRIM(73) = PRIMO ; A01PRIM(31) = PRIMO .

Algoritmo 2 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva um inteiro contendo a soma dos divisores inteiros de N, excluindo-se desta soma ele mesmo e a unidade.

```
1: função A02SOMD (N : inteiro) : inteiro
2: M,T,Z : inteiro
3: M ← 0
4: para T de 2 a N-1
5:   Z ← N mod T
6:   se 0 = Z
7:     M ← M + 1
8:   fim{se}
9: fim{para}
10: retorne M
11: fimfunção
```

Esta função correta, deu os seguintes resultados:
A02SOMD(23) = 0 ; A02SOMD(66) = 77 ; A02SOMD(17) = 0 .

Algoritmo 3 Escreva uma função que receba dois inteiros N e X. Se X for maior que zero, a função deve calcular e devolver o número N^X e se não for deve devolver -1.

```
1: função A03POTE (N,X : inteiro) : inteiro
2: R : inteiro
3: R ← 1
4: se X > 0
5:   enquanto X > 0
6:     R ← R × N
7:     X ← X - 1
8:   fim{enquanto}
9:   retorne N
10: senão
11:   retorne -1
12: fim{se}
```

13: fimfunção
Esta função correta, deu os seguintes resultados:
A03POTE(3 4) = 81 ; A03POTE(4 5) = 1024 ; A03POTE(3 5) = 243 .

Algoritmo 4 Escreva uma função que receba três inteiros D, M e A representando uma data (D é o dia, M é o mês e A é o ano). O algoritmo deve devolver 1 se a data estiver errada e 0 senão.

```
1: função A04DATA (D, M, A : inteiro) : inteiro
2: biss, erro : inteiro
3: se 0 = (A mod 400)
4:   biss←1
5: senão
6:   se 0 = (A mod 100)
7:     biss←0
8:   senão
9:     se 0 = (A mod 4)
10:       biss← 1
11:     senão
12:       biss← 0
13:   fim{se}
14: fim{se}
15: fim{se}
16: erro ← 0
17: se M = 2
18:   se biss = 1
19:     se D > 28
20:       erro ← 1
21:   fim{se}
22: senão
23:   se D > 28
24:     erro ← 1
25:   fim{se}
26: fim{se}
27: senão
28:   se (M=4)∨(M=6)∨(M=9)∨(M=11)
29:     se D > 30
30:       erro ← 1
31:   fim{se}
32: senão
33:   se D > 31
34:     erro ← 1
35:   fim{se}
36: fim{se}
37: fim{se}
38: se M > 12
39:   erro ← 1
40: fim{se}
41: retorne erro
42: fimfunção
```

Esta função correta, deu os seguintes resultados:
A04DATA(34 12 7162) = 1 ; A04DATA(27 2 1940) = 0 ; A04DATA(29 2 1700) = 1 .

Algoritmo 5 Brinquedos “PIRRALHOS ENDIABRADOS” é um grande distribuidor de presentes em todo o país. Recentemente, a empresa teve a oportunidade de comprar pequenos brinquedos, todos embalados em caixas retangulares. O objetivo da compra, foi colocar cada brinquedo em uma esfera colorida, para revendê-los como surpresa, mais ou menos como o Kinder ovo. Existem esferas de raios 10, 20 e 30 cm. Cada brinquedo, tem as suas 3 dimensões A, B e C, medidas em centímetros. Escreva uma função que receba A,B,C e retorne o raio da menor esfera possível. Todos os brinquedos caberão em uma das esferas.

```
1: função A05DIAG (X, Y, Z : inteiro) : inteiro
2: real diag
3: diag ← (X2) + (Y2) + (Z2)
4: diag ← √diag
5: se diag > 40
6:   retorne 30
7: senão
8:   se diag > 40
9:     retorne 20
10:   senão
11:     retorne 10
12:   fim{se}
13: fim{se}
```

14: fimfunção
Esta função correta, deu os seguintes resultados:
A05DIAG(8 3 1) = 10 ; A05DIAG(26 25 29) = 30 ; A05DIAG(7 2 6) = 10 .

Responda

Coloque ao lado de cada algoritmo a palavra BOM (para algoritmos certos)ou RUIM (para algoritmos errados)

o algoritmo 1 está	
o algoritmo 2 está	
o algoritmo 3 está	
o algoritmo 4 está	
o algoritmo 5 está	



Vetores numéricos

A melhor maneira de aprender a programar é lendo código. (Manual do Hacker, disponível na Internet).

Nesta folha, você receberá 5 algoritmos, que podem ou não estar corretos. O erro, se existente, foi introduzido deliberadamente em um (E um só) comando do algoritmo.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chinses reverso.

Note que nem sempre tais resultados foram obtidos pelo algoritmo acima deles. O resultado numérico sempre foi obtido com o algoritmo correto.

Identifique qual(is) dos 5 algoritmos estão corretos e escreva na resposta ao final.

Algoritmo 1 Escreva uma função que receba um vetor de 10 números inteiros e devolva a posição daquele número que é o maior do vetor. Se houver dois ou mais iguais a este maior, deve-se retornar o endereço do primeiro. Por exemplo, se o vetor for 4, 7, 2, 19, 21, 8, 7, 19, 8, 6 a resposta deve ser 5, já que o 21 (o maior) é o quinto elemento. Se o vetor for 4, 12, 5, 7, 12, 7, 1, 2, 3, 8, a resposta deve ser 2.

```
1: função B01ACHM (N[10] : inteiro) : inteiro
2: MAI,QUA,I : inteiro
3: I ← 1
4: MAI ← -99999
5: enquanto I < 11
6:   se N[I] > MAI
7:     MAI ← N[I]
8:     QUA ← I
9:   fim{se}
10:  I ← I + 1
11: fim{enquanto}
12: retorne QUA
13: fimfunção
```

Esta função correta, deu os seguintes resultados:

B01ACHM(4 5 1 7 4 4 4 1 3 7) = 4 ;
B01ACHM(2 17 5 14 20 18 15 19 16 13) = 5 ;
B01ACHM(2 4 4 1 3 4 4 3 5 1) = 9 .

Algoritmo 2 Escreva uma função que receba um número K (inteiro, não é preciso testar) e um vetor N de 10 inteiros e devolva a posição de K em N. Se houver mais de 1 valor de K em N, deve retornar o primeiro. Se não houver K em N, retornar -1.

Por exemplo, Se K=8 e N=1 3 5 7 9 2 4 6 8 10, a resposta deve ser 9. Se K=7 e N é o mesmo, a resposta é 4. Se K=13 e N é o mesmo, a resposta é -1.

```
1: função B02ACHKK (K, N[10] : inteiro) : inteiro
2: inteiro RES,I
3: RES ← -1
4: I ← 1
5: enquanto I < 11
6:   se N[I] = K
7:     RES ← K
8:     I ← 11
9:   fim{se}
10:  I ← I + 1
11: fim{enquanto}
12: retorne RES
13: fimfunção
```

Esta função correta, deu os seguintes resultados:

B02ACHK(13,16 29 28 20 25 13 23 15 26 14)=6 ;
B02ACHK(28,22 15 30 28 20 23 21 25 16 17)=4 ;
B02ACHK(17,14 26 13 27 17 37 33 20 11 34)=5 .

Algoritmo 3 Escreva uma função que receba um vetor N de 10 elementos inteiros, em princípio sem nenhuma ordem (ou seja, desordenado) e uma chave K também inteira e devolva a quantidade de vezes que K aparece em N.

Por exemplo, se N=1 3 5 7 2 3 4 5 6 10 e K=5 a resposta é 2. Se K=18, a resposta é 0 e se K=1, a resposta é 1.

```
1: função B03CONK (K,N[10] : inteiro) : inteiro
2: inteiro QTD,I
3: QTD ← 0
4: para I de 1 a 11 passo 1
5:   se N[I] = K
6:     QTD ← QTD + 1
7:   fim{se}
8: fim{para}
9: retorne QTD
10: fimfunção
```

Esta função correta, deu os seguintes resultados:

B03CONK(18,13 14 18 17 15 12 12 18 14 15)=2 ;
B03CONK(16,17 13 14 15 18 16 15 18 13 12)=1 ;
B03CONK(17,20 26 31 21 34 40 15 37 13 28)=0 .

Algoritmo 4 Escreva uma função que receba um vetor ordenado de 11 inteiros. A função deve devolver 3 valores (um real e dois inteiros), a saber:

média A soma dos elementos dividido por 11

moda o valor mais freqüente no vetor

mediana O valor do meio (neste caso o N[6])

```
1: função B04ESTA (N[11] : inteiro) : real,inteiro,inteiro
2: MED : real
3: I,J,MOO,MAN,SOM,ASA,MAI,CTDR[11] : inteiro
4: I ← 1
5: SOM ← 0
6: enquanto I < 12
7:   SOM ← SOM + N[I]
8:   I ← I + 1
9: fim{enquanto}
10: MED ← SOM ÷ 11
11: MAN ← N[6]
12: para I de 1 a 11
13:   SOM ← 0
14:   para J de 1 a 11
15:     se N[I] = N[J]
16:       SOM ← SOM + 1
17:   fim{se}
18:   fim{para}
19:   CTDR[I] ← SOM
20: fim{para}
21: MAI ← -9999
22: para J de 1 a 11
23:   se CTDR[J] > SOM
24:     MAI ← CTDR[J]
25:     ASA ← J
26:   fim{se}
27: fim{para}
28: MOO ← N[ASA]
29: retorne MED,MOO,MAN
30: fimfunção
```

Esta função correta, deu os seguintes resultados:

B04ESTA(1 1 2 2 2 3 4 5 5 5 8) = 3.5 2 3 ;
B04ESTA(1 2 3 3 3 4 6 6 6 6 7) = 4.3 6 4 ;
B04ESTA(2 2 2 3 3 4 5 5 5 8 8) = 4.3 2 4 .

Algoritmo 5 O vetor está em ordem ? Escreva o algoritmo de uma função que receba um vetor de 10 inteiros e devolva 0 se o vetor está em ordem crescente e 1 senão. Um vetor V está em ordem crescente para quaisquer i e j, sendo que j > i, é válida a expressão V[j] ≥ V[i]. Note a presença da igualdade na condição. Além disso, essa expressão tem que ser válida para todos os pares dentro do vetor.

Por exemplo se o vetor for 1,3,4,5,6,7,9,10,16,21 a resposta é 0. Se o vetor for 1,3,4,5,6,8,7,10,16,21 a resposta é 1 e se for 2,2,2,2,2,2,2,2,2 a resposta é 0.

```
1: função B05ESORN (N[10] : inteiro) : inteiro
2: inteiro I, RES
3: RES ← 0
4: I ← 1
5: enquanto I < 10
6:   se N[I] > N[I+1]
7:     RES ← 1
8:   fim{se}
9:   I ← I + 1
10: fim{enquanto}
11: retorne I
12: fimfunção
```

Esta função correta, deu os seguintes resultados:

B05ESOR(5 9 28 43 57 69 71 73 83 90) = 0 ;
B05ESOR(18 25 37 41 54 58 40 74 86 87) = 1 ;
B05ESOR(7 14 26 57 62 73 31 79 81 82) = 1 .

Responda

Coloque ao lado de cada algoritmo a palavra BOM (para algoritmos certos)ou RUIM (para algoritmos errados)

o algoritmo 1 está	
o algoritmo 2 está	
o algoritmo 3 está	
o algoritmo 4 está	
o algoritmo 5 está	



Vetores alfanuméricos

A melhor maneira de aprender a programar é lendo código. (Manual do Hacker, disponível na Internet).

Nesta folha, você receberá 5 algoritmos que podem estar certos ou podem estar errados. Se errados, o erro deriva-se de uma (e apenas uma) linha errada no algoritmo.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chinês reverso.

Nem sempre o resultado gerado o foi pelo algoritmo que está imediatamente acima. Só se o algoritmo estiver correto.

Identifique qual(is) dos 5 algoritmos estão corretos escrevendo na resposta no final da folha.

Note que todos os algoritmos desta folha têm a origem dos índices em 1 (e não em zero).

Algoritmo 1 Escreva uma função que receba uma frase (cadeia de até 80 caracteres, terminada por um ponto) e devolva 1 se a frase for um palíndromo e 0 senão. Um palíndromo é aquela frase que pode ser lida tanto da direita para a esquerda como ao contrário. Exemplos:

roma me tem amor
socorram me subi no ô nibus em marrocos
sairam o tio e oito marias
ovo, rotor, radar
me vê se a panela da moça é de aço madalena paes e vem
a diva da vida
luz azul
ato idiota
o treco certo
a base do teto desaba
atila toledo mata modelo talita
anotaram a data da maratona
a droga da gorda
o romano acata amores a damas amadas e roma ataca o namoro
sá dá tapas e sapatadas
em inglês: a man, a plan, a cat, a canal, panama
murder for a jar of red rum
madam I'm Adam
doc, note. I dissent. A fast never prevents a fatness. I diet on cod
em francês: esope reste ici et se repose

Observação: note que as posições do espaço em branco não são as mesmas na ida e na volta. Também devem ser desprezados os caracteres eventualmente acentuados, bem como o “ç”.

```
1: função C01PALI (X[80] : caracter) : inteiro
2: inteiro INI,FIM,EPAL,TAM
3: TAM ← 1
4: enquanto X[TAM] ≠ ''
5:   TAM ← TAM + 1
6: fim{enquanto}
7: INI ← 1
8: FIM ← TAM-1
9: EPAL ← 1
10: enquanto INI < FIM
11:   enquanto X[INI] = ''
12:     INI ← INI + 1
13:   fim{enquanto}
14:   enquanto X[FIM] = ''
15:     FIM ← FIM - 1
16:   fim{enquanto}
17:   se X[INI] ≠ X[FIM]
18:     EPAL ← 0
19:   fim{se}
20:   INI ← INI+1
21:   FIM ← FIM-1
22: fim{enquanto}
23: retorne EPAL
24: fimfunção
```

Esta função correta, deu os seguintes resultados:

C01PALI(ROMA AMOR.) = 1 ; C01PALI(MAN A PLAN A CAT A CANAL PANAMA.) = 1 ; C01PALI(XUCRICUNICO.) = 0 .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma chave K (Inteiro) e um vetor V (de 10 inteiros) crescente e devolva outro vetor de 11 inteiros, também crescente onde a chave K foi incluída em seu local correto.

```
1: função C02INCL (K, V[10] : inteiro) : R[11]
   inteiro
2: inteiro I,J
3: para I de 1 a 10
4:   R[I] ← V[I]
5: fim{para}
6: I ← 1
7: enquanto (R[I] ≤ K)^(I<11)
8:   I ← I+1
9: fim{enquanto}
10: J ← 10
11: enquanto J ≥ I
12:   R[J+1] ← R[J]
13:   J ← J-1
14: fim{enquanto}
15: R[J+1] ← K
16: retorne R
17: fimfunção
```

Esta função correta, deu os seguintes resultados:

C02INCL(60,7 14 21 28 35 42 49 56 63 70)=7 14 21 28 35 42 49 56 60 63 70 ; C02INCL(189,6 12 18 24 30 36 42 48 54 60 189) ; C02INCL(7,16 22 28 34 40 46 52 58 64 70)=7 16 22 28 34 40 46 52 58 64 70 .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma frase (cadeia de 80 caracteres, o último útil é um ponto) e devolva o número da palavra que é a maior da frase. Se houver empate (mais de uma palavra com o mesmo comprimento), devolver o número da primeira. Por exemplo, na frase "IVO VIU A UVA", a maior palavra tem 3 letras é como há empate a resposta deve ser 1. Na frase "OUIRAM DO IPIRANGA AS MARGENS" a resposta deve ser 3.

```
1: função C03PALM (X[80] : caracter) : inteiro
2: TMS[40] : inteiro
3: I,J,TAM,CTP,MAI,QUA : inteiro
4: TAM ← 1
5: enquanto X[TAM] ≠ ''
6:   TAM ← TAM+1
7: fim{enquanto}
8: CTP ← 1
9: I ← 1
10: J ← 0
11: enquanto I ≤ TAM
12:   se X[I] = ' '
13:     TMS[CTP] ← I - J
14:     TMS[CTP] ← TMS[CTP]-1
15:     CTP ← CTP+1
16:     J ← I
17:   fim{se}
18:   I ← I + 1
19: fim{enquanto}
20: TMS[CTP] ← I - J
21: TMS[CTP] ← TMS[CTP] - 2
22: MAI ← -99999
23: para I de 1 a CTP
24:   se TMS[I] > MAI
25:     MAI ← TMS[I]
26:     QUA ← I
27:   fim{se}
28: fim{para}
29: retorne QUA
30: fimfunção
```

Esta função correta, deu os seguintes resultados:

C03PALM(MAS INSINUOU SE QUE O CORPO.)=2 ; C03PALM(NO PRESENTE CASO PARECE ME.)=2 ; C03PALM(MAS SE NAO POSSO DEMOS IMAGINAR.)=5 .

Algoritmo 4 Escreva o algoritmo de uma função que receba uma chave K (inteiro) e um vetor

V de 10 inteiros. A função deve excluir de V todas as ocorrências (se houver) de K, incluindo zeros ao final se necessário. Por exemplo, se chamada com 22 e 1 7 21 22 23 24 28 29 22 90 deve responder 1 7 21 23 24 28 29 90 0 0.

```
1: função C04EXCL (K, V[10] : inteiro) : R[10]
   inteiro
2: I, J : inteiro
3: I ← 1
4: J ← 1
5: enquanto I<11
6:   se V[I] ≠ K
7:     R[J] ← V[I]
8:     J ← J+1
9:   fim{se}
10:   I ← I+1
11: fim{enquanto}
12: enquanto J < 11
13:   R[J] ← 0
14:   J ← J+1
15: fim{enquanto}
16: retorne R
17: fimfunção
```

Esta função correta, deu os seguintes resultados:

C04EXCL(3,7 6 6 5 7 3 4 5 7 6) = 7 6 6 5 7 4 5 7 6 0 ; C04EXCL(3,48 42 14 20 43 29 11 9 19 44) = 48 42 14 20 43 29 11 9 19 44 ; C04EXCL(11,1 7 4 8 6 6 6 7 5 2) = 1 7 4 8 6 6 6 7 5 2 .

Algoritmo 5 Esta função deve receber um vetor de 10 inteiros e devolver a amplitude entre os seus elementos. A amplitude de um vetor é calculada fazendo-se o maior elemento menos o menor elemento.

```
1: função C05AMPL (X[10] : inteiro) : inteiro
2: inteiro I,MAI,MEN,AMP
3: MAI ← +9999
4: MEN ← +9999
5: para I de 1 a 10
6:   se X[I] > MAI
7:     MAI ← X[I]
8:   fim{se}
9:   se X[I] < MEN
10:     MEN ← X[I]
11:   fim{se}
12: fim{para}
13: AMP ← MAI - MEN
14: retorne AMP
15: fimfunção
16:
```

Esta função correta, deu os seguintes resultados:

C05AMPL(86 45 90 96 4 43 18 28 74 22) = 92 ; C05AMPL(1 6 6 3 2 5 1 8 7 1) = 7 ; C05AMPL(21 72 95 76 27 29 36 40 90 5) = 90 .

Responda

Coloque ao lado de cada algoritmo a palavra BOM (para algoritmos certos) ou RUIM (para algoritmos errados)

o algoritmo 1 está	
o algoritmo 2 está	
o algoritmo 3 está	
o algoritmo 4 está	
o algoritmo 5 está	



Manipulação de matrizes

A melhor maneira de aprender a programar é lendo código. (Manual do Hacker, disponível na Internet).

Nesta folha, você receberá 5 algoritmos que podem ou não estar corretos. Se estiverem errados, o erro deve-se a uma (e apenas uma) linha errada.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chins reverso.

Não há garantia de que o resultado mostrado seja resultado do algoritmo acima. Só será se o algoritmo estiver certo.

Identifique qual(is) dos 5 algoritmos estão corretos na resposta.

Algoritmo 1 Escreva o algoritmo de uma função que receba uma matriz de 9 inteiros (3 linhas por 3 colunas), totalize as colunas e devolva um vetor de 3 totais.

```
1: função D01VESO (M [1..3] [1..3] : inteiro)
  V[1..3]:inteiro
2: I,J,SOM : inteiro
3: V ← 0,0,0
4: para J de 1 a 3
5:   SOM ← 0
6:   para I de 1 a 3
7:     SOM ← SOM + M[I][J]
8:   fim{para}
9:   V[J] ← SOM
10: fim{para}
11: retorne V
12: fimfunção
```

Esta função correta, deu os seguintes resultados:

D01VESO (7 5 5,8 2 6,8 9 1) = 23 16 12 ;
D01VESO (1 1 2,7 6 6,5 5 4) = 13 12 12 ; D01VESO (3 7 7,5 8 7,9 5 2) = 17 20 16 .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 4 colunas de números inteiros. A função deve gerar um vetor de 4 números reais. O elemento k da resposta é calculado como a soma dos elementos da coluna k , que estão acima da linha k , dividido pela soma dos elementos que estão na coluna k e nas linhas $k, k+1, \dots, 4$.

```
1: função D02MSMI (M[1..4] [1..4] : inteiro)
  V[1..4]:real
2: J,NUM,DEN,I : inteiro
3: para J de 1 a 4
4:   NUM ← 0
5:   DEN ← 0
6:   para I de 1 a J-1
7:     NUM ← NUM + M[I][J]
8:   fim{para}
9:   para I de J a 4
10:    DEN ← DEN + M[I][J]
11:   fim{para}
12:   V[J] ← I ÷ DEN
13: fim{para}
14: retorne V
15: fimfunção
```

Esta função correta, deu os seguintes resultados:

D02MSMI (15 3 12 18, 8 6 19 20,13 13 12 6, 7 5 15 10)= .0 .1 1.1 4.4 ; D02MSMI (17 20 5 11, 3 10 16 16, 5 15 12 12, 9 13 14 20)= .0 .5 .8 2.0 ;
D02MSMI (19 17 14 3,18 20 2 10,15 19 8 11,17 8 1 20)= .0 .4 1.8 1.2 .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 5 colunas contendo inteiros e devolva um vetor de 4 inteiros que contém o menor valor em cada linha.

```
1: função D03MEHO (M[1..4][1..5]:inteiro)
  V[1..4]:inteiro
2: I,J,MEN : inteiro
3: para I de 1 a 4
4:   MEN ← 999999
5:   para J de 1 a 5
6:     se M[I][J] < MEN
7:       MEN ← M[I][J]
8:   fim{se}
9:   fim{para}
10:  V[I] ← MEN
11: fim{para}
12: retorne V
13: fimfunção
```

Esta função correta, deu os seguintes resultados:

D03MEHO (4 9 8 3 2,4 8 5 3 7,4 4 4 4 2,3 7 10 1 5)=2 3 2 1 ; D03MEHO (2 6 6 5 2,5 1 4 6 6,9 6 3 2 5,6 1 9 9 2)=2 1 2 1 ; D03MEHO (8 3 3 9 6,2 5 5 5 1,9 7 5 10 10,3 7 9 8 6)=3 1 5 3 .

Algoritmo 4 Escreva uma função que receba duas matrizes conformáveis, isto é a primeira de 3 linhas por 4 colunas e a segunda de 4 linhas por 3 colunas. A função deve multiplicar as duas matrizes, gerando o resultado que terá 3 linhas por 3 colunas.

```
1: função D04MUMA (A[1..3][1..4]:inteiro,
  B[1..4][1..3]:inteiro)
2: V[1..3][1..3]:real // até aqui é o cabeçalho...
3: I,J,K,SOM:inteiro
4: para I de 1 a 3
5:   para J de 1 a 3
6:     SOM ← 0
7:     para K de 1 a 4
8:       SOM ← SOM+(A[I][J]×B[K][J])
9:     fim{para}
10:    V[I][J] ← SOM
11:  fim{para}
12: fim{para}
13: fim{para}
14: retorne V
15: fimfunção
```

Esta função correta, deu os seguintes resultados:

D04MUMA (6 1 6 4,5 0 0 1,4 4 3 1;4 0 6,4 3 6,3 6 6,4 4 6) = 62 55 102,24 4 36,45 34 72 ;
D04MUMA (1 6 2 5,5 5 4 3,0 1 1 3;0 6 5,6 5 0,4 3 0,5 4 0) = 69 62 5,61 79 25,25 20 0 ; D04MUMA (4 0 6 6,0 1 0 1,0 1 0 0;6 5 1,6 0 5,2 1 5,6 6 0) = 72 62 34,12 6 5, 6 0 5 .

Algoritmo 5 Escreva o algoritmo de uma função que receba uma matriz de 6 linhas por 2 colunas contendo o resultado de uma eleição. A primeira coluna corresponde ao número do candidato, sendo que nem todas as linhas estarão preenchidas. Neste caso as linhas não usadas conterão zeros. Existem dois códigos em especial que estarão sempre presentes e que são 998=votos em branco e 999=votos nulos. A segunda coluna contém a quantidade de votos a cada candidato e também brancos e nulos.

A função deve devolver o número do candidato eleito, ou -1 se a eleição não for válida. O eleito é aquele que obteve o maior número de votos se a eleição foi válida. Uma eleição é válida quando o total de votos dados a candidatos mais os votos em branco corresponderem a 50% mais 1 voto do total de votantes. Se o vencedor for o branco ou o nulo, vence o candidato da linha 1 (por hipótese é o mais velho).

```
1: função D05ELEI (VOT[1..6][1..2]:inteiro)
2: VAL,I,NUL,TOT,MAI,QUEM:inteiro
3: VAL ← 0
4: para I de 1 a 6
5:   se VOT[I][1] ≠ 999
6:     VAL ← VAL + VOT[I][2]
7:   senão
8:     NUL ← VOT[I][2]
9:   fim{se}
10: fim{para}
11: TOT ← VAL ÷ 2
```

```
12: se NUL > TOT
13:   retorne -1
14: senão
15:   MAI ← -999999
16:   para I de 1 a 6
17:     se VOT[I][2] > MAI
18:       MAI ← VOT[I][2]
19:   QUEM ← VOT[I][1]
20:   fim{se}
21:   fim{para}
22: se (MAI = 998) ∨ (MAI = 999)
23:   retorne 1
24: senão
25:   retorne QUEM
26:   fim{se}
27: fim{para}
28: fimfunção
```

Esta função correta, deu os seguintes resultados:

D05ELEI (70 140, 80 160, 40 280, 0 0,998 80,999 210) = 40 ; D05ELEI (60 100, 20 1600, 0 0, 0 0,998 120,999 330) = 20 ; D05ELEI (50 160, 40 360, 30 280, 70 220,998 380,999 30) = 1 .

Responda

Coloque ao lado de cada algoritmo a palavra BOM (para algoritmos certos) ou RUIM (para algoritmos errados)

o algoritmo 1 está	
o algoritmo 2 está	
o algoritmo 3 está	
o algoritmo 4 está	
o algoritmo 5 está	



Exercícios:

- Escreva um algoritmo que leia uma matriz 10×10 e ache o maior valor, bem como sua linha e coluna.
- Escreva um algoritmo que leia uma matriz 10×10 e ache a soma dos elementos da diagonal principal e também da diagonal secundária.
- Escreva um algoritmo que leia uma matriz 10×10 e também um valor qualquer e informe se este valor está ou não na matriz.
- Escreva um algoritmo que leia uma matriz 10×10 e também um valor qualquer (que está na matriz, pode ter certeza) e imprima os vizinhos (até 8) deste valor.

Totalização de matrizes É comum ter-se uma matriz de valores referentes a um dado fenômeno e pretender-se obter totais em qualquer uma das duas dimensões. Acompanhe: Seja a matriz M composta pelo resultado das vendas de 8 filiais (8 colunas) nos 6 dias úteis (linhas) da semana passada. Pretende-se responder algumas perguntas:

- Qual a maior filial ?;
- qual o dia da semana de maior movimento?;
- Qual a amplitude de filiais ? (a maior menos a menor);
- qual a amplitude de dias ?
- Quantos dias respondem por 50% das vendas ? etc etc.

Para responder a perguntas como essas, há que se criar um vetor de 8 valores (soma das filiais) e um de 6 valores para a soma dos dias. Acompanhe o algoritmo

```
1: função SOMA (VENDAS[6][8]:real) : (VF[8], VD[6]:real)
2: inteiro J, K
3: VF ← 0
4: VD ← 0
5: para J de 1 até 6
6:   para K de 1 até 8
7:     VD[J] ← VD[J] + VENDAS[J][K]
8:     VF[K] ← VF[K] + VENDAS[J][K]
9:   fim{para}
10: fim{para}
11: fim {função}
```

Zerando a matriz abaixo da diagonal principal Este é um problema típico em teoria de grafos. A matriz de adjacência de um digrafo (grafo direcionado) usa todas as linhas da matriz. Se este grafo passar a ser considerado como não direcionado, a parte abaixo da diagonal principal precisa ser zerada. Eis o algoritmo que o faz:

```
1: função ZERAB (M[i][i]:real) : (MZ[i][i]:real)
2: inteiro L, C
3: para L de 1 até i
4:   para C de 1 até i
5:     se C < L
6:       MZ[L][C] ← 0
7:   senão
8:     MZ[L][C] ← M[L][C]
9:   fim{se}
10: fim{para}
11: fim{para}
12: devolva MZ
13: fim{função}
```

multiplicação matricial Este é um algoritmo famoso na matemática. Dadas as matrizes A (de i linhas por j colunas) e B (de k linhas por m colunas), onde $j = k$, obter a matriz C, multiplicação de A por B, de i linhas por m colunas, a partir de $C[i][m] = \sum_{x=1}^j A[i][x] \times B[x][m]$

Para implementá-lo, tem-se o

```
1: função MM (A[i][j], B[j][m]:real) : (C[i][m]:real)
2: inteiro AUX
3: inteiro IND1, IND2, IND3
4: para IND1 = 1 até i
5:   para IND2 = 1 até m
6:     AUX ← 0
7:     para IND3 = 1 até j
8:       AUX ← AUX + A[IND1][IND3] × B[IND3][IND2]
9:   fim{para}
10:  C[IND1][IND2] ← AUX
11: fim{para}
12: fim{para}
13: devolva C
14: fim{função}
```

Transposta Supondo a matriz M de i linhas por j colunas, a matriz transposta de M, reconhecida como $\phi M = M'$ é aquela obtida rotacionando-se seus elementos. M' terá j linhas por i colunas. Acompanhe o algoritmo

```
1: função ACHATRANS (M[i][j]:real) : (TRA[j][i]:real)
2: inteiro K, L
3: para K de 1 até i
4:   para L de 1 até j
5:     TRA[L][K] ← M[K][L]
6:   fim{para}
7: fim{para}
8: devolva TRA
9: fim{função}
```

Solução de sistema de equações lineares Seja A a matriz de coeficientes e seja B a matriz coluna de termos independentes em um sistema $Ax + B = 0$. O parâmetro do algoritmo é uma matriz de n linhas e $n + 1$ colunas, onde a matriz coluna B ocupa a $(n + 1)$ -ésima coluna de A.

```
1: função RSEL (A[n][n+1]:real) : (R[n]:real)
2: real OLA
3: inteiro I, J, K
4: I ← 1
5: enquanto I < n
6:   J ← n
7:   enquanto J > I
8:     se A[J-1][I] = 0
9:       troque as linhas A[J] e A[J-1]
10:    fim{se}
11:    se A[J-1][I] = 0
12:      OLA ← 0
13:    senão
14:      OLA ← -A[J][I] ÷ A[J-1][I]
15:    fim{se}
16:    para K de 1 até n+1
17:      A[J][K] ← A[J][K] + (A[J-1][K] × OLA)
18:    fim{para}
19:    J ← J - 1
20:  fim{enquanto}
21:  I ← I + 1
22: fim{enquanto}
23: I ← n
24: enquanto I ≥ 1
25:  R[I] ← A[I][I+1] ÷ A[I][I]
26:  para K de 1 até n
27:    A[K][I] ← A[K][I+1] - (A[K][I] × R[I])
28:  fim{para}
29:  I ← I - 1
30: fim{enquanto}
31: devolva R
```

32: fim{função}

Este algoritmo, pela sua complexidade vale alguma explicação. Seja o seguinte sistema:

$$\begin{array}{rrrr} a.x & + & b.y & + & c.z & = & M \\ d.x & + & e.y & + & f.z & = & N \\ g.x & + & h.y & + & i.z & = & P \end{array}$$

O objetivo inicial é zerar g , e para isso, a nova linha (3) passará a ser a linha (3) somada com a linha (2) esta devidamente multiplicada por $-\frac{g}{d}$ e fica

$$\begin{array}{rrrr} a.x & + & b.y & + & c.z & = & M \\ d.x & + & e.y & + & f.z & = & N \\ 0.x & + & h'.y & + & i'.z & = & P' \end{array}$$

Depois, o objetivo é zerar d e para isso a nova linha (2) passa a ser a linha (2) somada com a linha (1) esta devidamente multiplicada por $-\frac{d}{a}$ e fica

$$\begin{array}{rrrr} a.x & + & b.y & + & c.z & = & M \\ 0.x & + & e'.y & + & f'.z & = & N' \\ 0.x & + & h'.y & + & i'.z & = & P' \end{array}$$

Finalmente, zera-se h' e para isso a linha (3) passa a ser a linha (3) somada com a linha (2) esta devidamente multiplicada por $-\frac{h'}{e'}$ e fica

$$\begin{array}{rrrr} a.x & + & b.y & + & c.z & = & M \\ 0.x & + & e'.y & + & f'.z & = & N' \\ 0.x & + & 0.y & + & i''.z & = & P'' \end{array}$$

Note que neste ponto, z pode ser calculado, fazendo-se $z = \frac{P''}{i''}$.

Conhecido z , a coluna 3 da matriz pode ser toda calculada e jogada para a coluna do termo independente. Ao fazer isto, passa-se a ter um sistema de 2 equações e 2 incógnitas, x e y .

Agora o processo se repete até o final. Vamos ver um exemplo disto. Seja o sistema

$$\begin{array}{rrrr} 2x & + & 3y & + & 1z & = & 11 \\ 5x & + & -2y & + & 3z & = & 10 \\ 2x & + & y & + & -z & = & 1 \end{array}$$

Ao se zerar a primeira coluna

$$\begin{array}{rrrr} 2x & + & 3y & + & 1z & = & 11 \\ & + & -9.5y & + & 0.5z & = & -17.5 \\ & + & 1.8y & + & -2.2z & = & -3 \end{array}$$

Zerando a segunda coluna, fica:

$$\begin{array}{rrrr} 2x & + & 3y & + & 1z & = & 11 \\ & -9.5y & + & 0.5z & = & -17.5 \\ & & -2.11z & = & -6.33 \end{array}$$

Com isto, recupera-se o valor $z = 3$, que aplicado no sistema,

$$\begin{array}{rrrr} 2x & + & 3y & = & 8 \\ & -9.5y & = & -19 \end{array}$$

E daqui, sai $y = 2$. Aplicando-o fica

$$2x = 2 \text{ E finalmente } x = 1.$$

🔗 Para você fazer

Resolva (a mão ou via computador, você decide ;-)) o seguinte sistema de 8 equações lineares a 8 incógnitas:

$$\begin{array}{rrrrrrrrrr} -1 & 2 & 1 & 1 & 2 & 2 & 1 & 3 & & 53 \\ 2 & 2 & 4 & -3 & 3 & 2 & -2 & 1 & & 75 \\ 2 & 1 & 1 & 1 & 1 & 2 & 3 & 1 & & 64 \\ 3 & -3 & 1 & 1 & -2 & 2 & 2 & 2 & & 22 \\ -2 & 1 & 4 & 2 & 3 & 5 & 5 & 1 & & 73 \\ 2 & 2 & -2 & 2 & 1 & 2 & -1 & 2 & & 49 \\ -2 & 3 & 5 & 5 & 1 & 1 & 2 & -1 & & 71 \\ -2 & 1 & 1 & 1 & 1 & 3 & 1 & 3 & & 30 \end{array}$$

E, informe o valor das 8 variáveis encontradas:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

$$\begin{array}{rrrrrrrr} \text{|||||} & \text{|||||} & \text{|||||} & & & & & \\ & & & & & & & -1 - / \end{array}$$

Exercícios Para todos os exercícios desta folha considere o enunciado (programa) descrito. Siga-o, ou melhor, implemente-o. Responda com o valor que o programa vai imprimir. Observação importante é que todos os algoritmos estão com a **origem dos índices em 1**.

Exercício 1

```
1: funcao EXERC1
2: MA : matriz[1..5][1..6] de inteiro
3: J,K,AUX : inteiro
4: AUX ← 10
5: J ← 5
6: enquanto J > 0
7:   K ← 6
8:   enquanto K > 0
9:     MA[J][K] ← AUX
10:    AUX ← AUX + 2
11:    K --
12:  fim{enquanto}
13:  J --
14: fim{enquanto}
15: imprima MA[5][4]+MA[1][5]+MA[2][2]
```

Exercício 2

```
1: funcao EXER2
2: M[1..6][1..7] de inteiros
3: SOMA,DELTA,AUX,J,K : inteiro
4: AUX ← -83
5: SOMA ← DELTA ← 0
6: para J=1;J<7;J++
7:   para K=1;K<8;K++
8:     M[J][K] ← AUX+DELTA
9:     AUX ← AUX + DELTA + 6
10:   fim{para}
11:   DELTA ← DELTA + 1
12: fim{para}
13: SOMA ← SOMA + M[4][3]
14: SOMA ← SOMA + M[4][4]
15: SOMA ← SOMA + M[4][5]
16: SOMA ← SOMA + M[5][3]
17: SOMA ← SOMA + M[5][5]
18: SOMA ← SOMA + M[6][3]
19: SOMA ← SOMA + M[6][4]
20: SOMA ← SOMA + M[6][5]
21: imprima SOMA
```

Exercício 3

```
1: funcao EXER3
2: MX[1..5][1..5] : inteiro
3: Z : inteiro
4: MX[1][] = {2,5,2,1,4}
5: MX[2][] = {5,1,3,3,5}
6: MX[3][] = {2,1,3,2,3}
7: MX[4][] = {4,3,2,2,5}
8: MX[5][] = {1,4,3,2,3}
9: Z ← MX[MX[MX[4][4]][3]][MX[2][4]]
10: Z ← Z + MX[MX[5][5]][MX[3][3]]
11: imprima Z × MX[MX[1][1]][MX[2][5]]
```

Exercício 4

```
1: funcao EXER4
2: M[1..6][1..8] de inteiros = 19 // todos iguais
3: L,C,T,R,SOMA : inteiro
4: T ← 12; R ← 3
5: SOMA ← 0
6: para L=3;L<5;L++
7:   M[L][4] ← T; T=T+R
8: fim{para}
9: para L=4;L>1;L--
10:   M[L][5] ← T; T=T+R
```

```
11: fim{para}
12: para C=4;C>2;C--
13:   M[2][C] ← T; T=T+R
14: fim{para}
15: para L=3;L<6;L++
16:   M[L][3] ← T; T=T+R
17: fim{para}
18: para C=4;C<7;C++
19:   M[5][C] ← T; T=T+R
20: fim{para}
21: para L=4;L>0;L--
22:   M[L][6] ← T; T=T+R
23: fim{para}
24: para C=5;C>1;C--
25:   M[1][C] ← T; T=T+R
26: fim{para}
27: para L=2;L<7;L++
28:   M[L][2] ← T; T=T+R
29: fim{para}
30: para C=3;C<8;C++
31:   M[6][C] ← T; T=T+R
32: fim{para}
33: para L=5;L>0;L--
34:   M[L][7] ← T; T=T+R
35: fim{para}
36: para C=1;C<9;C++
37:   SOMA ← SOMA + M[3][C]
38: fim{para}
39: para L=1;L<7;L++
40:   SOMA ← SOMA - M[L][5]
41: fim{para}
42: imprima SOMA
```

Exercício 5

```
1: funcao EXER5
2: M[1..5][1..6] de inteiros
3: L,C,T : inteiro
4: T ← 3
5: para L=1;L<6;L++
6:   para C=1;C<7;C++
7:     M[L][C] ← T+(2 × L)+C
8:   fim{para}
9: fim{para}
10: imprima M[5][4]
```

Exercício 6 Neste exercício, você deve criar uma matriz de 6 linhas por 6 colunas e deve colocar no elemento linha=1, coluna=1 o valor -3. O preenchimento deve seguir na maneira normal da escrita (primeiro a linha 1 e nela todas as colunas, depois a linha 2, todas as colunas e assim por diante). A cada novo elemento preenchido adicionar 3 ao valor anterior. Depois de preencher toda a matriz, somar os valores que estão na diagonal principal e também os valores que estão na diagonal secundária. Ao final, imprimir o valor desta soma.

Exercício 7 Este exercício calcula a amplitude de cada linha e depois imprime a menor amplitude de todas.

```
1: funcao EXER7
2: M[1..5][1..10] de inteiro
3: L,C,MAXL,MINL,MI : inteiro
4: M[1][] = {68,25,93,39,72,24,69,29,38,74}
5: M[2][] = {14,28,26,56,77,91, 7,96,75,98}
6: M[3][] = {52,81,61,37,22,63,34, 1,19,80}
7: M[4][] = {11,79,10,35,84,89,44, 2,71,67}
8: M[5][] = {53,65, 4,31,30,64,16,23,90,92}
9: MI ← +99999
10: para L=1;L<6;L++
11:   MAXL ← -99999
12:   MINL ← +99999
13:   para C=1;C<11;C++
14:     se M[L][C]> MAXL
15:       MAXL ← M[L][C]
16:   fim{se}
17:   se M[L][C]< MINL
18:     MINL ← M[L][C]
```

```
19:   fim{se}
20:   fim{para}
21:   se (MAXL-MINL) < MI
22:     MI ← MAXL-MINL
23:   fim{se}
24: fim{para}
25: imprima MI
```

Exercício 8 Este algoritmo é a multiplicação matricial tal e qual. Portanto, guarde-o em algum lugar que seja acessível no futuro.

```
1: funcao EXER8
2: M1[1..3][1..4] de inteiro
3: M2[1..4][1..5] de inteiro
4: M3[1..3][1..5] de inteiro
5: L, C, K, T, X : inteiro
6: T ← 6
7: para L=1; L<4 ;L++
8:   para C=1; C<5; C++
9:     M1[L][C] ← T
10:    T ← T + 3
11:   fim{para}
12: fim{para}
13: T ← 9
14: para L=1; L<5; L++
15:   para C=1; C<6; C++
16:     M2[L][C] ← T
17:     T ← T + 4
18:   fim{para}
19: fim{para}
20: para L=1; L<4; L++
21:   para C=1; C<6; C++
22:     X ← 0
23:     para K=1; K<5; K++
24:       X ← X + (M1[L][K] × M2[K][C])
25:     fim{para}
26:     M3[L][C] ← X
27:   fim{para}
28: fim{para}
29: imprima M3[2][4]
```

Exercício 9 Neste exercício você deve criar uma matriz 4 × 4 começando com o valor 13 e continuando na sequência natural (primeiro a linha 1, e nesta todas as colunas e depois a linha 2 e assim por diante). Após cada atribuição incrementalmente o valor usado em 6. Ao final, totalize a diagonal principal.

Exercício 10 Neste exercício você deve criar uma matriz 4 × 4 começando com o valor 11 e continuando na sequência natural (primeiro a linha 1, e nesta todas as colunas e depois a linha 2 e assim por diante). Após cada atribuição incrementalmente o valor usado em 3. Ao final, descubra o maior valor da diagonal secundária.

Para você fazer

1	2	3	4	5
6	7	8	9	10



Árvores São estruturas muito úteis e usadas em ciência da computação. O nome vem de uma certa semelhança (?) com as árvores do mundo real, só que aqui ao contrário da natureza, as árvores crescem para baixo, como se elas estivessem penduradas no céu.

Nossas árvores serão usadas para representar procedimentos ou para guardar dados, mas em ambos os casos elas terão as seguintes características comuns:

- uma estrutura hierárquica de nodos interligados
- o nodo inicial é a RAIZ
- os nodos finais são as FOLHAS
- uma ligação entre a raiz e uma folha é um CAMINHO
- em uma ligação, o nodo superior é o PAI e o inferior o FILHO
- a analogia familiar se mantém: irmão, neto, avô etc ¹
- A altura da árvore é o comprimento do maior caminho
- O nível de um nodo é o comprimento da ligação entre esse nodo e a raiz (O nível da raiz é zero).
- O grau de uma árvore é o número máximo de filhos que qualquer nodo pode ter. Uma árvore binária tem grau=2.
- O grau de um nodo é o número de filhos que ele tem.
- Na árvore não pode haver ciclos: se houver vira um grafo.

Árvores são muito comuns em nosso dia a dia, senão vejamos alguns exemplos: a) O esquema de diretórios do UNIX (e do windows); b) Uma tabela da copa do Brasil (e de qualquer torneio mata-mata); c) Uma árvore genealógica; etc.

As árvores como estruturas de armazenamento de dados serão estudadas com bastante detalhe no segundo ano em estruturas de dados, já que elas são a base de inúmeros desenvolvimentos da ciência da computação. Por exemplo, elas dão funcionalidade aos índices de praticamente todos os sistemas de gerenciamento de bancos de dados existentes, estão no âmago dos algoritmos dos principais compressores de dados, operacionalizam os métodos mais rápidos de busca de dados, só para ficar no básico.

Outra característica importante das árvores é que elas são estruturas recursivas ² por excelência e só usando este método é que sua exuberante funcionalidade se mostra por inteiro. Novamente, tem-se que esperar: o tema recursividade é estudado também no segundo ano.

O assunto aqui é a árvore como estrutura auxiliar de um algoritmo. Eis alguns dos problemas que ela ajuda a resolver:

Problema das 8 moedas Existem 8 moedas, visualmente indistinguíveis. Uma delas tem peso diferente. É mais leve ou mais pesada. Usando uma balança de 2 pratos no máximo 3 vezes, identificar qual a moeda diferente e qual a diferença.

O jogo do 8 Este jogo é jogado por dois adversários que lançam em sua vez uma jogada composta do número 1, 2 ou 3. O jogador está impedido de repetir a jogada anterior de seu adversário. As jogadas vão sendo somadas, e o jogador que atingir 8 ganha. Quem passa de 8 perde. Não há empate. Qual deve ser a estratégia geral ?

Matrículas em cursos livres Suponha uma oferta de turmas e horários de um determinado curso em que os alunos devem se inscrever. Cada aluno pode construir seu horário, sujeito à regra de que não pode haver horários encavalados. A solução pode ser obtida desenhando-se uma árvore em que cada nível é ocupado por uma disciplina. A ligação entre níveis ocorre apenas quando não há incompatibilidade. Um caminho sinaliza um conjunto compatível.

Representação de expressões Na expressão $1+2\times 3$, o que deve ser feito antes ? $1+(2\times 3)$ ou $(1+2)\times 3$?

¹Se bem que nem sempre dá muito certo. Veja-se o texto abaixo, publicado no início do século em um jornal alemão: "...eu me casei com uma viúva que tinha uma filha crescida. Meu pai que nos visitava com frequência apaixonou-se pela minha enteada e se casou com ela. Assim meu pai se tornou meu genro e minha enteada se tornou minha mãe. Alguns meses mais tarde minha esposa deu a luz à um filho que se tornou cunhado de meu pai e ao mesmo tempo meu tio. A esposa de meu pai que é minha enteada também teve um filho. Desta maneira eu ganhei um irmão e ao mesmo tempo um neto. Minha esposa tornou-se agora minha avó já que ela é mãe de minha mãe. Assim sendo eu sou o marido de minha esposa e ao mesmo tempo seu enteado-neto. Em outras palavras, tornei-me meu proprio avô."

²Uma estrutura é recursiva quando é definida em termos dela própria. Por exemplo, a definição dos números naturais, em termos dos axiomas de Peano:

- O primeiro natural é o zero
- O sucessor de um número natural é um número natural

O jogo do 31 Jogue um dado: a face que fica para cima dá o valor inicial da soma corrente. Daí por diante cada jogador faz uma jogada tomando o dado para um lado (uma das quatro faces possíveis. A face oposta não pode ser escolhida. A soma entre a face apresentada e a oposta é sempre 7). A face que aparece é então somada. O jogo prossegue até que o jogador que ao tomar o dado totalize 31 ganha. Quem ultrapassar 31 perde. Não há empate. Para se achar na árvore, na hora de desenhá-la, use [] para minha jogada e () para a jogada do adversário.

Cruzamento de fronteiras Que caminho seguir entre dois países de maneira a minimizar o cruzamento de fronteiras ? Basta colocar a origem na raiz e buscar uma folha = destino que esteja no menor nível. Ao examinar o mapa, a relação pai-filho significa "faz fronteira com".

Achar Duplicatas Uma boa e eficiente maneira de procurar itens duplicados em uma lista é construir com ela uma Árvore Binária de Pesquisa, também conhecida como ABP. Nela, usa-se uma única regra: dado um elemento qualquer na raiz, o filho esquerdo e todos os seus descendentes são MENORES do que a raiz. O filho direito e todos os seus descendentes são MAIORES que a raiz.

O jogo dos 3 peões Imagine um tabuleiro 3×3 de xadrez. Na linha 1 há 3 peões brancos e na linha 3, 3 peões pretos. O movimento é o usual entre peões, a saber:

- As brancas ou pretas podem mover-se um espaço para a frente, ocupando uma casa desocupada.
- Também, em diagonal, para capturar um peão adversário que é retirado do tabuleiro

Vence o jogo quem chegar à linha base do adversário OU bloqueando o adversário de modo que ele fica sem ter para onde se mover OU capturando todas as peças do adversário.

Qual uma boa estratégia ? Ainda, a propósito, como representar os nodos da árvore ?

Encontrar a saída de um labirinto Constrói-se a árvore colocando o ponto inicial na raiz. Daí, a cada local de decisão cria-se um nodo, cujos filhos serão todas as alternativas nesse ponto de decisão. Prossegue-se até que um nodo coincida com a saída. O caminho que liga esta folha à raiz é a solução do labirinto.

Tries Estas árvores cujo nome deriva da palavra reTRIEval (=recuperação) são a alternativa mais rápida que existe para a criação de dicionários (por exemplo aquele usado pelo Word quando se lhe manda fazer uma verificação num texto recém digitado).

Xadrez Os jogadores automáticos de xadrez tem sua estratégia de jogo construída a partir de uma árvore denominada árvore Minimax. A situação atual do tabuleiro está na raiz. Cada uma das jogadas possíveis gera um filho. Como em média, a cada lance há cerca de 30 respostas possíveis (em média), a cada ciclo do jogo (2 lances) a árvore cresce 900 vezes. Ou seja: 1 ciclo=900 nodos, 2 ciclos=900² = 810.000 nodos, 3 ciclos=900³ = 729.000.000 nodos, 4 ciclos=900⁴ = 656.100.000.000 nodos, 5 ciclos =900⁵ = 5.904 $\times 10^{14}$, 6 ciclos =900⁶ = 5.314 $\times 10^{17}$, 10 ciclos=900¹⁰ = 3.486 $\times 10^{29}$, 20 ciclos=900²⁰ = 1.215 $\times 10^{59}$, e assim por diante. O Deep Blue, que ganhou do campeão mundial Gary Kasparov em 1997, conseguia em certos casos estudar a profundidade 48 (24 ciclos) na árvore.

Deve-se lembrar que um jogo comum de xadrez tem entre 40 e 50 ciclos, logo a árvore total do jogo de xadrez poderia ter 900¹⁰⁰ = 2.656139889 $\times 10^{295}$ nodos. Apenas para comparação os físicos dizem que o número total de elétrons em todo o universo é menor do que 10⁸⁰. Este algoritmo está baseado no algoritmo A* (Hart, 1968 e 1972) e será estudado em inteligência artificial no quarto ano (Tópicos Especiais).

Árvore de decisão Veja-se também o conceito de árvore de decisão (/material/arvores/arvores_decisao.tex).

 **Para você fazer**

Fronteiras Você está no estado de SE e quer ir para o estado de RJ atravessando o menor número possível de fronteiras estaduais. Esse menor número é de _____.

Jogo 31 É sua vez de jogar e a soma apresenta 22 e o seu adversário acabou de jogar 2 . Você deve construir toda a árvore do jogo até as folhas. Depois disso, diga aqui _____ quantas folhas indicam que você ganhou o jogo.

ABP A Árvore Binária de Pesquisa construída com os elementos

83 15 29 59 78 57 76 89 54 63 86 23 49 56 55

terá altura de _____



Definição

Seja X um conjunto composto por *i* elementos entre os quais se pode estabelecer uma relação de ordem. Dados *X_i* e *X_j* com *i* ≠ *j*, sempre pode-se estabelecer *X_i* > *X_j* ou *X_i* = *X_j* ou *X_i* < *X_j*.

- X estará em ordem crescente se e somente se *X_i* ≤ *X_j*, ∀ *i* < *j*.
- X estará em ordem estritamente crescente se e somente se *X_i* < *X_j*, ∀ *i* < *j*.
- X estará em ordem decrescente se e somente se *X_i* ≥ *X_j*, ∀ *i* < *j*.
- X estará em ordem estritamente decrescente se e somente se *X_i* > *X_j*, ∀ *i* < *j*.

Um algoritmo de ORDENAÇÃO é aquele que recebe um X qualquer (possivelmente desordenado), permuta seus elementos e devolve X em ordem.

Tipicamente fazem parte de *X_i* um conjunto de informações. Neste caso, haverá uma parte de *X_i* denominada CHAVE é identificada por *k* (k=key) pela qual se fará a ordenação. Embora devamos ter em mente a existência dos outros campos, apenas a chave será tratada nos algoritmos.

Existem 2 classes de algoritmos de ordenação: os que trazem todos os dados para a memória (sort interno) e os que ordenam dados em mídia magnética sequencial (sort externo). Note que neste caso o algoritmo apenas acessa uma parte do subconjunto total.

Um algoritmo de ordenação é estável se no caso particular em que k[i] = k[j], a ordem original dos dados é preservada. É sempre bom que um método de ordenação seja estável.

Antes de prosseguir, vale uma lembrança. Para conjunto de dados pequenos (quanto?) não há necessidade de grandes pesquisas e deve-se usar o algoritmo mais simples possível.

Primeiro critério: buscar efetuar as trocas in situ. Por isso, o algoritmo a seguir deixa a desejar. (Ele movimenta os dados de um vetor para o outro)

Algoritmo de movimentação Este caso é um contra-exemplo. O algoritmo a seguir é um que apresenta uma ordenação ineficiente

```
1: algoritmo MOVIMENTA (inteiro V[1000])
2: inteiro Y[1000]
3: inteiro MAIOR, QUAL
4: inteiro I,J
5: leia X
6: para I de 1 até 1000
7:   MAIOR ← -∞
8:   para J de 1 até 1000
9:     se V[J] > MAIOR
10:      MAIOR ← X[J]
11:      QUAL ← J
12:   fim{se}
13:   fim{para}
14:   Y[I] ← X[QUAL]
15:   V[QUAL] ← -∞
16: fim{para}
17: devolva Y
18: finalgoritmo
```

Os algoritmos de ordenação, grosso modo, podem ser divididos em 2 categorias: aqueles de complexidade O(n²) e os de complexidade O(n × log₂n), ambas as expressões – como sempre – a menos das constantes multiplicativas.

O estudo da primeira família se justifica pelas seguintes razões:

- Constantes multiplicativas eventualmente menores
- Abundância de recursos de hardware
- Irrelevância do desempenho para pequenas instâncias

- Simplicidade (inclusive conceitual) dos algoritmos envolvidos

Este exercício tratará de 3 algoritmos, além de uma pequena derivação de um deles, todos com desempenho quadrático. São eles:

Bolha Trata-se do algoritmo mais simples que existe. Pares de vizinhos são comparados e se eles estiverem desordenados, são invertidos. A cada passada, o elemento mais leve é levado para a ponta. Daí o nome.

Bolha* O mesmo algoritmo acima, mas com um pequeno truque para encerrar o algoritmo mais cedo caso o vetor a ordenar seja do tipo “quasi-ordenado”.

Inserção O mesmo algoritmo usado pelo jogador de buraco ao ordenar as cartas que recebe. Há um monte de cartas desordenadas. As cartas, uma a uma, vão sendo retiradas desse monte e colocadas em ordem na mão do freguês. Quando o monte terminar, a mão está ordenada.

Seleção O menor elemento do vetor a ordenar é intercambiado com o primeiro elemento do vetor. Depois, o segundo menor com o segundo elemento, e assim por diante.

Algoritmo Bolha

```
1: inteiro I,J,AUX
2: I ← tamanho do vetor a ordenar
3: enquanto (I ≥ 2)
4:   J ← 2
5:   enquanto (J ≤ I)
6:     se V[J - 1] > V[J]
7:       AUX ← V[J - 1]
8:       V[J - 1] ← V[J]
9:       V[J] ← AUX
10:    fim{se}
11:    J ++
12:  fim{enquanto}
13:  I --
14: fim{enquanto}
```

Note as duas características que este algoritmo tem: uma ineficiência grande aliada à simplicidade do código. A grande maioria dos programadores profissionais, conhece “de cabeça”este algoritmo.

Algoritmo Bolha-Estrela

```
1: inteiro I,J,AUX
2: lógico MUDOU ← .V.
3: I ← tamanho do vetor a ordenar
4: enquanto (I ≥ 2) ∧ MUDOU
5:   J ← 2
6:   MUDOU ← .F.
7:   enquanto (J ≤ I)
8:     se V[J - 1] > V[J]
9:       AUX ← V[J - 1]
10:      V[J - 1] ← V[J]
11:      V[J] ← AUX
12:      MUDOU ← .V.
13:   fim{se}
14:   J ++
15: fim{enquanto}
16: I --
17: fim{enquanto}
```

A única diferença em relação ao original é a presença da variável MUDOU que determina o fim do loop externo, quando nenhuma mudança mais for feita.

Algoritmo de Inserção

```
1: inteiro I, J, X
2: I ← 2
3: TAM ← Tamanho do vetor a ordenar
4: enquanto (I ≤ TAM)
5:   X ← V[I]
6:   J ← I - 1
7:   enquanto (J ≥ 1) ∧ (V[J] > X)
8:     V[J + 1] ← V[J]
9:     J --
10:  fim{enquanto}
11:  V[J + 1] ← X
12:  I ++
13: fim{enquanto}
```

Na linha 5, o elemento que vai ser colocado em seu lugar definitivo é sacado do vetor. Com isso ele abre um espaço para remanejamento dentro do vetor. Isso é feito enquanto V[J] > X nas linhas 8

e 9. Se o sujeito que foi sacado é o menor do vetor, a condição do enquanto é encerrada pela outra condição (J ≥ 1). Depois que o remanejamento foi feito, o elemento sacado volta para o vetor, agora no seu lugar certo (linha 11).

Algoritmo Seleção

```
1: inteiro I,J,COR,INUI,AUX
2: TAM ← Tamanho do vetor a ordenar
3: I ← 1
4: enquanto (I ≤ TAM)
5:   COR ← V[I]
6:   INUI ← I
7:   J ← I + 1
8:   enquanto (J ≤ TAM)
9:     se V[J] < COR
10:      COR ← V[J]
11:      INUI ← J
12:   fim{se}
13:   J ++
14: fim{enquanto}
15: AUX ← V[I]
16: V[I] ← V[INUI]
17: V[INUI] ← AUX
18: I ++
19: fim{enquanto}
```

O menor elemento é procurado (linhas 8 a 14). Encontrado (agora ele é apontando por INUI, ele é trocado de lugar com o elemento apontado por I (linhas 15 a 17).

Desempenho

Compare os desempenhos dos 4 algoritmos estudados:

Tam.	Bol	Bol*	Ins	Sel
100 aleat.	0.6	0.7	0.3	0.4
100 q.-o.	0.4	0.06	0.01	0.4
500 aleat.	16	17	7	10
500 q.-o.	11	0.1	0.1	10
1000 aleat.	68	70	31	41
1000 q.-o.	47	0.3	0.2	40
2500 aleat.	429	436	198	255
2500 q.-o.	294	0.6	0.5	255

Obs1: q-o. significa quasi-ordenado, ou seja apenas um elemento está fora de ordem no conjunto.

Obs2: todos os tempos acima estão em segundos.

Obs3: medidas feitas em um Pentium 500Mhz com Windows ME, usando APL*PLUS, versão 6.0, interpretada.

Para você fazer

Estão disponíveis na intranet para esta aula, 6 arquivos denominados ORDE01.ord até ORDE06.ord Cada um deles contém 5000 registros de 40 bytes cada. Desses 40 bytes, os primeiros 4 correspondem a um número binário e deverão ser interpretados como a chave dos registros. Com base nos algoritmos acima, crie programas em C capazes de implementar os 4 algoritmos estudados (bolha, bolha*, seleção e inserção). Não esqueça de imprimir no início e no final do seu programa a hora do processamento ou então use qualquer ferramenta de aferição de tempo.

Arquivo	algoritmo	tempo (seg)
ORDE03	selecao	1.
ORDE05	insercao	2.
ORDE01	bolh*	3.

IMPORTANTE: junto com esta folha entregue as listagens dos 3 programas produzidos por você.



O cubo RUBIK

Criado por Erno Rubik em 1974, tem um mecanismo simples que surpreende tanto do ponto de vista mecânico – ao estudar seu interior, como pela complexidade das combinações que se conseguem ao girar suas faces.

O invento, descendente de um protótipo que tinha apenas duas faces é um tipo de quebracabeças que consiste de um cubo em que cada uma das 6 faces está dividida em 9 peças. Elas se articulam graças ao mecanismo da peça interior central que fica oculta dentro do cubo. O resto das peças são visíveis e pode-se observar 3 tipos de peças que não perdem sua condição, a despeito de qualquer movimento que o cubo sofra. São elas:

- Central As 6 peças centrais definem a cor da face. Mantém sempre a orientação relativa entre elas. No modelo original o branco se opunha ao amarelo, o vermelho ao laranja e o verde ao azul.
- Aresta Peças formadas por 2 cores, em número de 12.
- Vértices Peças dos cantos do cubo, em número de 8 e formadas por 3 cores.

Representação

Ara representar em 2D um cubo (que é 3D), vai-se fazer o seguinte mapeamento:

1. Cada instância de cubro vai ser mostrado como
- | | |
|-----------------|--------------------|
| 123 | SSS |
| 456 | SSS |
| 789 | SSS |
| abc ABC jkl JKL | EEE FFF DDD PPP |
| def DEF mno MNO | ou EEE FFF DDD PPP |
| ghi GHI pqr PQR | EEE FFF DDD PPP |
| stu | III |
| vxy | III |
| z() | III |

- onde a face marcada com 1 corresponde à aresta superior do cubo. A 2 é a face esquerda, a 3 é a frontal, a 4 é a direita e a 5 é a face posterior. Finalmente, a face 6 é a face inferior do cubo.
2. Note que as 6 peças centrais das 6 faces são fixas (5,e,E,n,N,x) e não mudam de orientação entre si (umas em relação à outras).
3. Finalmente, para economia de papel, a configuração acima vai ser apresentada como 123456789abcdefghijklmnopqrJ KLMNOPQRstuvwxyz()

Operadores

Na tentativa de escrever um programa de computador que resolva o cubo de Rubik, usa-se em geral o algoritmo A* (Hart, 68 e 72). Uma parte importante do algoritmo é a lista de operadores que se aplicam a um estado qualquer.

Para os operadores que virão a seguir, considere o estado inicial como

123
456
789
abc ABC jkl JKL
def DEF mno MNO
ghi GHI pqr PQR
stu
vxy
z()

Podem-se definir os seguintes operadores:

nome	como ficou
1=girar cubo no eixo vertical, anti-horário	369 258 147 JKL abc ABC jkl MNO def DEF mno PQR ghi GHI pqr zvs (xt)yu
2=girar cubo no eixo frontal, anti-horário	lor knq jmp 369 CFI uy) PMJ 258 BEH tx(QNK 147 ADG svz ROL cfi beh adg
3=girar cubo no eixo lateral, anti-horário (observador à esquerda)	ABC DEF GHI cfi stu pmj 987 beh vxy qnk 654 adg z() rol 321 RQP ONM LKJ
4=girar face superior, anti-horário	369 258 147 JKL abc ABC jkl def DEF mno MNO ghi GHI pqr PQR stu vxy z()
5=girar face frontal, anti-horário	123 456 jmp ab9 CFI uk1 JKL de8 BEH tno MNO gh7 ADG sqr PQR cfi vxy z()
6=girar face lateral direita, anti-horário (observador à esquerda)	12C 45F 78I abc ABu pmj 9KL def DEy qnk 6NO ghi GH) rol 3QR stP vxM z(J
7=girar face superior, horário	741 852 963 ABC jkl JKL abc def DEF mno MNO ghi GHI pqr PQR stu vxy z()

nome	como ficou
8=girar face frontal, horário	123 456 ifc abs GDA 7kl JKL det HEB 8no MNO ghu IFC 9qr PQR pmj vxy z()
9=girar face lateral direita, horário (observador à esquerda)	12P 45M 78J abc AB3 lor)KL def DE6 knq yNO ghi GH9 jmp uQR stC vxF z(I

Observação importante: Note que quando ocorrem giros na face direita, a posição do operador SEM-PRE é a esquerda do cubo. Se tiver dúvidas sobre este operador, olhe as tabelas acima nos casos 3, 6 e 9.

👉 Para você fazer

1. Aplique o operador **8** ao cubo

gd1
he2
JMP
zv3 lor)KL aOR
(xf DEq y58 BNQ
96i GHp u47 AFC
stI
knb
jmc

e obterá como resposta um cubo que terá, na face **frontal** na posição L= **3** e C= **3** o valor

2. Aplique o operador **1** ao cubo

zvL
(x0
AHP
RQ7 ctr)da 1hg
kno MEq yeb 4N6
jmI pD3 J8i GK9
ufl
F5B
C2s

e obterá como resposta um cubo que terá, na face **posterior** na posição L= **2** e C= **1** o

valor

3. Quais (2) operadores devem ser usados no cubo

741
85F
96I
ABC jku pmJ 3bc
def DEy qnK 2NO
ghi GH) roL 1QR
stP
vxM
z(a

a fim de resolvê-lo ?



Obs: vide o programa RUBIK135.C

Programação I

Usar JS Como este exercício está sendo apresentado na disciplina de **Programação em front end** os exercícios abaixo deverão ser construídos dentro de uma página HTML e assim apresentados. Fica a seu critério o uso (ou não) de CSS, bem como a distribuição de entradas e saídas dentro da página HTML. De qualquer maneira verifique que os textos em azul deverão ser pedidos no HTML e as respostas em vermelho deverão ser também mostradas no HTML depois de devidamente calculadas por uma rotina em Javascript.

 Para você fazer

Atente que nos exemplos abaixo, tudo o que é impresso na cor azul é escrito pelo programa e na cor vermelho é digitado pelo usuário. Veja também que em geral, as mensagens não são acentuadas, para prevenir dificuldades a depender do ambiente empregado.

1. Escreva um programa que leia n (indeterminado, condição de fim = 0) e imprima a amplitude dos valores fornecidos (amplitude = maior - menor)

```
Exemplos de execução
Informe sequencia de positivos. (0 para fim)
117 56 13 17 18 98 0
Amplitude: 104

Informe sequencia de positivos. (0 para fim)
1 2 3 0
Amplitude: 2
```

2. Uma salada de frutas industrial é produzida com Laranja, Mamão, Abacaxi e Banana, na seguinte proporção: 1 kg de laranja : 1.5 kg de mamão : 0.5 kg de abacaxi : 2 Kg de banana, para produzir 5 kg de salada de fruta. Escreva um programa que receba as quantidades disponíveis de L , M , A e B e imprima a quantidade de salada que pode ser produzida. Todas as quantidades estão dadas em kilogramas e são números reais.

```
Exemplos de execução:
Informe laranja, mamao, abacaxi e banana:
1.0 1.0 3.0 1.0
Salada: 2.5

Informe laranja, mamao, abacaxi e banana:
1 2 3 4
Salada: 5
```

3. Escreva um programa que leia 3 números de no máximo 2 dígitos cada e verifique se a soma deles é maior do que 100. Se o valor da soma for maior do que 100 o programa deve imprimir a mensagem "limite indisponível". Caso contrário, a mensagem "limite permitido" deve ser exibido. O programa não pode aceitar valores com mais de 2 dígitos.

```
Exemplo de execução:
Indique 3 numeros inteiros: 2 20 12
limite permitido

Outro exemplo:
Indique 3 numeros inteiros: 2 90 12
limite indisponivel

Mais um:
Indique 3 numeros inteiros: 2 190 12
valores invalidos
```

4. Uma empresa decidiu dar uma gratificação de natal a seus funcionários. Esta gratificação é calculada com base no número de horas extras trabalhadas e o número de faltas ao trabalho. A fórmula para calcular as horas premiadas é $HorasPremiadas = HorasExtras - \frac{2}{3} \times HorasFaltas$. O prêmio é distribuído segundo a tabela

Horas Premiadas	Prêmio
até 10 horas (inclusive)	10,00
de 11 até 20 horas (inclusive)	20,00
de 21 até 30 horas (inclusive)	30,00
de 31 até 40 horas (inclusive)	40,00
acima de 40 horas	50,00

Escreva um programa que pergunte ao usuário o número de horas extras trabalhadas e o número de horas em que faltou ao trabalho e mostre na tela o prêmio devido em reais.

```
Exemplo de execução:
Horas extras: 30
Horas de faltas: 3
Prêmio: R$ 30.00
```

5. Escreva um programa que leia dois números: o primeiro é o peso de uma pessoa (real, em quilogramas) e o segundo a altura dela em metros, real também. O programa deve calcular e imprimir o IMC - Índice de massa corporal que é calculado pela fórmula

$$IMC = \frac{\text{peso em kg}}{\text{quadrado da altura em metros}}$$

```
Informe o peso em quilogramas = 80
altura em metros = 1.80
IMC: 24.6914

Informe o peso em quilogramas = 93.5
altura em metros = 1.63
IMC: 35.1914
```

Espaço para avaliação do trabalho:

1	2	3	4	5

Listagem de todos os problemas de 136b

1. Escreva um programa que leia 3 números de no máximo 2 dígitos cada e verifique se a soma deles é maior do que 100. Se o valor da soma for maior do que 100 o programa deve imprimir a mensagem "limite indisponível". Caso contrário, a mensagem "limite permitido" deve ser exibido. O programa não pode aceitar valores com mais de 2 dígitos.

Exemplo de execução:

```
Indique 3 numeros inteiros:  2 20 12
limite permitido
```

Outro exemplo:

```
Indique 3 numeros inteiros:  2 90 12
limite indisponivel
```

Mais um:

```
Indique 3 numeros inteiros: 2 190 12
valores invalidos
```

```
#include<iostream>
using namespace std;
int main() {
    int a,b,c;
    cout << "Indique 3 numeros inteiros: "<<endl;
    cin>>a>>b>>c;
    if ((a>99) || (b>99) || (c>99)) {
        cout<<"Valores invalidos"<<endl;
    }
    else {
        if ((a+b+c)>100){
            cout<<"Limite indisponivel"<<endl;
        }
        else {
            cout<<"Limite permitido"<<endl;
        }
    }
}
```

2. Uma empresa decidiu dar uma gratificação de natal a seus funcionários. Esta gratificação é calculada com base no número de horas extras trabalhadas e o número de faltas ao trabalho. A fórmula para calcular as horas premiadas é $HorasPremiadas = HorasExtras - \frac{2}{3} \times HorasFaltas$. O prêmio é distribuído segundo a tabela

Horas Premiadas	Prêmio
até 10 horas (inclusive)	10,00
de 11 até 20 horas (inclusive)	20,00
de 21 até 30 horas (inclusive)	30,00
de 31 até 40 horas (inclusive)	40,00
acima de 40 horas	50,00

Escreva um programa que pergunte ao usuário o número de horas extras trabalhadas e o número de horas em que faltou ao trabalho e mostre na tela o prêmio devido em reais.

Exemplo de execução:

```
Horas extras:  30
Horas de faltas:  3
Prêmio: R$ 30.00
```

```
#include<iostream>
using namespace std;
int main() {
    int ht, hf;
    float hp;
    cout << "Horas extras: "<<endl;
    cin>>ht;
    cout << "Horas de faltas: "<<endl;
    cin>>hf;
    hp = ht-((2/3.0)*hf);
    if (hp<=10){ cout<<"Premio: R$ 10.00"<<endl; }
    else {
        if (hp<=20){ cout<<"Premio: R$ 20.00"<<endl; }
        else {
            if (hp<=30){ cout<<"Premio: R$ 30.00"<<endl; }
            else {
                if (hp<=40){ cout<<"Premio: R$ 40.00"<<endl; }
                else {
                    cout<<"Premio: R$ 50,00"<<endl;
                }
            }
        }
    }
}
```

3. Escreva um programa que receba três numeros e mostre-os em ordem decrescente. O programa só deve aceitar 3 números diferentes.

Exemplo de execução:

```
Indique 3 numeros:  12 2 90
90 12 2
```

Mais uma:

```
Indique 3 numeros:  12 190 12
Valores invalidos
```

```

Outra:
Indique 3 numeros:  12 12 12
Valores invalidos

```

```

#include<iostream>
using namespace std;
int main() {
    int a,b,c;
    cout<<"Indique 3 numeros: "<<endl;
    cin>>a>>b>>c;
    if ((a==b)|| (a==c)|| (b==c)){
        cout<<"Valores invalidos"<<endl;
    }
    else {
        if ((a<b) && (a<c) && (b<c)) {cout<<c<<" "<<b<<" "<<a;}
        if ((a<b) && (a<c) && (c<b)) {cout<<b<<" "<<c<<" "<<a;}
        if ((b<a) && (b<c) && (a<c)) {cout<<c<<" "<<a<<" "<<b;}
        if ((b<a) && (b<c) && (a>c)) {cout<<a<<" "<<c<<" "<<b;}
        if ((c<b) && (c<a) && (a<b)) {cout<<b<<" "<<a<<" "<<c;}
        if ((c<b) && (c<a) && (a>b)) {cout<<a<<" "<<b<<" "<<c;}
    }
}

```

4. Escreva um programa que leia 3 números reais, A , B e C , calcule e imprima as raízes reais da equação de segundo grau $Ax^2 + Bx + C = 0$. Se não houverem raízes reais, imprimir a mensagem RAIZES IMAGINARIAS.

```

Informe A, B e C:
1.2 3.4 5.6
RAIZES IMAGINARIAS

```

```

Informe A, B e C:
1 5 3
x1=-3.19722
x2=-6.80278

```

```

#include<iostream>
#include<cmath>
using namespace std;
int main() {
    float x1,x2,delta;
    float a,b,c;
    cout<<"Informe A, B e C: "<<endl;
    cin>>a>>b>>c;
    delta = b*b - 4*a*c;
    if (delta>=0) {
        x1 = (-b)+sqrt(delta)/2*a;
        x2 = (-b)-sqrt(delta)/2*a;
        cout<<"x1="<<x1<<endl;
        cout<<"x2="<<x2<<endl;
    }
    else {
        cout<<"RAIZES IMAGINARIAS"<<endl;
    }
    return 0;
}

```

5. Escreva um programa que leia dois números: o primeiro é o peso de uma pessoa (real, em quilogramas) e o segundo a altura dela em metros, real também. O programa deve calcular e imprimir o IMC - Índice de massa corporal que é calculado pela fórmula

$$IMC = \frac{\text{peso em kg}}{\text{quadrado da altura em metros}}$$

```

Informe o peso em quilogramas = 80
altura em metros = 1.80
IMC: 24.6914

```

```

Informe o peso em quilogramas = 93.5
altura em metros = 1.63
IMC: 35.1914

```

```

#include<iostream>
using namespace std;
int main(){
    float peso;
    float altura;
    float imc;
    cout<<"Informe o peso em quilogramas = ";
    cin>>peso;
    cout<<"altura em metros = ";
    cin>>altura;
    imc=peso/(altura*altura);
    cout<<"IMC: "<<imc;
    return 0;
}

```

6. Suponha que a receita federal mede os rendimentos em litros/uisque 12 anos. Neste exemplo, os primeiros 10 litros do rendimento são isentos. Os seguintes 10 litros pagam 5% de I.R. Os seguintes 10 litros pagam 10% de i.r. e finalmente, qualquer rendimento que ultrapasse 30 litros pagará 25% de imposto. Escreva um programa que receba um certo rendimento medido em litros e imprima o imposto devido também medido em litros (números reais).

Alguns exemplos:
Informe o rendimento em litros = 66
10 10 10 36 ir=10.5

Informe o rendimento em litros = 100
10 10 10 70 ir=19
(os números antes do IR são os litros de cada alíquota)

```
#include<iostream>
using namespace std;
int main(){
double gent,g1,g2,g3,g4,ir;
cout<<"Informe o rendimento em litros ";
cin>>gent;
g1=min(10.0,gent);
g2=max(0.0,min(10.0,gent-10));
g3=max(0.0,min(10.0,gent-20));
g4=max(0.0,gent-30);
ir=(0.05*g2)+(0.1*g3)+(0.25*g4);
cout<<g1<<" "<<g2<<" "<<g3<<" "<<g4<<" ir="<<ir<<endl;
}
```

7. Uma salada de frutas industrial é produzida com Laranja, Mamão, Abacaxi e Banana, na seguinte proporção: 1 kg de laranja : 1.5 kg de mamão : 0.5 kg de abacaxi : 2 Kg de banana, para produzir 5 kg de salada de fruta. Escreva um programa que receba as quantidades disponíveis de L , M , A e B e imprima a quantidade de salada que pode ser produzida. Todas as quantidades estão dadas em kilogramas e são números reais.

Exemplos de execução:
Informe laranja, mamao, abacaxi e banana:
1.0 1.0 3.0 1.0
Salada: 2.5

Informe laranja, mamao, abacaxi e banana:
1 2 3 4
Salada: 5

```
#include<iostream>
using namespace std;
int main(){
float L,M,A,B;
float tl,tm,ta,tb;
cout<<"Informe laranja, mamao, abacaxi e banana: ";
cin>>L>>M>>A>>B;
tl=L/1.0;
tm=M/1.5;
ta=A/0.5;
tb=B/2.0;
cout<<"Salada: "<<min(min(min(tl,tm),ta),tb)*5.0;
}
```

8. Está previsto para o próximo dia 14 a ocorrência de um eclipse lunar. Em nossa região, ele deve começar em algum momento entre 21h00 e 23h45 do dia 14 e terminar entre 03h05 e 05h41 do dia 15. A variação de duração e de início e final depende do lugar onde você estará. Escreva um programa que receba a hora (e minuto) do início e também a hora (e minuto) do final. Seu programa deve imprimir a duração em minutos do eclipse nesse local.

Exemplos de execução
Informe hh mm do inicio e hh mm do final
23 01 04 59
Duracao 358

Informe hh mm do inicio e hh mm do final
21 10 3 6
Duracao 356

```
#include<iostream>
using namespace std;
int main(){
int hi,mi,hf,mf;
int p1,p2;
cout<<"Informe hh mm do inicio e hh mm do final";
cin>>hi>>mi>>hf>>mf;
p1=(hi*60)+mi;
p2=(24*60)+(hf*60)+mf;
cout<<"Duracao "<<p2-p1<<endl;
}
```

9. Uma calça jeans masculina de um certo tamanho usa $1.7m^2$ de brim e pode ser vendida por M reais. Uma calça similar, mas feminina usa $1.58m^2$ e pode ser vendida por F reais. Dada uma certa peça (retalho) de jeans ela poderá ser usada para fazer qualquer número de um ou outro tipo de calça, mas não ambas no mesmo retalho. As sobras após o corte não têm valor. Suponha ainda que o mercado absorve tudo o que for produzido. Escreva um programa que receba a área de um retalho de jeans, (medido em m^2) os valores M e F medidos em reais e imprima quanto será faturado na produção e venda de calças masculinas e quanto será faturado na produção e vendas de calças femininas.

Exemplos de execução:
Informe area retalho, preco M e preco F:
10.5 129 136
Masculinas = 796.765
Femininas = 903.797

Informe area retalho, preco M e preco F:
100 10 10
Masculinas = 588.235
Femininas = 632.911

```
#include<iostream>
using namespace std;
int main(){
float ret, fm, ff, pem, pef;
cout<<"Informe area retalho, preco M e preco F: ";
cin>>ret>>fm>>ff;
pem=ret/1.7;
pef=ret/1.58;
cout<<"Masculinas = "<<fm*pem<<endl;
cout<<"Femininas = "<<ff*pef;
}
```

10. Escreva um programa que receba um inteiro positivo N e informe "PRIMO" se ele for primo e "COMPOSTO" se ele não for. Definição: um número inteiro positivo é primo se ele for maior que 1 e tiver apenas dois divisores: a unidade e o próprio N . Desta definição, percebe-se que há necessidade de contar a quantidade de divisores do número dado e depois comparar esta quantidade com 2

Exemplos de execução

```
Informe N: 121
composto
```

```
Informe N: 991
primo
```

Dicas de eficiência, já que diversas melhorias podem ser feitas sobre este código:

- deixar de fora o 1 e o N e comparar os divisores com > 0 .
- excluir os pares (fora o 2), e acelerar de 2 em 2.
- ir só até $\sqrt{N}$, já que só é necessário achar 1 divisor (fora o 1 e o N) para concluir que o número não é primo.
- ao achar um divisor, dá para cair fora.
- dá para acelerar de 6 em 6, mas aí é preciso estudar um pouco os primos, já que todo primo tem a forma $P = 6k \pm 1$.

```
#include<iostream>
using namespace std;
int main(){
int N;
int divisores, x;
divisores=0;
x=1;
cout<<"Informe N: ";
cin>>N;
while (x<=N){
if ((N%x)==0){
divisores++;
}
x++;
}
if (divisores>2) {
cout<<"composto"<<endl;
}
else {
cout<<"primo"<<endl;
}
}
```

11. Escreva um programa que leia n (indeterminado, condição de fim = 0) e imprima a amplitude dos valores fornecidos (amplitude = maior - menor)

Exemplos de execução

```
Informe sequencia de positivos. (0 para fim)
117 56 13 17 18 98 0
Amplitude: 104
```

```
Informe sequencia de positivos. (0 para fim)
1 2 3 0
Amplitude: 2
```

```
#include<iostream>
using namespace std;
int main() {
int x, mai, men;
mai=-9999999;
men=9999999;
cout<<"Informe sequencia de positivos. (0 para fim)"<<endl;
cin>>x;
while(x>0){
if (x>mai) {mai=x;}
if (x<men) {men=x;}
cin>>x;
}
cout<<"Amplitude: "<<mai-men<<endl;
}
```

12. Escreva um programa que imprima uma tabela dos primeiros 10 números ímpares (começando no 1 terminando no 19) com seus quadrados e cubos.

1	1	1
3	9	27
5	25	125
7	49	343
9	81	729

11	121	1331
13	169	2197
15	225	3375
17	289	4913
19	361	6859

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
int main(){
    int i;
    for (i=1;i<=20;i=i+2){
        cout<<setw(4);
        cout<<i;
        cout<<setw(7);
        cout<<pow(i,2);
        cout<<setw(10);
        cout<<pow(i,3);
        cout<<endl;
    }
}
```

Programação II

 Para você fazer

Atente que nos exemplos abaixo, tudo o que é impresso na cor azul é escrito pelo programa e na cor vermelho é digitado pelo usuário. Veja também que em geral, as mensagens não são acentuadas, para prevenir dificuldades a depender do ambiente empregado.

1. Escreva um programa que leia a partir do teclado, a data de nascimento (dia, mês e ano) de uma pessoa e determine quantos anos completos ela terá no dia 08 de julho de 2024.

```
Data nascimento: (d m a)
7 7 2000
Idade: 24

Data nascimento: (d m a)
9 7 2000
Idade: 23
```

2. Faça um programa que leia do teclado três números reais, calcule a sua média aritmética e identifique qual deles é o mais distante da média calculada. O programa deverá considerar inválida a entrada de 3 números iguais.

```
Exemplos de execução
Indique 3 numeros:
2 34 12.5
Media: 16.1667
34 esta mais distante da media

Indique 3 numeros:
1 3 1
Media: 1.66667
3 esta mais distante da media

Indique 3 numeros:
1 1 1
Dados invalidos
```

3. Escreva um programa que leia uma sequência de números positivos inteiros terminados por 0 e imprima o maior valor lido.

```
Exemplos de execução
Informe numeros (0 para fim):
2 45 23 9 89 11 2 0
89
```

4. Dados 3 inteiros positivos, verificar se eles formam os lados de um triângulo retângulo

```
Informe 3 lados:
3 4 5
SIM

Informe 3 lados:
4 5 3
SIM

Informe 3 lados:
1 2 3
```

5. Escreva um programa que leia dois números N_1 e N_2 e some os inteiros entre eles (extremos incluídos) que sejam múltiplos de 7 imprimindo o resultado ao final.

```
Exemplos de execução:
Informe N1 e N2: 11 20
Soma dos multiplos de 7: 14

Informe N1 e N2: 10 30
Soma dos multiplos de 7: 63
```

Espaço para avaliação do trabalho:

1	2	3	4	5

Listagem de todos os problemas de 136c

1. Dados n , i e j imprimir em ordem crescente os n primeiros naturais múltiplos de i , de j ou de ambos.

Exemplo de execução:

```
Informe n, i e j: 10 3 8
3 6 8 9 12 15 16 18 21 24
```

Outro exemplo:

```
Informe n, i e j: 8 1 2
1 2 3 4 5 6 7 8
```

```
#include<iostream>
using namespace std;
int main(){
    int i,j,n;
    int qtd=0,cand=1;
    cout<<"Informe n, i e j: ";
    cin>>n>>i>>j;
    while (qtd<n){
        if (((cand%i)==0) || ((cand%j)==0)){
            cout<<cand<<" ";
            qtd=qtd+1;
        }
        cand++;
    }
}
```

2. Dado um inteiro não negativo n determinar $n!$ (fatorial de n).

Exemplo de execução:

```
Informe N para o !N: 5
120
```

```
Informe N para o !N: 0
1
```

```
#include<iostream>
using namespace std;
int main(){
    long int fa=1;
    int n;
    cout<<"Informe N para o !N: ";
    cin>>n;
    while (n>1){
        fa=fa*n;
        n--;
    }
    cout<<fa;
}
```

3. Diz-se que um número i é congruente a j (módulo m) se e somente se $i\%m = j\%m$, por exemplo 35 é congruente a 39 módulo 4 pois $35\%4 = 39\%4 = 3$. Dados n , j e m imprimir os n primeiros naturais congruentes a j módulo m .

Exemplo de execução:

```
Informe n, j e m: 10 7 4
3 7 11 15 19 23 27 31 35 39
```

Mais uma:

```
Informe n, j e m: 12 5 2
1 3 5 7 9 11 13 15 17 19 21 23
```

```
#include<iostream>
using namespace std;
int main(){
    int n,j,m;
    cout<<"Informe n, j e m: ";
    cin>>n>>j>>m;
    int i=1,cuca;
    cuca=j%m;
    while (n>0){
        if ((i%m)==cuca){
            cout<<i<<" ";
            n--;
        }
        i++;
    }
}
```

4. Dados 3 inteiros positivos, verificar se eles formam os lados de um triângulo retângulo

```
Informe 3 lados:
3 4 5
SIM
```

```
Informe 3 lados:
4 5 3
SIM
```

```
Informe 3 lados:
1 2 3
```

```
#include<iostream>
using namespace std;
int main(){
    int a,b,c;
    cout<<"Informe 3 lados: ";
    cin>>a>>b>>c;
    if (((a*a)+(b*b)==(c*c)) || ((a*a)+(c*c)==(b*b)) ||
        ((b*b)+(c*c)==(a*a))) {cout<<"SIM";}
}
```

5. (FIS 88) Qualquer número natural de 4 dígitos pode ser dividido em duas dezenas formadas pelos seus primeiros dois dígitos e pelos últimos dois dígitos. Por exemplo

- 1297: 12 e 97
- 5314: 53 e 14

Escreva um programa que imprima todos os milhares (4 algarismos) cuja raiz quadrada seja a soma das dezenas formadas acima. Por exemplo, 9801 tem como raiz 99 que é a soma de 98 com 01. Logo 9801 é um dos números a serem impressos.

```
2025
3025
9801
```

```
#include<iostream>
#include<cmath>
using namespace std;
int main() {
    int ini=1000;
    int p1,p2;
    while (ini<10000){
        p1=ini/100;
        p2=ini%100;
        if(sqrt(ini)==p1+p2){
            cout<<ini<<endl;
        }
        ini++;
    }
    // sao impressos: 2025, 3025 e 9801
}
```

6. Escreva um programa que leia uma série indeterminada de números (condição de fim=0) e imprima a mensagem "em ordem crescente" se eles estiverem. Uma série está em ordem crescente se $s_i \leq s_j \quad \forall \quad i < j$.

```
Alguns exemplos:
Informe sequencia: (0 para fim)
1 3 8 90 12 300 301 0

Informe sequencia: (0 para fim)
1 2 3 66 0
em ordem crescente
```

```
#include<iostream>
using namespace std;
int main() {
    int x,esta,ant;
    ant=-99999;
    esta=1;
    cout<<"Informe sequencia: (0 para fim)"<<endl;
    cin>>x;
    while(x>0){
        if (x<ant) {esta=0;}
        ant=x;
        cin>>x;
    }
    if (esta==1){cout<<"em ordem crescente";}
}
```

7. Escreva um programa que leia a_1 , r , n e imprima a PA e a PG associados.

```
Exemplos de execução:
Informe a1, r e n:
3 4 5
3 7 11 15 19
3 12 48 192 768

Informe a1, r e n:
0 5 6
0 5 10 15 20 25
0 0 0 0 0
```

```
#include<iostream>
using namespace std;
int main(){
    long int a0,a1,r,n,x;
    cout<<"Informe a1, r e n: ";
    cin>>a0>>r>>n;
    x=1;
    a1=a0;
    while(x<=n){
```



```

cout<<a1<<" ";
a1=a1+r;
x++;
}
cout<<endl;
x=1;
a1=a0;
while(x<=n){
cout<<a1<<" ";
a1=a1*r;
x++;
}
}

```

8. Dados m e n inteiros e positivos, eles são relativamente primos se não tiverem fator comum (por exemplo, 14 e 25 são relativamente primos). Escreva um programa que leia m e n e imprima "RELATIVAMENTE PRIMOS" se eles o forem e "NAO SAO" se não forem.

Exemplos de execução

Informe m e n:

14 25

RELATIVAMENTE PRIMOS

Informe m e n:

33 11

NAO SAO

```

#include<iostream>
using namespace std;
int main() {
char R='N';
int d=2;
int m,n;
cout<<"Informe m e n: ";
cin>>m>>n;
while (d<=m){
if ((m%d==0) && (n%d==0)){
R='S';
}
d++;
}
if (R=='N'){
cout<<"RELATIVAMENTE PRIMOS"<<endl;
}
else {
cout<<"NAO SAO"<<endl;
}
}

```

9. Escreva um programa que leia dois números N_1 e N_2 e some os inteiros entre eles (extremos incluídos) que sejam múltiplos de 7 imprimindo o resultado ao final.

Exemplos de execução:

Informe N1 e N2: 11 20

Soma dos multiplos de 7: 14

Informe N1 e N2: 10 30

Soma dos multiplos de 7: 63

```

#include<iostream>
using namespace std;
int main(){
int n1,n2,soma;
soma=0;
cout<<"Informe N1 e N2: ";
cin>>n1>>n2;
while (n1<=n2){
if ((n1%7)==0){
soma=soma+n1;
}
n1=n1+1;
}
cout<<"Soma dos multiplos de 7: "<<soma;
}

```

10. Escreva um programa que leia uma sequência de números positivos inteiros terminados por 0 e imprima o maior valor lido.

Exemplos de execução

Informe numeros (0 para fim):

2 45 23 9 89 11 2 0

89

```

#include<iostream>
using namespace std;
int main() {
int n,mai;
mai=-999999;
cout<<"Informe numeros (0 para fim)";
cin>>n;
while(n!=0){
if (n>mai){

```

```

        mai=n;
    }
    cin>>n;
}
cout<<mai;
}

```

11. Faça um programa que leia do teclado três números reais, calcule a sua média aritmética e identifique qual deles é o mais distante da média calculada. O programa deverá considerar inválida a entrada de 3 números iguais.

Exemplos de execução
Indique 3 numeros:
2 34 12.5
Media: 16.1667
34 esta mais distante da media

Indique 3 numeros:
1 3 1
Media: 1.66667
3 esta mais distante da media

Indique 3 numeros:
1 1 1
Dados invalidos

```

#include<iostream>
#include<cmath>
using namespace std;
int main() {
    float a,b,c,m,d1,d2,d3;
    cout<<"Indique 3 numeros: "<<endl;
    cin>>a>>b>>c;
    if ((a==b)&&(a==c)){
        cout<<"Dados invalidos"<<endl;
    }
    else {
        m=(a+b+c)/3;
        cout<<"Media: "<<m<<endl;
        d1 = abs(a-m);
        d2 = abs(b-m);
        d3 = abs(c-m);
        if ((d1>d2) && (d1>d3)) {cout<<a<<" esta mais distante da media"<<endl;}
        if ((d2>d1) && (d2>d3)) {cout<<b<<" esta mais distante da media"<<endl;}
        if ((d3>d2) && (d3>d1)) {cout<<c<<" esta mais distante da media"<<endl;}
    }
}

```

12. Escreva um programa que leia a partir do teclado, a data de nascimento (dia, mês e ano) de uma pessoa e determine quantos anos completos ela terá no dia 08 de julho de 2024.

Data nascimento: (d m a)
7 7 2000
Idade: 24

Data nascimento: (d m a)
9 7 2000
Idade: 23

```

#include<iostream>
#include<cmath>
using namespace std;
int main() {
    int d,m,a,i;
    cout<<"Data nascimento: (d m a)"<<endl;
    cin>>d>>m>>a;
    if (m<7) {i=2024-a;}
    if (m>7) {i=2024-(a+1);}
    if ((m==7) and (d<=8)) {i=2024-a;}
    if ((m==7) and (d>8)) {i=2024-(a+1);}
    cout<<"Idade: "<<i<<endl;
}

```

Programação III

👉 Para você fazer

Atente que nos exemplos abaixo, tudo o que é impresso na cor azul é escrito pelo programa e na cor vermelho é digitado pelo usuário. Veja também que em geral, as mensagens não são acentuadas, para prevenir dificuldades a depender do ambiente empregado.

1. Escreva um programa que leia A[N] (com N = 10) e imprima para cada número entre 0..50 quantas vezes cada um destes números ocorre em A.

Exemplos de execução:

Informe 10 valores:

1 2 1 2 3 6 20 30 1 2

0 ocorre 0 vezes

1 ocorre 3 vezes

2 ocorre 3 vezes

3 ocorre 1 vezes

4 ocorre 0 vezes

...

50 ocorre 0 vezes

2. Suponha que o seu ambiente de programação não tem a função raiz-quadrada. Como seria possível construir uma ? Lembrando que dado um número real x sua raiz é y se e somente se $y^2 = x$. Rescrevendo $y = x/y$. Isto sugere um possível algoritmo:
- chute um valor g para y
 - Calcule $\frac{x}{g}$
 - Se x/g é suficiente perto de g retorne-o, senão tente um chute melhor.

O chute inicial pode ser qualquer número, mas vamos padronizar como sendo 1.

Informe n para raiz:

2

1.41422

Informe n para raiz:

1000

31.6228

3. (a) Escreva uma função que recebe como parâmetros dois inteiros positivos m e n, um vetor inteiro A com m números e um vetor B com n números inteiros, ambos representando conjuntos e verifica se A está contido em B ($A \in B$).
- (b) Use a função acima, para ver se os conjuntos A e B são iguais. ($A = B$ se e somente se $A \in B \wedge B \in A$)

Exemplos de execução:

Informe m: 3

Informe valores de X[m]: 1 2 4

Informe n: 5

Informe valores de Y[n]: 1 2 3 5 6

Informe m: 4

Informe valores de X[m]: 1 2 1 3

Informe n: 3

Informe valores de Y[n]: 1 2 3

A e subconjunto de B

iguais

4. Um número a é dito permutação de um número b se os dígitos de a formam uma permutação dos dígitos de b. Como por exemplo, 5412434 é uma permutação de 4321445 mas não é uma permutação de 4312455. Considere que o dígito 0 não aparece nos números.
- (a) Escreva uma função de nome *contadigitos* que recebendo um inteiro n e um inteiro d, $0 < d \leq 9$, devolve quantas vezes o dígito d aparece no número n.
- (b) Usando a função do item anterior escreva um programa que lê dois números a e b e responda se a é permutação de b.

Exemplos de execução:

Informe a e b:

54124 12445

permutacao

Informe a e b:

5412434 4312455

5. Escreva um programa que imprima os números perfeitos menores que 10000. n é perfeito se $n = \sum (divisores\ proprios\ de\ n)$. Os divisores próprios são os números que dividem n, à excessão do próprio n. Por exemplo, os divisores próprios de 6 são: 1, 2 e 3. Que somados são igual a 6, logo 6 é um número perfeito. Os divisores próprios de 8 são: 1, 2 e 4, que somados dão 7, logo 8 não é perfeito

Exemplo de execução:

1 6 28 496 8128

Espaço para avaliação do trabalho:

1	2	3	4	5

Listagem de todos os problemas de 136d

1. Escreva um programa que chame uma função para calcular a hipotenusa de um triângulo retângulo. A função deve receber os dois catetos e devolver a hipotenusa. Depois o programa deve receber do teclado os dois catetos, chamar a função e imprimir o resultado.

Exemplo de execução:

Calculo de hipotenusa - informe catetos:

3 4

Hipotenusa: 5

Outro exemplo:

alculo de hipotenusa - informe catetos:

1 2

Hipotenusa: 2.23607

```
#include<iostream>
#include<cmath>
using namespace std;
float pitagoras(float cateto1, float cateto2) {
    float a,b;
    a = cateto1 * cateto1;
    b = cateto2 * cateto2;
    return sqrt(a+b);
}
int main() {
    float c1,c2,hip;
    cout<<"Calculo de hipotenusa - informe catetos: "<<endl;
    cin>>c1;
    cin>>c2;
    hip = pitagoras(c1,c2);
    cout<<"Hipotenusa: "<<hip;
}
```

2. Os egipcios antigos (há mais de 5.000 anos) já conheciam o conceito de π não com esse nome, claro, mas como a constante que iguala o comprimento de uma circunferência ao seu raio. Como eles não dominavam a aritmética real que temos hoje, π era representado por uma fração ordinária, nomeadamente $22/7$. Seu problema aqui é receber um número real qualquer com bastantes casas decimais e localizar qual a fração de números inteiros que chega mais perto do valor do número real original. Numa primeira abordagem ambos – numerador e denominador – devem ser menores do que 100 (ou seja, com 2 dígitos) e no segundo caso devem ser menores do que 1000 (ou seja, com 3 dígitos). Veja-se alguns exemplos:

número	n	d
3.141592654 c/ 2 dígitos	22	7
3.141592654 c/ 3 dígitos	355	113
2.718281828 c/ 2 dígitos	87	32
2.718281828 c/ 3 dígitos	878	323

Escreva um programa que ache as frações inteiras para um determinado real fornecido, no primeiro caso com números de 2 dígitos e no segundo caso com números de 3 dígitos. Use uma função de nome **egipcio**.

Exemplo de execução:

Informe real e limite:

3.14159265 100

22 7

```
#include<iostream>
using namespace std;
int egipcio(float nr, int lim){
    float minimo=+9999999;
    float ca,i,j,mini,minj;
    i=1.0;
    while (i<lim){
        j = 1.0;
        while (j<lim){
            ca=i/j;
            if ((abs(nr-ca))<minimo){
                minimo=abs(nr-ca);
                mini=i;
                minj=j;
            }
            j=j+1;
        }
        i=i+1;
    }
    cout<<mini<<' '<<minj<<endl;
}
int main(){
    float re;
    int li;
    cout<<"Informe real e limite: "<<endl;
    cin>>re>>li;
    egipcio(re,li);
}
```

3. Escreva um programa que imprima os números perfeitos menores que 10000. n é perfeito se $n = \sum (divisores\ proprios\ de\ n)$. Os divisores próprios são os números que dividem n , à excessão do próprio n . Por exemplo, os divisores próprios de 6 são: 1, 2 e 3. Que somados são igual a 6, logo 6 é um número perfeito. Os divisores próprios de 8 são: 1, 2 e 4, que somados dão 7, logo 8 não é perfeito

Exemplo de execução:

1 6 28 496 8128

```
#include<iostream>
using namespace std;
int divisores(int n){
    int x,som;
```

```

som=1;
x=2;
while (x<n){
if ((n%x)==0){som=som+x;}
x++;
}
return som;
}
int main() {
int sem=1;
while (sem<100000){
if (sem==divisores(sem)){cout<<sem<<" ";}
sem++;
}
}
}

```

4. Suponha que o seu ambiente de programação não tem a função raiz-quadrada. Como seria possível construir uma ? Lembrando que dado um número real x sua raiz é y se e somente se $y^2 = x$. Reescrevendo $y = x/y$. Isto sugere um possível algoritmo:

- chute um valor g para y
- Calcule $\frac{x}{g}$
- Se x/g é suficiente perto de g retorne-o, senão tente um chute melhor.

O chute inicial pode ser qualquer número, mas vamos padronizar como sendo 1.

Informe n para raiz:

2
1.41422

Informe n para raiz:

1000
31.6228

```

#include<iostream>
using namespace std;
int perto(float a, float b){
return b*0.001 > abs(a-b);
}
float melhor(float a, float b){
return (b+(a/b))/2.0;
}
float testa(float a, float b){
if (perto((a/b),b)){
return b;}
else {
return testa(a, melhor(a,b));
}
}
int main(){
float a;
cout<<"Informe n para raiz: ";
cin>>a;
cout<<testa(a,1);
}

```

5. Escreva um programa que leia n e k ($n \geq k$, não precisa testar) e imprima a soma dos números primos existentes entre n e k inclusive.

Exemplo de execução:

Informe N e K:

100 200
3167

Informe N e K:

20 30
52

```

#include<iostream>
using namespace std;
int eprimo(int n) {
int divs,valor;
divs=0;
valor=2;
while(valor<n){
if ((n%valor)==0){
divs=divs+1;
}
valor=valor+1;
}
return divs==0;
}
int main(){
int N, K;
int soma=0;
int xave;
cout<<"Informe N e K: ";
cin>>N>>K;
xave=N;
while (xave<=K){
if (eprimo(xave)){
soma=soma+xave;
}
}
}

```

```

xave=xave+1;
}
cout<<soma;
}

```

6. Sabendo que

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

escreva um programa que leia n , inteiro e ≥ 2 e imprima o valor de e calculado com n parcelas.

```

Alguns exemplos:
Informe parcelas:
3
E = 2.5

Informe parcelas:
20
E = 2.7182819843292236328

```

```

#include<iostream>
using namespace std;
int fatr(int x){ // solucao recursiva
if (x<2){
return 1;
}
else{
return x*fatr(x-1);
}
}
int main() {
float nepe=0.0;
int parc;
int k=0;
cout<<"Informe parcelas: ";
cin>>parc;
while (k<parc){
nepe=nepe+1.0/fatr(k);
k=k+1;
}
cout.precision(20);
cout<<"E = "<<nepe;
}

```

7. Escreva um programa que leia uma sequência de inteiros (até chegar a sentinela 0) e para cada valor N lido imprima o seu dígito das centenas.

```

Exemplos de execução:
1234
2
567
5
100000
0
0

```

```

#include<iostream>
using namespace std;
int centena(int n){
int x,y;
y=n/100;
x=y%10;
return x;
}
int main() {
int lido;
cin>>lido;
while (lido !=0){
cout<<centena(lido)<<endl;
cin>>lido;
}
return 0;
}

```

8. Escreva um programa que leia $A[N]$ e $B[N]$ (com $N = 7$). Calcular e imprimir $A \cap B$.

```

Exemplos de execução
Informe 7 valores para A:
1 3 5 7 8 9 10
Informe 7 valores para B:
2 3 4 5 6 7 8
0 conjunto interseccao com tamanho 4 eh:
3 5 7 8

```

```

#define N 7
#include<iostream>
using namespace std;
int main(){
int a[N], b[N], r[N];
int i,j,k;
cout<<"Informe "<<N<<" valores para A: "<<endl;
for (i=0;i<N;i++){

```

```

cin>>a[i];
}
cout<<"Informe "<<N<<" valores para B: "<<endl;
for (i=0;i<N;i++){
cin>>b[i];
}
k=0;
for(i=0;i<N;i++){
for(j=0;j<N;j++){
if (a[i]==b[j]){
r[k]=a[i];
k++;
}
}
}
cout<<"O conjunto interseccao com tamanho "<<k<<" eh: "<<endl;
for(i=0;i<k;i++){
cout<<r[i]<<" ";
}
}

```

9. Escreva um programa que leia A[N] (com $N = 10$) e imprima para cada número entre 0..50 quantas vezes cada um destes números ocorre em A.

Exemplos de execução:

```

Informe 10 valores:
1 2 1 2 3 6 20 30 1 2
0 ocorre 0 vezes
1 ocorre 3 vezes
2 ocorre 3 vezes
3 ocorre 1 vezes
4 ocorre 0 vezes
...
50 ocorre 0 vezes

```

```

#include<iostream>
#define N 10
using namespace std;
int main(){
int v[N];
int i,q,j;
cout<<"Informe "<<N<<" valores "<<endl;
for (i=0;i<N;i++){
cin>>v[i];
}
for (i=0;i<51;i++){
q=0;
for (j=0;j<N;j++){
if (i==v[j]){q++;}
}
cout<<i<<" ocorre "<<q<<" vezes"<<endl;
}
}

```

10. Dados dois inteiros m e n determinar dentre todos os pares de números inteiros x e y , tais que $0 \leq x \leq m$ e $0 \leq y \leq n$, um par para o qual o valor da expressão $xy - x^2 + y$ seja máximo. Mostrar o valor do máximo e de x e y .

Exemplos de execução

```

Informe M e N:
1000 1000
0 maximo e 251000 com 500 e 1000

Informe M e N:
20 10
0 maximo e 35 com 5 e 10

```

```

#include<iostream>
#define N 10
using namespace std;
int main(){
int x,y,m,n,gy,gx,max=-99999;
cout<<"Informe M e N: ";
cin>>m>>n;
for (x=0;x<=m;x++){
for (y=0;y<=n;y++){
if ((y+(x*y)-(x*x))>max){
max=y+(x*y)-(x*x);
gy=y;
gx=x;
}
}
}
cout<<" 0 maximo e "<<max<<" com "<<gx<<" e "<<gy;
}

```

11. Dado um inteiro positivo n determinar todos os inteiros entre 1 e n que podem ser comprimentos de uma hipotenusa de um triângulo retângulo com catetos inteiros.

Exemplos de execução

```

Informe n:
20
5: com catetos: 3 e 4

```

```

10: com catetos: 6 e 8
13: com catetos: 5 e 12
15: com catetos: 9 e 12
17: com catetos: 8 e 15
20: com catetos: 12 e 16

```

```

#define N 10
using namespace std;
int main(){
    int n,i,j,k;
    cout<<"Informe n: ";
    cin>>n;
    for (i=5;i<=n;i++){
        for(j=1;j<=i;j++){
            for (k=1;k<=i;k++){
                if (((j*j)+(k*k))==(i*i)){
                    cout<<i<<" com catetos: "<<j<<" e "<<k<<endl;
                    k=99999;
                    j=99999;
                }
            }
        }
    }
}

```

12. Um número a é dito permutação de um número b se os dígitos de a formam uma permutação dos dígitos de b . Como por exemplo, 5412434 é uma permutação de 4321445 mas não é uma permutação de 4312455. Considere que o dígito 0 não aparece nos números.

- Escreva uma função de nome *contadigitos* que recebendo um inteiro n e um inteiro d , $0 < d \leq 9$, devolve quantas vezes o dígito d aparece no número n .
- Usando a função do item anterior escreva um programa que lê dois números a e b e responda se a é permutação de b .

Exemplos de execução:

```

Informe a e b:
54124 12445
permutacao

```

```

Informe a e b:
5412434 4312455

```

```

#include<iostream>
#define N 10
using namespace std;
int contadigitos(int n, int d){
    int qtd=0;
    while (n>0){
        if ((n%10)==d){ qtd++;}
        n=n/10;
    }
    return qtd;
}
int main(){
    int a,b,i,nao=0;
    cout<<"Informe a e b: ";
    cin>>a>>b;
    for (i=1;i<10;i++){
        if (contadigitos(a,i)!=contadigitos(b,i)){nao=1;}
    }
    if (nao==0){cout<<"permutacao"<<endl;}
}

```

- Escreva uma função que recebe como parâmetros dois inteiros positivos m e n , um vetor inteiro A com m números e um vetor B com n números inteiros, ambos representando conjuntos e verifica se A está contido em B ($A \in B$).
 - Use a função acima, para ver se os conjuntos A e B são iguais. ($A = B$ se e somente se $A \in B \wedge B \in A$)

Exemplos de execução:

```

Informe m: 3
Informe valores de X[m]: 1 2 4
Informe n: 5
Informe valores de Y[n]: 1 2 3 5 6

```

```

Informe m: 4
Informe valores de X[m]: 1 2 1 3
Informe n: 3
Informe valores de Y[n]: 1 2 3
A e subconjunto de B
iguais

```

```

#include<iostream>
using namespace std;
int subconj(int m, int n, int A[100], int B[100]){
    int i,j,simnao,ctd=0;
    for (i=0;i<m;i++){
        simnao=0;
        for(j=0;j<n;j++){
            if (A[i]==B[j]){simnao=1;}
        }
        ctd=ctd+simnao;
    }
    if (ctd==m){return 1;}
}

```



```

else {return 0;}
}
int main(){
int X[100],Y[100];
int m,n,i,r;
cout<<"Informe m: ";
cin>>m;
cout<<"Informe valores de X[m]: ";
for(i=0;i<m;i++){
cin>>X[i];
}
cout<<"Informe n: ";
cin>>n;
cout<<"Informe valores de Y[n]: ";
for(i=0;i<n;i++){
cin>>Y[i];
}
r=subconj(m,n,X,Y);
if (r==1){cout<<"A e subconjunto de B"<<endl;}
if (subconj(m,n,X,Y)&&subconj(n,m,Y,X)){cout<<"iguais"<<endl;}
}

```

Programação IV

👉 Para você fazer

Atente que nos exemplos abaixo, tudo o que é impresso na cor azul é escrito pelo programa e na cor vermelho é digitado pelo usuário. Veja também que em geral, as mensagens não são acentuadas, para prevenir dificuldades a depender do ambiente empregado.

1. Escreva um programa que receba dois valores representando a concentração inicial de duas bactérias e outros dois valores com as respectivas taxas diárias de crescimento das culturas de bactérias. O programa deve calcular e mostrar o número de dias necessários para que a cultura da primeira bactéria ultrapasse a da segunda. Assumir que a primeira taxa é maior do que a segunda e sempre será um valor entre zero e um.

```
Concentração inicial da bacteria 1:
40
Taxa de crescimento da bacteria 1 (entre 0 e 1):
0.5
Concentração inicial da bacteria 2:
100
Taxa de crescimento da bacteria 1 (entre 0 e 1):
0.1
A 1a bacteria ultrapassa a 2a em 3 dia(s)
```

2. Escreva um programa que leia um número inteiro positivo N e imprima o próximo inteiro primo P tal que $P \geq N$. Um número primo é aquele que só tem 2 divisores que são a unidade e o próprio número. Assim, 13 é primo porque seus divisores são apenas 1 e 13, enquanto 12 não é primo, já que seus divisores são 1,2,3,4,6 e 12.

```
Exemplos de execução:
12 13
29 29
49 53
```

3. Faça um programa que leia um vetor V de inteiros com um número fixo N de elementos indicado via `#define`, e rotacione seus elementos para a direita, fazendo com que o valor de V_{i+1} receba o valor de V_i . Cuidado com as extremidades, o primeiro elemento deve receber o último elemento do vetor. Além disso, o programa deve imprimir o somatório de todos os elementos rotacionados.

```
Exemplo de execução (com N=6):
Entre vetor: 7 2 3 -1 9 0
Vetor rotacionado: 0 7 2 3 -1 9
Soma dos elementos: 20
```

4. após seus treinos para uma competição de natação, Alfredo quer verificar seu desempenho em relação a seus tempos de treino e ao tempo recorde regional. Para ajudar nessa tarefa, você deve escrever um programa que solicite a quantidade N de treinos realizados, o tempo do último recorde regional (em segundos) e para cada treino solicite o tempo cronometrado em segundos. Para o conjunto de tempos, seu programa deve calcular o maior e o menor tempo e a média aritmética dos tempos de treino. No final, se o tempo médio for menor que o recorde informado deve mostrar uma mensagem indicando tal fato.

```
Exemplo de execução:
Quantidade de treinos: 5
Recorde regional: 47
Tempo 1:45
Tempo 2:50
Tempo 3:44
Tempo 4:46
Tempo 5:48
Maior tempo: 50 segundos
Menor tempo: 44 segundos
Média: 46.6 segundos
Você está abaixo do recorde !
```

```
Mais um exemplo de execução:
Quantidade de treinos: 3
Recorde regional: 47
Tempo 1:47
Tempo 2:48
Tempo 3:49
Maior tempo: 49 segundos
Menor tempo: 47 segundos
Média: 48 segundos
```

5. Faça um programa que leia dois inteiros h e m tais que $0 \leq h < 24$ e $0 \leq m < 60$ que representam o horário (h horas e m minutos) do início de uma determinada tarefa a ser realizada. Em seguida, seu programa deve solicitar ao usuário um número inteiro n que representa a quantidade de minutos necessários para a realização dessa tarefa. Por fim, o programa deve imprimir na tela qual o horário de término da tarefa realizada e uma mensagem indicando se a tarefa foi curta (duração de menos de 2 horas) ou longa.

```
horario de inicio:09 32
duracao da tarefa:45
horario de termino: 10:17 hrs
tarefa curta
Outro exemplo de execução:
horario de inicio:21 0
duracao da tarefa:350
horario de termino: 02:50 hrs
tarefa longa
```

Espaço para avaliação do trabalho:

1	2	3	4	5

Listagem de todos os problemas de 136e

1. Escreva um programa que leia um número inteiro positivo N e imprima o próximo inteiro primo P tal que $P \geq N$. Um número primo é aquele que só tem 2 divisores que são a unidade e o próprio número. Assim, 13 é primo porque seus divisores são apenas 1 e 13, enquanto 12 não é primo, já que seus divisores são 1,2,3,4,6 e 12.

Exemplos de execução:

12 13
29 29
49 53

```
#include<iostream>
using namespace std;
//--- imprime o próximo primo ----
int eprimo(int x){
    int i=2,d=0;
    while (i<x){
        if (x%i==0){d++;}
        i++;
    }
    return d==0;
}

int proxprim(int n){
    while (!eprimo(n)){
        n++;
    }
    return n;
}

int main(){
    int n;
    cout<<"Informe N: ";
    cin>>n;
    cout<<proxprim(n);
}
```

2. Escreva uma função de nome dp que receba um inteiro n ($1 < n \leq 10$) e um vetor de até 10 flutuantes e devolva o desvio padrão dos elementos do vetor. n sinaliza quantos dos elementos do vetor realmente estão sendo usados nesta chamada, sempre alinhados à esquerda do vetor. Como neste cálculo é necessário conhecer a média aritmética, a função que a calcula deve obrigatoriamente ser usada. O desvio padrão tem como fórmula

$$dp = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

Por exemplo, se a função for chamada com 10,20,30,40,50, a média como vimos será 30, e o programa deverá calcular as diferenças: $10 - 30 = -20$, $20 - 30 = -10$, $30 - 30 = 0$, $40 - 30 = 10$ e $50 - 30 = 20$. Elevando estes números ao quadrado $-20^2 = 400$, $-10^2 = 100$, $0^2 = 0$, $10^2 = 100$ e $20^2 = 400$. Somando tudo, fica-se com $400 + 100 + 0 + 100 + 400 = 1000$ e dividindo por 5 fica $1000/5 = 200$. O desvio padrão é $\sqrt{200} = 14.14$. Depois, usando as duas funções acima, escreva um programa que leia n (inteiro e positivo) e depois n valores reais, imprimindo ao final o desvio padrão dessa amostra.

Exemplo de execução:

5
10 20 30 40 50
14.1421

```
#include<cmath>
using namespace std;
float media(int n, float v[10]){
    float s=0;
    int i;
    for(i=0;i<n;i++){
        s=s+v[i];
    }
    return s/n;
}

//----- desvio padrão -----
float dp(int n, float v[10]){
    float me;
    float dvs=0;
    int i;
    me=media(n,v);
    for (i=0;i<n;i++){
        dvs=dvs+((me-v[i])*(me-v[i]));
    }
    dvs=dvs/n;
    return sqrt(dvs);
}

int main(){
    int n,i;
    float v[10];
    cin>>n;
    for (i=0;i<n;i++){
        cin>>v[i];
    }
    cout<<dp(n,v);
}
```

3. Escreva uma função de nome $somacum$ que receba o inteiro n ($n \leq 10$) e um vetor x de n inteiros e devolva outro vetor também de n inteiros onde o elemento i é igual a $\sum_{j=0}^{j=i} x_j$. Depois escreva a função principal que lê n e v e imprime o resultado da aplicação da função sobre o vetor.

Exemplo de execução:

```
5
1 2 3 4 5
1 3 6 10 15
```

Mais uma:

```
4
88 12 3 45
88 100 103 148
```

```
#include<iostream>
#include<cmath>
using namespace std;
void somacum(int n, int ve[10], int vs[10]){
    int i,s,j;
    vs[0]=ve[0];
    for (i=1;i<n;i++){
        s=0;
        for(j=0;j<=i;j++){
            s=s+ve[j];
        }
        vs[i]=s;
    }
}
//

int main(){
    int ve[10], vs[10], n, i;
    cin>>n;
    for(i=0;i<n;i++){cin>>ve[i];}
    somacum(n,ve,vs);
    for(i=0;i<n;i++){cout<<vs[i]<<" ";}
}
```

4. Faça um programa que leia dois inteiros h e m tais que $0 \leq h < 24$ e $0 \leq m < 60$ que representam o horário (h horas e m minutos) do início de uma determinada tarefa a ser realizada. Em seguida, seu programa deve solicitar ao usuário um número inteiro n que representa a quantidade de minutos necessários para a realização dessa tarefa. Por fim, o programa deve imprimir na tela qual o horário de término da tarefa realizada e uma mensagem indicando se a tarefa foi curta (duração de menos de 2 horas) ou longa.

```
horario de inicio:09 32
duracao da tarefa:45
horario de termino: 10:17 hrs
tarefa curta
Outro exemplo de execução:
horario de inicio:21 0
duracao da tarefa:350
horario de termino: 02:50 hrs
tarefa longa
```

```
int main(){
    int h,m,d,duracao=0;
    cin>>h>>m>>d;
    cout<<"Horario de inicio "<<h<<" "<<m<<endl;
    cout<<"Duracao "<<d<<endl;
    if (d>=120){duracao=1;}
    while (d>0){
        if(d>59){h++;d=d-60;}
        else{m=m+d;d=0;}
    }
    if (m>60){h=h+1;m=m-60;}
    if (h>23){h=h-24;}
    cout<<"Horario de fim "<<h<<":"<<m<<" hrs"<<endl;
    if (duracao==1){cout<<"Tarefa longa";}
    else{cout<<"Tarefa curta";}
}
```

5. Você está visitando os pontos turísticos de Manhattan (Nova Iorque), onde as ruas são numeradas de 1 a 155 no sentido leste/oeste (eixo x), e 1 a 16 no sentido norte sul (eixo y). Por conveniência, os endereços das atrações são dados pelos cruzamentos das ruas (exemplo: o Empire State Building fica no cruzamento da rua 5 com a rua 33). Crie um programa que leia um cruzamento de partida no formato $eixoX_1, eixoY_1$, e depois leia sucessivos cruzamentos de ruas em que você fez paradas para visitar algum ponto turístico, também no formato $eixoX_2, eixoY_2$. A cada cruzamento de parada exiba a distância percorrida desde a última parada, e também a distância total percorrida até o momento (ambas medidas em quadras). Em Manhattan, a distância entre dois cruzamentos de rua é dada por $|eixoX_2 - eixoX_1| + |eixoY_2 - eixoY_1|$ onde esta distância é medida em quadras. Não se esqueça de validar se o usuário digitou uma coordenada de rua válida. O programa deve terminar quando o usuário digitar -1 -1 para as coordenadas de rua.

```
Informe coordenadas:
- cruzamento de partida:5 4
- proximo cruzamento:5 5
Deslocamento: 1 quadras
Deslocamento total: 1 quadras
- proximo cruzamento:10 8
Deslocamento: 8 quadras
Deslocamento total: 9 quadras
- proximo cruzamento:9 7
Deslocamento: 2 quadras
Deslocamento total: 11 quadras
- proximo cruzamento:156 3
Cruzamento invalido!
- proximo cruzamento:155 3
Deslocamento: 150 quadras
Deslocamento total: 161 quadras
- proximo cruzamento:-1 -1
Fim!
```

```

int main(){
int xi,yi,xa,ya,dp,dt=0;
cout<<"Informe coordenadas:"<<endl;
cout<<" - cruzamento de partida ";
cin>>xi>>yi;
while (1==1){
cout<<"proximo cruzamento ";
cin>>xa>>ya;
if ((xa>155)|| (ya>16)){
cout<<"cruzamento invalido!"<<endl;
continue;
}
if ((xa==1)&&(ya==1)){cout<<"fim"<<endl; break;}
dp=abs(xa-xi)+abs(ya-yi);
cout<<"Deslocamento " <<dp<<endl;;
dt=dt+dp;
cout<<"Deslocamento total " <<dt<<endl;;
xi=xa;
yi=ya;
}
}
}

```

6. Escrever um programa que leia do teclado um vetor de N elementos inteiros (N estabelecido via `#define`), calcule a média aritmética inteira truncada simples entre os elementos com o menor e maior conteúdo no vetor e altere os elementos com valores inferiores à media obtida para -1 e os acima daquela para 1. O vetor alterado deverá ser mostrado na tela.

Exemplo de execução (com $N=11$):
Vetor (11 elementos):
0 0 11 0 0 0 2 0 0 0
Média (min, max) = (0 + 11) / 2 = 5
Vetor alterado:
-1 -1 1 -1 -1 -1 -1 -1 -1 -1

Exemplo de execução (com $N=5$):
Vetor (5 elementos):
5 1 2 3 4
Média (min, max) = (1 + 5) / 2 = 3
Vetor alterado:
1 -1 -1 3 1

```

int main(){
int v[N];
cout<<"Vetor "<<N<<" elementos:"<<endl;
int i,j,med=0,maior=-99999,menor=99999;
for (i=0;i<N;i++){cin>>v[i];}
for (i=0;i<N;i++){
if (v[i]>maior){maior=v[i];}
if (v[i]<menor){menor=v[i];}
}
med=(maior+menor)/2;
cout<<"Media (min,max) = ("<<menor<<"+ "<<maior<<")/2="<<med<<endl;
for (i=0;i<N;i++){
if (v[i]<med){v[i]=-1;}
if (v[i]>med){v[i]=1;}
}
cout<<"Vetor alterado "<<endl;
for (i=0;i<N;i++){
cout<<v[i]<<" ";
}
}
}

```

7. Faça um programa em C++ para ler 2 números naturais de 2 algarismos e verificar se eles são similares. Para serem similares estes números precisam ter os mesmos algarismos na sua formação.

Exemplo de execução:
Digite 2 números:23 32
Números similares
Outro exemplo de execução:
Digite 2 números:45 35
Números não similares
Outro exemplo de execução:
Digite 2 números:67 67
Números similares

```

int main(){
int n,m,d1n,d2n,d1m,d2m;
cin>>n>>m;
d1n=n/10;
d2n=n%10;
d1m=m/10;
d2m=m%10;
if (((d1n==d1m)|| (d1n==d2m))&&((d2n==d1m)|| (d2n==d2m))){
cout<<"numeros similares";
}
else {
cout<<"numeros nao similares";
}
}
}

```

8. Faça um programa que receba do usuário, via teclado, um intervalo de números definido por dois limites, um inferior e outro superior, valide-o (o limite inferior não pode ser maior que o superior) e, a seguir, uma lista de números de quantidade desconhecida, encerrada pela informação do valor zero pelo usuário; este valor não faz parte da lista, apenas serve para indicar seu fim. O programa deverá ler toda a lista e, ao seu final, mostrar quantos números digitados foram menores que o limite inferior, quantos ficaram entre os limites e quantos acima do limite superior definido

Exemplo de execução:
 Limites (inferior e superior):
 20 10
 Limites invalidos!

Outro exemplo de execução:
 Limites (inferior e superior):
 -3.14159 12
 Lista de números (0 = Fim):
 [1]:25
 [2]:3.14159
 [3]:-3
 [4]:12
 [5]:-10
 [6]:0
 Nos. abaixo de -3.14159: 1
 Nos. de -3.14159 a 12: 3
 Nos. acima de 12: 1

```
int main(){
    float li,ls,n;
    int qi=0,qm=0,qs=0;
    cin>>li>>ls;
    if (li>ls){cout<<"Limites invalidos";return 0;}
    cin>>n;
    while (n!=0?0){
        if (n<li){qi++;}
        if ((n>=li)&&(n<=ls)){qm++;}
        if (n>ls){qs++;}
        cin>>n;
    }
    cout<<"Numeros abaixo de "<<li<<" "<<qi<<endl;
    cout<<"Numeros entre "<<li<<" e "<<ls<<" "<<qm<<endl;
    cout<<"Numeros acima de "<<ls<<" "<<qs<<endl;
}
```

9. Faça um programa que leia um vetor V de inteiros com um número fixo N de elementos indicado via `#define`, e rotacione seus elementos para a direita, fazendo com que o valor de V_{i+1} receba o valor de V_i . Cuidado com as extremidades, o primeiro elemento deve receber o último elemento do vetor. Além disso, o programa deve imprimir o somatório de todos os elementos rotacionados.

Exemplo de execução (com $N=6$):
 Entre vetor: 7 2 3 -1 9 0
 Vetor rotacionado: 0 7 2 3 -1 9
 Soma dos elementos: 20

```
int main(){
    int v[N];
    int i,j;
    for (i=0;i<N;i++){
        cin>>v[i];
    }
    j=v[N-1];
    for (i=N-2;i>=0;i--){
        v[i+1]=v[i];
    }
    v[0]=j;
    for (i=0;i<N;i++){
        cout<<v[i]<<" ";
    }
}
```

10. após seus treinos para uma competição de natação, Alfredo quer verificar seu desempenho em relação a seus tempos de treino e ao tempo recorde regional. Para ajudar nessa tarefa, você deve escrever um programa que solicite a quantidade N de treinos realizados, o tempo do último recorde regional (em segundos) e para cada treino solicite o tempo cronometrado em segundos. Para o conjunto de tempos, seu programa deve calcular o maior e o menor tempo e a média aritmética dos tempos de treino. No final, se o tempo médio for menor que o recorde informado deve mostrar uma mensagem indicando tal fato.

Exemplo de execução:
 Quantidade de treinos: 5
 Recorde regional: 47
 Tempo 1:45
 Tempo 2:50
 Tempo 3:44
 Tempo 4:46
 Tempo 5:48
 Maior tempo: 50 segundos
 Menor tempo: 44 segundos
 Média: 46.6 segundos
 Você está abaixo do recorde !

Mais um exemplo de execução:
 Quantidade de treinos: 3
 Recorde regional: 47
 Tempo 1:47
 Tempo 2:48
 Tempo 3:49
 Maior tempo: 49 segundos
 Menor tempo: 47 segundos
 Média: 48 segundos

```
#include<iostream>
using namespace std;
```

```

int main() {
    int q,rr,t,so,ma,me,i;
    float media;
    cout<<"Quantidade de treinos: "<<endl;
    cin>>q;
    cout<<"Recorde Regional: "<<endl;
    cin>>rr;
    so=0;
    ma=-99999999;
    me=99999999;
    for (i=1;i<=q;i++){
        cout<<"Tempo "<<i<<" ";
        cin>>t;
        if (t<me) {me=t;}
        if (t>ma) {ma=t;}
    }so=so+t;
    media=so/(q*1.0);
    cout<<"Maior tempo: "<<ma<<" segundos"<<endl;
    cout<<"Menor tempo: "<<me<<" segundos"<<endl;
    cout<<"Media: "<<media<<" segundos"<<endl;
    if (me<rr) {cout<<"Voce esta abaixo do recorde"<<endl;}
}

```

11. Maria comprou vários tipos de doce de côco, de diversos fornecedores. Após provar e guardar os doces, ela quis saber qual o doce mais barato e a economia em só comprar o mais barato. Faça um programa que receba a quantidade total de doces comprados e o peso em gramas e o valor em reais de cada doce comprado pela Maria. Calcule e imprima o preço unitário de cada doce e no final informe o número e preço unitário do doce mais barato, o gasto total realizado e a economia que seria gerada se o peso total de doces fosse comprado apenas no fornecedor mais barato.

Exemplo de execução
 Total de doces comprados: 3
 Doce 1:
 Peso (g) e preço (R\$): 1000 23.34
 Preço unitário = R\$23.34/kg
 Doce 2:
 Peso (g) e preço (R\$): 500 10.51
 Preço unitário = R\$21.02/kg
 Doce 3:
 Peso (g) e preço (R\$): 1500 36.12
 Preço unitário = R\$24.08/kg
 Produto mais barato: Doce 2, R\$21,02/kg
 Foram comprados 3000g de doce por R\$69.97
 Economia de R\$6.91 se comprar 3000g do doce 2

```

#include<iostream>
using namespace std;
int main() {
    int q,peso,speso,i,qual;
    float preco,menorpreco,punit,spreco,economia;
    menorpreco=99999999;
    spreco=0;
    spento=0;
    cout<<"Quantidade de doces comprados: "<<endl;
    cin>>q;
    for (i=1;i<=q;i++){
        cout<<"Doce "<<i<<": "<<endl;
        cout<<"Peso (g) e preço (R$): ";
        cin>>peso>>preco;
        spento=speso+peso;
        spreco=spreco+preco;
        punit=(1000.0/peso)*preco;
        cout<<"Preço unitario : "<<punit<<endl;
        if (punit<menorpreco){
            menorpreco=punit;
            qual=i;
        }
    }
    cout<<"Produto mais barato : Doce "<<qual<< ", R$ "<<menorpreco<< "/kg"<<endl;
    cout<<"Foram comprados "<<speso<<"g de doce por R$"<<spreco<<endl;
    economia=spreco-((speso*menorpreco)/1000.0);
    cout<<"Economia de "<<economia<<" se comprar "<<speso<<"g do doce "<<qual<<endl;
}

```

12. Escreva um programa que receba dois valores representando a concentração inicial de duas bactérias e outros dois valores com as respectivas taxas diárias de crescimento das culturas de bactérias. O programa deve calcular e mostrar o número de dias necessários para que a cultura da primeira bactéria ultrapasse a da segunda. Assumir que a primeira taxa é maior do que a segunda e sempre será um valor entre zero e um.

Concentração inicial da bacteria 1:
 40
 Taxa de crescimento da bacteria 1 (entre 0 e 1):
 0.5
 Concentração inicial da bacteria 2:
 100
 Taxa de crescimento da bacteria 1 (entre 0 e 1):
 0.1
 A 1a bacteria ultrapassa a 2a em 3 dia(s)

```

#include<iostream>
using namespace std;
int main() {
    float b1,b2,tx1,tx2;
    int qd;
    cout<<"Concentracao inicial da bacteria 1: ";
    cin>>b1;
}

```

```

cout<<"taxa de crescimento da bateria 1 (entre 0 e 1): ";
cin>>tx1;
cout<<"Concentracao inicial da bateria 2: ";
cin>>b2;
cout<<"taxa de crescimento da bateria 2 (entre 0 e 1): ";
cin>>tx2;
qd=0;
while (b1<b2) {
qd++;
b1=b1*(1.0+tx1);
}b2=b2*(1.0+tx2);
cout<< "A 1a. bacteria ultrapassa a segunda em "<<qd<<" dias"<<endl;
}

```


Programação V

👉 Para você fazer

Atente que nos exemplos abaixo, tudo o que é impresso na cor **azul** é escrito pelo programa e na cor **vermelho** é digitado pelo usuário. Veja também que em geral, as mensagens não são acentuadas, para prevenir dificuldades a depender do ambiente empregado.

1. Escrever um programa que leia do teclado dois números inteiros positivos com 3 algarismos cada, respectivamente os limites inferior e superior de um intervalo de números inteiros. Validado o intervalo (veja os exemplos abaixo), pede-se que o programa liste e conte os números inteiros entre os dois limites (incluindo estes) em que o dígito da dezena seja 3, 5 ou 7.

```
Exemplo de execução:
Informe limites (inferior e superior):
128 115
Limites invalidos

Outro exemplo de execução:
Informe limites (inferior e superior):
23 12000
Limites invalidos

Outro exemplo de execução:
Informe limites (inferior e superior):
110 150
130 131 132 133 134 135 136 137 138 139 150
Nos. listados: 11
```

2. Escreva um programa que leia dois inteiros *A* e *B* e imprima a mensagem NULO, POSITIVO ou NEGATIVO conforme o resultado da multiplicação de *A* por *B* SEM REALIZAR A OPERAÇÃO.

```
Exemplo de execução:
10 12
POSITIVO
Outro exemplo:
16 -34
NEGATIVO
Outro exemplo:
0 -34
NULO
```

3. Escreva um programa que calcule e exiba os tempos gastos e alturas percorridas por uma lesma em suas escaladas em arranha céus. Para isso, o usuário deve informar a altura a ser escalada (*A*), a quantidade de metros que a lesma consegue subir por turno de 12 horas de subida (*V_s*) e a quantidade de metros que ela escorrega por turno de 12 horas de descanso (*V_d*). Os valores de entrada para *A* e *V_s* são positivos e não nulos. O valor de *V_d* está no intervalo [0, *V_s*). Caso estas restrições não sejam satisfeitas, o programa deve terminar sem realizar nenhum cálculo exibindo mensagem que descreva o fato transgressor. A escalada deve ser descrita como nos exemplos a seguir:

```
Exemplo de execução:
Altura total a escalar (m): 33
Escalada de subida (m/12h): 10
Escorregamento de descanso (m/12h): 2
Desenvolvimento da escalada:
0h / 0m
12h / 10m
24h / 8m
36h / 18m
48h / 16m
60h / 26m
72h / 24m
82.8h / 33m <objetivo atingido>

Note que 82.8h=33m-24m=9m/10m=0.9*12h=10.8h+72h=82.8h

Outro exemplo de execução:
Altura total a escalar (m): 5
Escalada de subida (m/12h): 10
Escorregamento de descanso (m/12h): 2
Desenvolvimento da escalada:
0h / 0m
6h / 5m <objetivo atingido>

Note que 6hh=5m-0m=5m/10m=0.5*12h=6h+0h=6h
```

4. Um programa para gerenciar as contas de um banco deve possuir algum mecanismo para calcular o saldo dos clientes após as movimentações

diárias. Escreva um programa que receba o valor da quantia em conta, e após o código da operação realizada e o valor associado. O programa deve imprimir o saldo em conta. Se o saldo for negativo, ele deve imprimir mensagem informando que o cliente entrou no cheque especial. As operações permitidas são

100

depósito em dinheiro

101

depósito em cheque

102

depósito de outra conta

200

saque em dinheiro

201

saque com cheque

202

saque para outra agência

```
Exemplo de execução:
Saldo :2000
202 1000
Saldo em conta: 1000
Mais um exemplo de execução:
Saldo :1000
200 1500
Voce entrou no cheque especial !
```

Espaço para avaliação do trabalho:

1	2	3	4

- 1 -

Listagem de todos os problemas de 136f

1. Escrever um programa que leia do teclado dois numeros inteiros positivos com 3 algarismos cada, respectivamente os limites inferior e superior de um intervalo de numeros inteiros. Validado o intervalo (veja os exemplos abaixo), pede-se que o programa liste e conte os números inteiros entre os dois limites (incluindo estes) em que o dígito da dezena seja 3, 5 ou 7.

Exemplo de execução:

```
Informe limites (inferior e superior):  
128 115  
Limites invalidos
```

Outro exemplo de execução:

```
Informe limites (inferior e superior):  
23 12000  
Limites invalidos
```

Outro exemplo de execução:

```
Informe limites (inferior e superior):  
110 150  
130 131 132 133 134 135 136 137 138 139 150  
Nos. listados: 11
```

```
#include<iostream>  
using namespace std;  
int main() {  
    int ni,ns,x,y,qt;  
    cout<<"Informe limites (inferior e superior): ";  
    cin>>ni>>ns;  
    if (ni>ns) {cout<<"Limites invalidos"<<endl;}  
    else {  
        if ((ni>999) || (ns>999)) {cout<<"Limites invalidos "<<endl;}  
        else {  
            qt=0;  
            while (ni<=ns) {  
                x=ni;  
                y=(x-(x%10))/10;  
                if ((y%10)==3 || ((y%10)==5) || ((y%10)==7)){  
                    cout<<ni<<" ";  
                    qt++;  
                }  
                ni++;  
            }  
            cout<<endl<<"Nos. listados "<<qt<<endl;  
        }  
    }  
}
```

2. Escreva um programa que leia a partir do teclado uma quantidade arbitrária de pares de números reais, sendo um par de zeros a identificação final dos pares e determine a média dos produtos positivos e dos produtos negativos destes pares

Exemplo de execução:

```
73.5 9.8  
17.3 -25.3  
-8.7 19.5  
-22.9 -32.7  
91.2 19.2  
0 0  
Media dos produtos positivos:  
1073.39  
Media dos produtos negativos:  
-303.67
```

```
#include<iostream>  
using namespace std;  
int main() {  
    float p1,p2,mpos,spos,mneg,sneg;  
    int qpos,qneg;  
    p1=1;  
    p2=1;  
    spos=0;  
    sneg=0;  
    qpos=0;  
    qneg=0;  
    while (1==1) {  
        cin>>p1>>p2;  
        if ((p1==0) && (p2==0)) {break;}  
        if ((p1*p2)>=0) {  
            spos=spos+(p1*p2);  
            qpos++;  
        }  
        else {  
            sneg=sneg+(p1*p2);  
            qneg++;  
        }  
    }  
    mpos=spos/qpos;  
    mneg=sneg/qneg;  
    cout<<"Media dos produtos positivos: "<<endl;  
    cout<<mpos<<endl;  
    cout<<"Media dos produtos negativos: "<<endl;  
    cout<<mneg<<endl;  
}
```

3. Um programa para gerenciar as contas de um banco deve possuir algum mecanismo para calcular o saldo dos clientes após as movimentações diárias. Escreva um programa que receba o valor da quantia em conta, e após o código da operação realizada e o valor associado. O programa deve imprimir

o saldo em conta. Se o saldo for negativo, ele deve imprimir mensagem informando que o cliente entrou no cheque especial. As operações permitidas são

```
100 depósito em dinheiro
101 depósito em cheque
102 depósito de outra conta
200 saque em dinheiro
201 saque com cheque
202 saque para outra agência
```

Exemplo de execução:

```
Saldo :2000
202 1000
Saldo em conta: 1000
Mais um exemplo de execução:
Saldo :1000
200 1500
Voce entrou no cheque especial !
```

```
using namespace std;
int main() {
    int saldo,cod,valor;
    cout<<"Saldo: ";
    cin>>saldo;
    cin>>cod>>valor;
    if (cod<103) {saldo=saldo+valor;}
    if (cod>102) {saldo=saldo-valor;}
    if (saldo >=0) {
        cout<<"Saldo em conta: "<<saldo<<endl; }
    else {
        cout<<"Voce entrou no cheque especial !"<<endl;}
}
```

4. Escreva um programa que leia dois números inteiros x e n sendo n não negativo e que calcule o valor de x^n . Não é permitido o uso da função `float pow(float x, float y)` ou qualquer outra função pré-existente.

Exemplo de execução:

```
Digite valores de x e n:3 4
0 valor de 3^4 e: 81
```

Outro exemplo:

```
Digite valores de x e n:-3 3
0 valor de -3^3 e: -27
```

Outro exemplo:

```
Digite valores de x e n:3 -3
0 expoente deve ser positivo
```

Outro exemplo:

```
Digite valores de x e n:3 0
0 valor de 3^0 e: 1
```

```
#include<iostream>
using namespace std;
int main() {
    int x,n,r,i;
    r=1;
    cout<<"Digite valores x e n: ";
    cin>>x>>n;
    if (n<0) {cout<<"0 expoente deve ser positivo";}
    else {
        for (i=1;i<=n;i++){
            r=r*x;
        }
        cout<<"0 valor de "<<x<<"^"<<n<<" e: "<<r;
    }
}
```

5. Um atleta treina em uma pista de corrida circular de 400 metros. Ao final do treino, com base na quantidade de voltas dadas e no tempo total do treino (em horas, minutos e segundos), ele quer saber:

- O percurso total do treino em quilômetros
- O tempo médio por quilômetro (em horas, minutos e segundos)
- o tipo de atleta de acordo com o tempo médio por quilômetro:
 - Atleta de competição, se o tempo médio está no intervalo entre 0 e 4 minutos (inclusive);
 - Atleta participante, se o tempo médio está acima de 4 e abaixo de 7 minutos (7 inclusive);
 - Participante de caminhada, se o tempo médio é acima de 7 minutos.

Escreva um programa que auxilie o atleta a ter essas respostas. Para o tempo, se a sua solução tiver valores abaixo de segundos, ignore a parte decimal/fracionária após os segundos.

Exemplo de execução:

```
Numero de voltas: 13
Tempo do treino (hh mm ss): 0 55 23
Percurso total: 5.2
Tempo medio (km): 0 10 39
Tipo: Participante de caminhada
```

Outro exemplo de execução:

```
Numero de voltas: 13
Tempo do treino (hh mm ss): 0 20 30
Percurso total: 5.2
Tempo medio (km): 0 3 57
Tipo: Atleta de competicao
```

```

#include<iostream>
using namespace std;
int main() {
    int voltas,h,m,s,h1,m1,s1,tm,tt;
    float ptotal;
    cout<<"Numero de voltas: ";
    cin>>voltas;
    cout<<"Tempo treino (hh mm ss): ";
    cin>>h>>m>>s;
    ptotal=voltas*0.4;
    tt=(h*3600)+(m*60)+s;
    tm=tt/ptotal;
    h1=tm/3600;
    m1=(tm-(h1*3600))/60;
    s1=tm-((h1*3600)+(m1*60));
    cout<<"Percurso total "<<ptotal<<endl;
    cout<<"Tempo medio/km (hh mm ss): "<<h1<<" "<<m1<<" "<<s1<<endl;
    if (m1<5) {cout<<"Tipo: atleta de competicao"<<endl;}
    else {if (m1<8) {cout<<"Tipo: atleta participante"<<endl;}
    }else {cout<<"Tipo: participante de caminhada"<<endl;}
}

```

6. Escreva um programa que leia do teclado conjuntos de três medidas a , b e c (reais e positivos) correspondendo aos lados de um triângulo. Para cada tripla lida, o programa deve verificar e informar se é ou não é possível montar um triângulo. Sendo possível, o perímetro do triângulo deverá ser calculado e mostrado. Não sendo possível, calcular e mostrar a dimensão mínima acima da qual, a adição desta a algum dos lados menores informados possibilitará a formação de um triângulo. O conjunto de triplas se encerra com a informação da primeira medida nula ou negativa, esta tripla não deve ser processada e sinaliza final de dados. Ao final do programa devem ser mostrados o total de triângulos possíveis de montagem, seu perímetro médio, o total de triângulos não possível de montagem, a média dos incrementos faltantes e o total de triplas lidas. Seu programa não precisa validar se as medidas informadas são positivas. Lembre que para formar um triângulo a soma de dois lados quaisquer tem que ser maior que o outro lado. Ou seja dados a , b e c , eles só formam triângulo se $a + b > c \wedge b + c > a \wedge a + c > b$.

Exemplo de execução:

```

Medidas: 3 4 5
Formam um triangulo. Perimetro: 12
Medidas: 10 2 4
Nao formam um triangulo.
Incremento minimo > 4
Medidas: 2.5 3 4.5
Formam um triangulo. Perimetro: 10
Medidas: 6 6 5
Formam um triangulo. Perimetro: 17
Medidas: 1 1 7
Nao formam um triangulo.
Incremento minimo > 5
Medidas: -3 1 2
Triangulos montados: 3
Perimetro medio: 13
Triangulos nao montados: 2
Media dos incrementos faltantes: 4.5
Total de triplas: 5

```

```

#include<iostream>
using namespace std;
int main() {
    float a,b,c,mai,men,q,sp,si,qn,mm,mn;
    sp=0; si=0; q=0; qn=0;
    while (1==1){
        cout<<"Medidas: ";
        cin>>a>>b>>c;
        if (a<0) {break;}
        if (((a+b)>c) && ((a+c)>b) && ((b+c)>a)) {
            cout<<"Formam um triangulo. ";
            cout<<"Perimetro: "<<a+b+c<<endl;
            q=q+1;
        }sp=sp+a+b+c;
        else {
            if ((a>b)&&(a>c)){mai=a; men=b+c;}
            if ((b>c)&&(b>a)){mai=b; men=a+c;}
            if ((c>a)&&(c>b)){mai=c; men=a+b;}
            cout<<"Nao formam um triangulo"<<endl;
            cout<<"Incremento minimo > "<<mai-men<<endl;
            si=si+(mai-men);
        }qn=qn+1;
        mm=sp/q;
        mn=si/qn;
        cout<<"Triangulos montados: "<<q<<endl;
        cout<<"Perimetro medio: "<<mm<<endl;
        cout<<"Triangulos nao montados: "<<qn<<endl;
        cout<<"Media incrementos faltantes: "<<mn<<endl;
        cout<<"Total de triplas: "<<q+qn<<endl;
    }
}

```

7. Escreva um programa que leia 4 números inteiros H_a , M_a , H_d e M_d onde $H_a : M_a$ representa o horário atual e $H_d : M_d$ representa o horário no qual o despertador deve tocar. Seu programa deve calcular e mostrar quantas horas e quantos minutos faltam até que o alarme desperte. Se os horários forem identicos, o programa deve indicar que o alarme está despertando e se o horário programado for anterior ao atual, deve indicar que o alarme já despertou. Considere que as horas estarão entre 0 e 23 e os minutos entre 0 e 59. Além disso, considere que o horário programado no despertador é referente ao dia atual e nunca ao dia seguinte.

Exemplo de execução:

```

10 12
14 15
4 horas e 3 minutos para despertar

```

Outro exemplo de execução:

```
16 34
22 06
5 horas e 32 minutos para despertar
```

```
23 42
03 06
Alarme ja despertou!
```

```
09 42
09 42
Alarme esta despertando!
```

```
#include<iostream>
using namespace std;
int main() {
    int ha,ma,hd,md,du;
    cin>>ha>>ma;
    cin>>hd>>md;
    if ((ha==hd) &&(ma==md)) {
        cout<<"Alarme esta despertando"<<endl;
    }return 0;
    if ((ha>hd) || ((ha==hd) && (ma > md))) {
        cout<<"Alarme ja despertou"<<endl;
    }return 0;
    du=((hd*60)+md)-((ha*60)+ma);
    cout<<(du/60)<<" horas e "<<du%60<<" minutos para despertar"<<endl;
}
```

8. Escreva um programa que leia um conjunto de valores inteiros $n_1, n_2, \dots, n_d$ representando as faces obtidas em d lançamentos de um dado de seis faces e imprima uma mensagem indicando se o dado é regular ou está viciado. Considere que todo valor lido n_i está entre 1 e 6, isto é $1 \leq n_i \leq 6$ e que o final do conjunto de valores é indicado quando o usuário digitar o valor zero que não deve ser contado como um lançamento de dado. Considere que o dado é viciado se o número de ocorrências de alguma de suas faces for maior que o dobro do valor esperado, definido pelo número total de lançamentos multiplicado pela probabilidade de ocorrência da respectiva face, isto é: $viciado \Leftrightarrow alguma\ face > 2 \times total \times \frac{1}{6}$

Exemplo de execução:

```
1 1 2 5 4 6 0
Dado regular
```

Outro exemplo de execução:

```
6 4 3 4 1 4 4 2 0
Dado viciado
```

```
#include<iostream>
using namespace std;
int main() {
    int da,a1,a2,a3,a4,a5,a6,q;
    a1=0; a2=0; a3=0; a4=0; a5=0; a6=0; q=0;
    while (1==1){
        cin>>da;
        if (da==0) {break;}
        q++;
        if (da==1) {a1++;}
        if (da==2) {a2++;}
        if (da==3) {a3++;}
        if (da==4) {a4++;}
        if (da==5) {a5++;}
        if (da==6) {a6++;}
        if ((a1>((2*q)/6.0)) || (a2>((2*q)/6.0)) || (a3>((2*q)/6.0)) ||
            (a4>((2*q)/6.0)) ||
            (a5>((2*q)/6.0)) ||
            (a6>((2*q)/6.0))) {
            cout<<"Dados viciado"<<endl;
        }
        else {
            cout<<"Dados regular"<<endl;
        }
    }
}
```

9. Escreva um programa que leia dois inteiros A e B e imprima a mensagem NULO, POSITIVO ou NEGATIVO conforme o resultado da multiplicação de A por B SEM REALIZAR A OPERAÇÃO.

Exemplo de execução:

```
10 12
POSITIVO
Outro exemplo:
16 -34
NEGATIVO
Outro exemplo:
0 -34
NULO
```

```
#include<iostream>
using namespace std;
int main() {
    int a,b;
    cin>>a>>b;
    if ((a>0) && (b>0)) || ((a<0) && (b<0))) {
        cout<<"Sinal positivo" <<endl;
    }
    else {
        if ((a==0) || (b==0)) {
            cout<<"Nulo"<<endl;
        }
        else {
            if ((a>0) && (b<0)) || ((a<0) && (b>0))) {
                cout<<"Sinal negativo"<<endl;
            }
        }
    }
}
```

10. Escreva um programa que leia um conjunto de pares de valores inteiros positivo, (p_1, n_1) , (p_2, n_2) , ..., (p_k, n_k) , onde p_i representa uma praça de pedágio saindo de Curitiba e n_i representa a quantidade de carros que passaram nas últimas 24 horas nessa praça de pedágio p_i . Foram avaliadas 5 praças. Considere que todos os valores são inteiros e positivos e que o valor p_i está entre 1 e 5, isto é $1 \leq p_i \leq 5$ e que o final do conjunto de valores é indicado quando for digitada a dupla 0,0. No final, o programa deverá imprimir:

- A quantidade total de veículos que passaram em cada praça de pedágio,
- A quantidade de carros que passaram por todas as praças e
- A quantidade média por praça.

Cada praça p_i pode receber vários lançamentos de n_i que deverão ser somados.

Exemplo de execução:

Informe praças e quantidades (0 0= fim):

```
1 1250
2 3020
4 3018
5 4109
3 3298
2 765
1 435
2 3278
0 0
Praca de pedagio 1: 1685 veiculos
Praca de pedagio 1: 1685 veiculos
Praca de pedagio 2: 7063 veiculos
Praca de pedagio 3: 3289 veiculos
Praca de pedagio 4: 3018 veiculos
Praca de pedagio 5: 4109 veiculos
Quantidade total: 19173 veiculos
Media por praça de pedagio: 3834.60 veiculos
```

```
#include<iostream>
using namespace std;
int main() {
    int p1,p2,p3,p4,p5;
    int ped,car,s;
    float med;
    p1=0; p2=0; p3=0; p4=0; p5=0; s=0;
    while (1==1) {
        cin>>ped>>car;
        if ((ped==0) && (car==0)){
            break;
        }
        if (ped==1) {p1=p1+car;}
        if (ped==2) {p2=p2+car;}
        if (ped==3) {p3=p3+car;}
        if (ped==4) {p4=p4+car;}
        if (ped==5) {p5=p5+car;}
        s=s+car;
        med=s/5.0;
        cout<<"Praca de pedagio 1: "<<p1<<" veiculos"<<endl;
        cout<<"Praca de pedagio 2: "<<p2<<" veiculos"<<endl;
        cout<<"Praca de pedagio 3: "<<p3<<" veiculos"<<endl;
        cout<<"Praca de pedagio 4: "<<p4<<" veiculos"<<endl;
        cout<<"Praca de pedagio 5: "<<p5<<" veiculos"<<endl;
        cout<<"Quantidade total: "<<s<<" veiculos"<<endl;
        cout<<"Media por praça de pedagio : "<<med<<" veiculos"<<endl;
    }
}
```

11. Você foi contratado por uma nova república que precisa um programa para realizar o ajuste anual de rendimentos pessoa física. O programa deve ler:

- RBA: renda bruta anual
- NPD: número de dependentes
- IPG: imposto pago no ano

A base de cálculo (BC) é dada por $RBA - NPD \times 1500,00$. O número máximo de dependentes dedutíveis é 4. Para o cálculo do imposto devido, você deve utilizar a tabela progressiva abaixo para definir o imposto devido (IBC) que é calculado aplicando a percentagem indicada (alíquota) sobre a Base de Cálculo:

BC	alíquota
até 15.000,00	0% (isento)
de 15.000,01 até 35.000,00	7.5%
de 35.000,01 até 280.000,00	15.0%
acima de 280.000,00	60%

Com estes dados, você deve calcular o imposto devido e descontar deste o imposto que já foi pago no ano, informando ao usuário:

- Valor final do imposto a pagar OU
- Valor final da restituição a receber OU
- Mensagem indicando que o usuário quitou o imposto

Exemplo de execução:

```
RBA: 55000.00
NPD: 3
IPG: 7230.0
Imposto a pagar: 345.00
```

Outro exemplo de execução:

```
RBA: 35000.00
NPD: 2
IPG: 10230.0
Restituicao a receber: 7830.00
```

```

#include<iostream>
using namespace std;
int main() {
float rba,ipg,bc,ali,ibc,dif;
int ndp;
cout<<"RBA: ";
cin>>rba;
cout<<"NDP: ";
cin>>ndp;
cout<<"IPG: ";
cin>>ipg;
if (ndp>4) {ndp=4;}
bc=rba-(ndp*1500.00);
if (bc<=15000.00) {ali=0.0;}
else {
if (bc<=35000.00) {ali=0.075;}
else {
if (bc<=280000.00) {ali=0.15;}
}}else {ali=0.6;}
ibc=bc*ali;
dif=ipg-ibc;
if (dif>0){cout<<"Restituicao a receber: R$ "<<dif<<endl;}
if (dif<0){cout<<"Imposto a pagar: R$ "<<(-dif)<<endl;}
if (dif==0){cout<<"Imposto quitado"<<endl;}
}
}

```

12. Escreva um programa que calcule e exiba os tempos gastos e alturas percorridas por uma lesma em suas escaladas em arranha céus. Para isso, o usuário deve informar a altura a ser escalada (A), a quantidade de metros que a lesma consegue subir por turno de 12 horas de subida (V_s) e a quantidade de metros que ela escorrega por turno de 12 horas de descanso (V_d). Os valores de entrada para A e V_s são positivos e não nulos. O valor de V_d está no intervalo $[0, V_s]$. Caso estas restrições não sejam satisfeitas, o programa deve terminar sem realizar nenhum cálculo exibindo mensagem que descreva o fato transgressor. A escalada deve ser descrita como nos exemplos a seguir:

Exemplo de execução:

```

Altura total a escalar (m): 33
Escalada de subida (m/12h): 10
Escorregamento de descanso (m/12h): 2
Desenvolvimento da escalada:
0h / 0m
12h / 10m
24h / 8m
36h / 18m
48h / 16m
60h / 26m
72h / 24m
82.8h / 33m <objetivo atingido>

```

Note que $82.8h = 33m - 24m = 9m / 10m = 0.9 * 12h = 10.8h + 72h = 82.8h$

Outro exemplo de execução:

```

Altura total a escalar (m): 5
Escalada de subida (m/12h): 10
Escorregamento de descanso (m/12h): 2
Desenvolvimento da escalada:
0h / 0m
6h / 5m <objetivo atingido>

```

Note que $6h = 5m - 0m = 5m / 10m = 0.5 * 12h = 6h + 0h = 6h$

```

#include<iostream>
using namespace std;
int main() {
int ate,sub,esd,alt,hora,i;
float fin;
cout<<"Altura total a escalar (m): ";
cin>>ate;
cout<<"Escalada de subida (m/12h): ";
cin>>sub;
cout<<"Escorregamento de descanso (m/12h): ";
cin>>esd;
cout<<"Desenvolvimento da escalada:"<<endl;
hora=0;
alt=0;
i=0;
cout<<" 0h / 0m"<<endl;
while (alt<ate){
hora=hora+12;
i++;
if (i%2==1){
}alt=alt+sub;
else {alt=alt-esd;}
if (alt<ate) {
}cout<<hora<<"h / "<<alt<<" m"<<endl;
}
fin=12*((ate-(alt-sub))/(sub*1.0));
cout<<(hora-12)+fin<<"h / "<<ate<<" m
<objetivo atingido>"<<endl;
}
}

```


Listagem de todos os problemas de 136g

1. Dada a atual e futura crise hídrica do país, as pessoas começaram a construir reservatórios para armazenar água em suas propriedades. Escreva um programa que auxilie os utilizadores do reservatório a controlarem seu consumo. Obtenha do teclado as dimensões de um reservatório cúbico (altura, largura e comprimento, em centímetros) e o consumo médio diário dos utilizadores (em litros/dia). Assuma que o reservatório esteja cheio e informe
 - (a) capacidade total do reservatório, em litros
 - (b) autonomia do reservatório, em dias
 - (c) classificação do consumo, de acordo com a autonomia em dias

elevado se a autonomia for menor que 2 dias
moderado se a autonomia estiver entre 2 e 7 dias
reduzido se a autonomia for maior que 7 dias.

Considere que cada litro equivale a 1000cm^3 ou 1dm^3 .

Exemplo de execução:

```
Altura do reservatorio (cm): 100
Largura do reservatorio (cm): 100
Comprimento do reservatorio (cm): 100
Consumo medio diario (litros/dia): 400
Capacidade do reservatorio: 1000 litros
Autonomia do reservatorio: 2.5 dias
Consumo moderado
```

Outro exemplo de execução:

```
Altura do reservatorio (cm): 50
Largura do reservatorio (cm): 100
Comprimento do reservatorio (cm): 150
Consumo medio diario (litros/dia): 400
Capacidade do reservatorio: 750 litros
Autonomia do reservatorio: 1.875 dias
Consumo elevado
```

```
#include<iostream>
using namespace std;
int main() {
    int a,l,c,cd;
    float aut;
    cout<<"Altura do reservatorio (cm): ";
    cin>>a;
    cout<<"Largura do reservatorio (cm): ";
    cin>>l;
    cout<<"Comprimento do reservatorio (cm): ";
    cin>>c;
    cout<<"Consumo medio diario (litros/dia): ";
    cin>>cd;
    cout<<"Capacidade Reservatorio: "<<(a*l*c)/1000.0<<" litros"<<endl;
    aut=(a*l*c)/(cd*1000.0);
    cout<<"Autonomia do reservatorio: "<<aut<<" dias"<<endl;
    if (aut<2){ cout<<"Consumo elevado"<<endl;}
    else {
        if (aut<=7.0){
            cout<<"Consumo moderado"<<endl;}
        else {
            cout<<"Consumo reduzido"<<endl;}
    }
}
```

2. Escreva um programa que leia do teclado uma sequência de números inteiros, até que seja lido um número que seja o dobro ou a metade do anterior. O programa deve imprimir como saída os seguintes valores
 - (a) A quantidade de números lidos
 - (b) A soma dos números lidos
 - (c) Os dois valores que forçaram a parada do programa

Exemplo de execução:

```
4 3 15 8 16
5 4 8 16
```

Outro exemplo:

```
-424 646 438 892 964 384 192
7 3092 384 192
```

```
#include<iostream>
using namespace std;
int main() {
    int atu,ant,q,s;
    cin>>atu;
    s=atu;
    q=1;
    while (1==1){
        ant=atu;
        cin>>atu;
        s=s+atu;
        q++;
    }if ((atu==ant*2) || (atu==ant/2)) {break;}
    cout<<q<<" "<<s<<" "<<ant<<" "<<atu<<endl;
}
```

3. Vamos chamar de **crescente** um número natural $n = d_1d_2\dots d_k$ cujos dígitos d_i estão em ordem crescente, isto é, $d_1 < d_2 < \dots < d_k$. Escreva um programa que leia um inteiro e positivo n de 3 dígitos e verifique se n é crescente. O programa deve também verificar se n tem os 3 dígitos determinados e imprimir mensagens adequadas em cada caso.

Exemplo de execução:

Entre com um inteiro positivo: 3416
Valor invalido

Mais um exemplo de execução:

Entre com um inteiro positivo: 152
152 nao e crescente

Mais um:

Entre com um inteiro positivo: 378
378 e crescente

```
#include<iostream>
using namespace std;
int main() {
    int a,d1,d2,d3;
    cin>>a;
    if (a>999) { cout<<"Valor invalido";}
    else{
        d1=a/100;
        d2=(a-(d1*100))/10;
        d3=a%10;
        if ((d3>d2) && (d2>d1)) {cout<<a<<" e crescente";}
        }else {cout<<a<<" nao e crescente";}
    }
}
```

4. O licenciamento anual de um veículo deve ser pago de acordo com o último dígito de sua placa nos meses previstos na tabela abaixo:

Final da placa	Mês de renovação
1	Abril
2	Maio
3	Junho
4	Julho
5 e 6	Agosto
7	Setembro
8	Outubro
9	Novembro
0	Dezembro

Uma locadora de veículos precisa saber quantos licenciamentos terá que pagar a cada mês, de acordo com as placas de sua frota. Escreva um programa que leia uma sequência de placas de automóveis (-1 indica o fim), cada placa formada apenas por 4 dígitos numéricos e determine

- (a) a quantidade de licenciamentos que deve ser paga a cada mês
- (b) a quantidade de carros da locadora nesse caso e
- (c) a média mensal de licenciamentos que deverão ser pagos

Exemplo de execução:

Informe as placas dos veiculos:
4444 2231 4451 3422 1230
3211 2376 6789 9820 3425 -1
Total de Abril: 3
Total de Maio: 1
Total de Junho: 0
Total de Julho: 1
Total de Agosto: 2
Total de Setembro: 0
Total de Outubro: 0
Total de Novembro: 1
Total de Dezembro: 2
Tamanho da frota: 10 veiculos
Media mensal de licenciamentos pagos: 1.1

```
#include<iostream>
using namespace std;
int main() {
    int placa,qabr,qmai,qjun,qjul,qago,qset,qout,qnov,qdez,qcar;
    float med;
    qabr=0; qmai=0; qjun=0; qjul=0; qago=0;
    qset=0; qout=0; qnov=0; qdez=0; qcar=0;
    while (1==1){
        cin>>placa;
        if (placa==-1) {break;}
        qcar++;
        if (placa%10==1) {qabr++;}
        if (placa%10==2) {qmai++;}
        if (placa%10==3) {qjun++;}
        if (placa%10==4) {qjul++;}
        if ((placa%10==5) || (placa%10==6)) {qago++;}
        if (placa%10==7) {qset++;}
        if (placa%10==8) {qout++;}
        if (placa%10==9) {qnov++;}
        }if (placa%10==0) {qdez++;}
        cout<<"Total de Abril: "<<qabr<<endl;
        cout<<"Total de Maio: "<<qmai<<endl;
        cout<<"Total de Junho "<<qjun<<endl;
        cout<<"Total de Julho: "<<qjul<<endl;
        cout<<"Total de Agosto: "<<qago<<endl;
        cout<<"Total de Setembro: "<<qset<<endl;
        cout<<"Total de Outubro: "<<qout<<endl;
        cout<<"Total de Novembro: "<<qnov<<endl;
        cout<<"Total de Dezembro: "<<qdez<<endl;
        cout<<"Tamanho da frota: "<<qcar<<" veiculos"<<endl;
        med=qcar/9.0;
        cout<<"Media mensal de licenciamentos pagos: "<<med<<" veiculos"<<endl;
    }
}
```

5. Escreva um programa que leia do teclado um número inteiro de 3 algarismos, construa e exiba outro número de 4 algarismos, de acordo com a seguinte regra:

- (a) os primeiros 3 algarismos contados da esquerda para a direita, são iguais ao número lido.
- (b) o quarto algarismo é um dígito de controle, calculado da seguinte forma: primeiro algarismo + segundo algarismo * 3 + o terceiro algarismo * 5; o dígito de controle será igual ao resto da divisão desta soma por 7, caso este resto seja ímpar. Caso contrário o dígito de controle deverá ser o resto da divisão obtido + 1.

Exemplo de execução:

No.: 123
No.obtido: 1231

Outro exemplo de execução:

No.: 1246
Entrada invalida

Outro exemplo de execução:

No.: 333
No. obtido: 3337

6. Após finalmente abrir a sua tão sonhada pizzeria, Giovanni decide fazer uma avaliação em tempo real, à medida em que os pedidos vão chegando, de todos os custos envolvidos com o queijo utilizado. Quatro tamanhos de pizzas são vendidos em sua pizzeria: P=pequeno, 6 fatias; M=médio, 8 fatias; G=grande, 10 fatias e F=família, 12 fatias. Sabendo que cada fatia da pizza utiliza uma fatia de queijo e que o queijo que Giovanni usa pesa em média 30 gramas/fatia, crie um programa que leia o custo do quilograma de queijo (R\$/kg) e logo após, leia uma sequência de tamanhos de pizza e mostre a cada nova pizza o total de queijo gasto até o momento (em kg) e qual o custo cumulativo de queijo (em reais). A sequência de pizzas é encerrada pelo tamanho=N. Ao final do programa mostre quantas fatias de queijo foram utilizadas no total e quanto tempo foi gasto no processo de fatiamento do queijo em minutos e segundos, considerando que Giovanni gasta 5 segundos para cortar 1 fatia de queijo.

Exemplo de execução:

Custo do queijo (R\$/kg): 10.5
Tamanho da pizza: P
Quantidade de queijo: 0.18 kg, custo = R\$ 1.89
Tamanho da pizza: M
Quantidade de queijo: 0.42kg, custo = R\$ 4.41
Tamanho da pizza: G
Quantidade de queijo: 0.72kg, custo = R\$ 7.56
Tamanho da pizza: F
Quantidade de queijo: 1.08kg, custo = R\$ 11.34
Tamanho da pizza: m
Tamanho invalido !
Quantidade de queijo: 1.08kg, custo = R\$ 11.34
Tamanho da pizza: M
Quantidade de queijo: 1.32kg, custo = R\$ 13.86
Tamanho da pizza: N
Fatias: 44
Tempo do fatiamento: 3 min 40 seg

7. Seu patrão comprou um barco novo para a empresa de transportes e agora precisa saber se o barco novo conseguirá atender aos pedidos dentro dos prazos contratados pelos clientes. O barco possui a capacidade de transportar até 1 tonelada, com velocidade de navegação que se reduz gradualmente dependendo da carga, de 50 km/hora (vazio) até 30 km/hora (lotado). Escreva um programa que leia o peso da carga a ser transportado (em kg), a distância a ser navegada (em km) e o tempo limite da entrega (em horas) e imprima se a carga chegará pontualmente ou se estará atrasada (ou adiantada) e o número de horas e minutos de atraso ou adiantamento. Lembre-se de verificar se as entradas estão dentro da faixa de valores válidos.

Exemplo de execução:

Peso a transportar (kg): 1000
Distancia a navegar (km): 30
Prazo de entrega (h): 2
Entrega com adiantamento de 1 h 0 min

Outro exemplo de execução:

Peso a transportar (kg): 500
Distancia a navegar (km): 450
Prazo de entrega (h): 10
Entrega com atraso de 1 h 15 min

Outro exemplo de execução:

Peso a transportar (kg): 250
Distancia a navegar (km): 900
Prazo de entrega (h): 20
Entrega pontual

Outro exemplo de execução:

Peso a transportar (kg): 5100
Peso invalido

8. A Secretaria do Meio Ambiente que controla o índice de poluição ambiental, mantém 3 grupos de indústrias que são altamente poluentes. O índice de poluição aceitável varia de 0.05 até 0.25. Se o índice passa de 0.25, as indústrias do grupo 1 são intimadas (a suspender a produção). Se o índice chega a 0.4, as indústrias dos grupos 1 e 2 são intimadas. Se o índice chega a 0.5 os 3 grupos são intimados, e se o índice passa de 0.5 a região deve ser evacuada. Escreva um programa que leia o índice e emita o aviso correspondente.

Exemplo de execução:

Indice de poluicao: 0.4
Grupos 1 e 2 intimados

Outro exemplo de execução:

Indice de poluicao: 0.27
Grupos 1 intimado

```

#include<iostream>
using namespace std;
int main() {
float ip;
cout<<"Informe o IP ";
cin>>ip;
cout<<(ip>0.4);
cout<<(0.4>0.4);
if (ip>0.5){
}cout<<"Evacuar a regioa "<<endl;
if ((ip<=0.5) && (ip>0.4)){
}cout<<"Parar todas industrias "<<endl;
if ((ip<=0.4) && (ip>0.3)){
}cout<<"Parar grupo 1 e 2 "<<endl;
if ((ip<=0.3) && (ip>0.25)){
}cout<<"Parar grupo 1 "<<endl;
}

```

9. Todo final do mês você prepara o orçamento do mês seguinte que está para começar. A partir do valor do seu salário (SAL) e as despesas do mês corrente: ALIM, alimentação; ESC, escola dos filhos; CC, cartão de crédito e PEX, pensão do ex-cônjuge, você deve tentar prever se vai conseguir pagar tudo no próximo mês. Para isto, escreva um programa que peça aos usuários os valores reais de SAL, ALIM, ESC, CC e PEX e faça uma análise mostrando quais despesas poderão ser pagas no mês. A despesa precisa ser paga integralmente, na seguinte ordem: escola, alimentação, cartão de crédito e pensão. Ao final, o programa deve mostrar quanto sobrou do salário após descontar as despesas efetivamente pagas.

Exemplo de execução:

```

Salario R$: 5675.00
Escolas R$: 2199.45
Alimentação R$: 834.87
Cartao de credito R$: 1523.37
Pensao ex-conjuge R$: 1749.72
Voce consegue pagar escolas
Voce consegue pagar alimentacao
Voce consegue pagar cartao
Voce nao consegue pagar pensao
Saldo do salário: R$ 1117.31

```

Outro exemplo:

```

Salario R$: 2100.00
Escolas R$: 2199.45
Alimentação R$: 834.87
Cartao de credito R$: 1523.37
Pensao ex-conjuge R$: 749.72
Voce nao consegue pagar escolas
Voce consegue pagar alimentacao
Voce nao consegue pagar cartao
Voce consegue pagar pensao
Saldo do salário: R$ 515.41

```

10. Escreva um programa que leia um número inteiro qualquer, determine e mostre quantos dígitos o formam; A solução obrigatoriamente deverá usar while().

Exemplos de execução:

```

N.: 339
3 digitos

N.: 0
1 digitos

N.: -5000000
7 digitos

```

```

#include<iostream>
#include<cmath>
using namespace std;
int main() {
int q,d,a;
q=0;
cout << "Informe o numero " ;
cin>>d;
if (d==0){
}q=1;
while (d!=0) {
q=q+1;
}d=d/10;
cout << q;
}

```

11. Escreva um programa que leia um par de números inteiros A e B . Se A for maior que B , imprimir a sequência $B, B + 1, B + 2, \dots, A - 1, A$. Caso contrário, imprima a sequência $A, A + 1, A + 2, \dots, B - 1, B$. Os números nunca serão iguais. Ao final, imprima a soma de todos os termos da sequência.

Exemplo de execução:

```

Indique par: 4 6
Sequencia: 4 5 6
Soma: 15

```

Outro exemplo:

```

Indique par: -2 1
Sequencia: -2 -1 0 1
Soma: -2

```

Mais uma:
Indique par: 5 -3
Sequencia: -3 -2 -1 0 1 2 3 4 5
Soma: 9

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main() {
    int a,b,q;
    q=0;
    cout<<"Informe a e b";
    cin>>a>>b;
    if (a>b) {
        while (b<=a) {
            cout << b << " " ;
            q=q+b;
        }b=b+1;
    } else {
        while (a<=b) {
            cout << a << " " ;
            q=q+a;
        }a=a+1;
    }
    cout << "total " << q << endl;
}
```

12. Escreva um programa que leia um inteiro positivo qualquer, calcule e imprima a soma de seus dígitos.

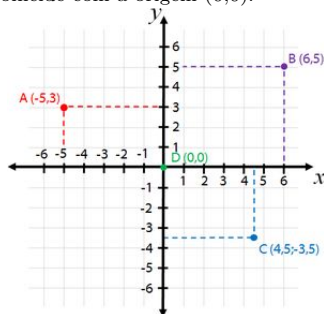
Exemplo de execução:
Digite um inteiro positivo: 1234
Soma dos digitos: 10
Mais um:
Digite um inteiro positivo: 227634
Soma dos digitos: 24

Prática em eixos cartesianos

Suponha um ambiente representado em 2 dimensões. Pode ser um mapa de estradas, uma planta de apartamento, um mapa de caça ao tesouro, um diagrama de linhas do metro ou qualquer coisa semelhante.

Uma maneira prática de localizar pontos em tais diagramas é pelo uso de eixos cartesianos. Trata-se de duas retas orientadas, perpendiculares entre si, que se cruzam em um ponto qualquer denominado origem. As retas são numeradas a partir da origem com números positivos em um sentido (cima e direita) e com números negativos no outro sentido (baixo e esquerda).

A partir daí, qualquer ponto pode ser localizado, bastando traçar sobre o ponto paralelas aos dois eixos e anotando em que lugar dos eixos tais paralelas os cruzam. Veja por exemplo na figura abaixo, 4 pontos assinalados: o ponto A tem como coordenadas os números (-5,3). O ponto B tem coordenadas (6,5), o C (4,5,-3,5) e finalmente o ponto D coincide com a origem (0,0).



O eixo horizontal é chamado eixos das abcissas, ou eixo x e sempre corresponde ao primeiro número do par. O eixo vertical é chamado eixo das ordenadas, ou eixo y e corresponde ao segundo número do par. Note que os números podem ser positivos, negativos, inteiros ou não.

O eixo é chamado cartesiano em homenagem a René Descartes, um filósofo-matemático francês do século XVII. Entre outras criações ele inventou a geometria analítica que vem a ser o casamento entre álgebra e a geometria e que representa suas maravilhas sobre o palco dos eixos cartesianos. Conta a lenda que ele teria tido o *insight* da geometria analítica ao observar, deitado numa cama, o teto do aposento no qual uma mosca andava. Sua mente matemática via a mosca mas enxergava um ponto se movendo. A pergunta: haverá uma função que descreva o caminho da mosca? deu origem a esta fantástica construção.

Modernamente, graças à proliferação de dispositivos de geo-localização baratos (GPS), tem tido crescimento exponencial os sistemas que fazem georreferenciamento de pontos. Explica-se: qualquer coisa em nosso planeta tem uma localização fixa em relação a dois eixos: a linha do equador e o meridiano de Greenwich. No caso da Terra, tais medidas são chamadas latitude (N=acima do equador e S=abaixo) e longitude W (à esquerda do meridiano de Greenwich – para quem olha o mapa, e L à direita). As medidas são aquelas da circunferência: graus, minutos e segundos.

Por exemplo, Curitiba se localiza a 25° 26' S e 49° 16' W. Em alguns contextos o Sul é representado por um número negativo e o Oeste por um positivo, ficando -25° 26' e +49° 16'. As medidas podem ser precisas, chegando a uma resolução de centímetros na superfície terrestre. No futuro, todos ou quase todos os sistemas de informação vão agregar o geo-referenciamento de dados, razão pela qual vale a pena estudar o sistema e praticar nele.

Prática

A seguir uma lista de 10 exercícios envolvendo questões relativas aos eixos cartesianos.

- ★ **1. Área de retângulo:** Suponha que um retângulo é dado por 4 números: Os dois primeiros são as coordenadas (X, Y) do canto superior

esquerdo e os dois últimos são as coordenadas do canto inferior direito. Sabe-se que os lados do retângulo sempre serão paralelos aos eixos. Pergunta-se a seguir qual a área do retângulo. Relembrando, a área de um retângulo de lados a e b é o produto $a \times b$. Logo, o primeiro a descobrir é o comprimento de cada aresta. Assim, o retângulo 2,5,5,3 tem arestas de $5 - 3 = 2$ e de $5 - 2 = 3$. Logo a área é $2 \times 3 = 6$ ou área de 6 u.a. (unidades de área).

- ★ **2. Área de triângulo:** Suponha que um triângulo é dado por 3 pares de números. Cada par representa um dos pontos do triângulo. Neste problema, uma das arestas do triângulo sempre vai ser paralela a um dos eixos. Isso vai facilitar as coisas. A área de um triângulo é $T = \frac{b \times h}{2}$. Seja o exemplo: achar a área do triângulo 2,5; 2,-1; 8,1. Desenhando-o descobre-se que a base pode ser a aresta 2,5 a 2,-1, já que ela é paralela ao eixo y e isto facilita o cálculo da altura que é $8 - 2 = 6$. Assim, a área é $(6 \times 6) \div 2 = 18$ u.a.

- ★ **3. Área de um círculo:** Suponha que um círculo é dado por 2 pontos: seu centro e um ponto qualquer situado na circunferência do círculo. Voltando à geometria analítica, uma circunferência é dada por uma aplicação imediata do Teorema de Pitágoras, ou $(x - \alpha)^2 + (y - \beta)^2 = R^2$, onde (α, β) são as coordenadas do centro e R é o raio da circunferência. Neste caso, o que nos interessa é a fórmula da área de um círculo que é $C = \pi \times R^2$. Para o cálculo do raio, basta achar a distância entre o centro (dado) e um ponto qualquer (também dado). Usa-se o Teorema de Pitágoras, sempre ele. Eis um exemplo: seja o círculo dado por 0, -3, -6, -5; onde o centro é 0, -3 e um ponto qualquer da circunferência é -6, -5. A área do círculo é 125 u.a.

- ★ **4. Pontos internos:** Neste exercício, você vai receber 10 pontos (pares de números). deve desenhar a envoltória convexa dos pontos e descobrir quantos pontos estão DENTRO dessa envoltória. Pense numa tábua. Desenhe os eixos nessa tábua e marque os 10 pontos. Pregue um prego em cada ponto. Passe um elástico pelos pontos mais externos. O elástico é a envoltória convexa. Os pontos que NÃO encostam no elástico são os pontos internos, que podem ser no máximo 7 (porquê?). Seja o exemplo: 1 -8, -5 -9, -8 7, 8 -4, 7 -4, -7 -9, 9 -4, 4 -2, 1 -9 e 1 7. Neste caso há 4 pontos internos à envoltória.

- ★ **5. O triângulo contém a origem?:** Agora você vai receber um triângulo (3 pares de números) e deve dizer 1 para SIM ou 0 para NÃO se o triângulo contiver ou não a origem dos eixos. Em outras palavras, se o ponto (0,0) estiver dentro do triângulo, responda 1. Se o ponto (0,0) estiver fora do triângulo, responda 0. Por exemplo, seja o triângulo -6 6, -1 1 e -5 2. Tal triângulo não engloba a origem, logo a resposta é 0.

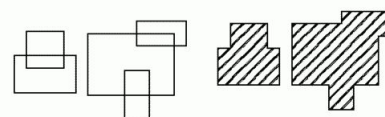
- ★ **6. Pontos por quadrante:** As regiões do plano delimitada pelos eixos são chamadas quadrantes. Há o primeiro, acima do eixo x e à direita do eixo y , o segundo, (acima do x e à esquerda do y), o terceiro (abaixo de x e à esquerda de y) e finalmente o quarto quadrante que fica abaixo do x e à direita do y . Neste exercício você vai receber 10 pontos e deve dizer quantos estão em cada um dos 4 quadrantes, somando esse resultado depois. Note que se um ponto está sobre um eixo, ele pertence aos dois quadrantes que são definidos pelo eixo em questão. Na situação limite, o ponto (0,0) pertence aos quatro quadrantes. Seja por exemplo, -3 -7, -4 -4, 0 -4, -6 8, 6 -7, -9 -3, 5 9, 0 0, 7 2 e -7 7. A resposta é 3,3,5,3 ou a sua soma 14.

- ★ **7. Qual o ponto mais distante da origem?:** Neste exercício, você vai receber 5 pontos (pares) e deve dizer qual dos 5 pontos é o mais distante da origem. A solução está no Teorema de Pitágoras aplicado a cada ponto. Note que a distância em questão é a hipotenusa de um triângulo cujos lados estão sobre os eixos x e y . Por exemplo, 5 -4, 1 2, 9 -7, -5 -9 e -3 8. Neste caso, o ponto mais distante é o 3 (ou seja 9 -7).

- ★ **8. Qual o ponto mais próximo ao mestre?:** Dado um ponto mestre e 5 pontos acessórios, a pergunta agora é qual o ponto acessório que

está mais perto do ponto mestre. É uma generalização do caso anterior (lá o mestre coincidia com a origem). O raciocínio é o mesmo, mas os lados do triângulo são dados pela subtração das coordenadas do acessório e do mestre, antes de usar o Teorema de Pitágoras. Seja o exemplo: Ponto mestre é -1 -9. Os pontos acessórios são 4 1, -6 -5, -4 8, 0 -7 e 7 5. O mais próximo é o ponto acessório 4 (ou 0 -7).

- ★ **9. Área coberta pela fotogrametria:** Uma cidade contratou um serviço de aerofotogrametria. Inúmeras fotos foram tiradas de avião. As fotos sempre são retangulares, mas devido a diferentes altitudes em que foram tomadas têm tamanhos diferentes entre si. No exercício aparecem 5 fotos cuja área é descrita por 2 pontos (X_1, Y_1, X_2, Y_2) , sendo o primeiro par (X_1, Y_1) o canto superior esquerdo e o segundo par (X_2, Y_2) o canto inferior direito da foto. O problema é que algumas áreas da cidade aparecem em mais de uma foto e portanto precisam ser descontadas da área final. veja um caso típico em que isto acontece:



Conjunto de cinco fotografias

Área Total

Seja o exemplo. Fotos: 20 40 32 26, 3 24 9 6 e 21 37 37 27. A área coberta é de 326 u.a.

- ★ **10. Com qual namorada?:** Patricio mora na cidade de O de Cima ($x=108, y=593$ no mapa). Tem 4 namoradas: Gloria que mora em Trobóbo (577,139); Lu que mora em Gloriba (283,532), Cristalda que mora em Cá de Judas (105,318) e Patrícia que mora a 431 u.d. de Patricio. A pergunta é: qual a namorada que mora mais perto dele? A resposta é a Lu.

Para você fazer

- Qual a área do retângulo -9, -6, -9, -9.
- Qual a área do triângulo -4, -7; 3, -1; -4, -1.
- Qual a área do círculo $C=1, 4P=-2, -1$.
- Quantos pontos internos à envoltória do conjunto -3, 5; 7, 4;-3, 6;-9, 0; 8, 5; 5,-7; 6,-3;-2, 2; 9,-7, 3.
- O triângulo abaixo contém ($R=1$) ou não contém ($R=0$) a origem? -8, 2; -2, 0; 6, -9.
- Ache quantos pontos há em cada quadrante. Depois some este número de pontos. 5,-9; 7, 1; 2, 7;-2, 6; 1, 2;-9, 1; 6,-4;-6,-1; 0, 0;-4,-3.
- Qual o número (1 a 5) do ponto mais distante da origem? 7,-1;-8,-3;-4,-9; 7, 3;-2, 2.
- Se o ponto mestre é -1, -9, qual o número (1 a 5) do ponto acessório mais próximo ao ponto mestre? 2, 0; 7, 4; 8, 1;-8,-4;-3,-7.
- Dadas 3 fotografias, qual a área total coberta pelas fotos? 21 35 31 27; 24 30 30 25; 20 48 34 26.
- Qual o ser amado mais próximo? Armindo mora em Ufala ($X=263, Y=237$). Ele tem 4 namoradas: Flavinha mora em Gloriba ($X=168, Y=258$), Cristalda mora em Rinoska ($X=363, Y=398$), Arminda mora em Trobobo ($X=494, Y=575$) e Veruska mora a 558 quilômetros da cidade dele. Qual o nome da namorada que mora mais perto dele?

1	2	3	4	5
6	7	8	9	10



- 1 - /

Numeração romana

O sistema de numeração romana desenvolveu-se na antiga Roma e utilizou-se em todo o seu império. Neste sistema as cifras escrevem-se com determinadas letras, que representam os números. As letras são sempre maiúsculas, já que no alfabeto romano não existem as minúsculas.

As equivalências dos numerais romanos com o sistema decimal são as seguintes:

Decimal	Romana	Decimal	Romana	Decimal	Romana
1	I	10	X	100	C
5	V	50	L	500	D
1000	M				

No sistema de numeração romano as letras devem situar-se da ordem de maior valor para a de menor valor. Não se devem escrever mais de três I, ou três X, ou três C em qualquer número. Se estas letras se situam à frente de um V, um L, ou um D, respectivamente, subtrai-se o seu valor à cifra das ditas letras.

Os romanos desconheciam o zero, introduzido posteriormente pelos árabes, de forma que não existe nenhuma forma de representação deste valor.

Para cifras elevadas os romanos utilizavam um hífen colocado por cima da letra correspondente. O hífen multiplicava o valor da letra por 1.000. Por exemplo, um "C" com hífen superior correspondia ao valor 100.000 (100×1.000), e um "M" com hífen superior correspondia ao valor 1.000.000 (1.000×1.000). Este método permitia escrever cifras realmente altas.

Apresentam-se vários exemplos de números romanos, com as suas equivalências decimais:

Decimal	Romana	Decimal	Romana	Decimal	Romana
1	I	2	II	3	III
4	IV	5	V	6	VI
7	VII	8	VIII	9	IX
10	X	104	CIV	1444	MCDXLIV

Deve-se notar que cada numeral romano básico é superior a todos os numerais romanos básicos de menor valor e cada numeral romano básico é subordinado a todos os numerais romanos básicos de maior valor.

Uma característica importante é que os números romanos não são únicos. Por exemplo, podemos considerar corretas as seguintes representações de 499: CDXCIX, LDVLIV, XDIX, VDIV e finalmente ID.

Conversão de romano para arábico

```
1: função R2A (cadeia NROM[n]) : inteiro
2: cadeia CADAUX [7] ← 'IVXLCDM'
3: inteiro VALAUX[7] ← 1 5 10 50 100 500 1000
4: inteiro SUM, I, K, TAM
5: TAM ← tamanho da cadeia NROM
6: SUM ← 0
7: I ← 1
8: enquanto I ≤ TAM faça
9:   K ← 1
10:  enquanto CADAUX[K] ≠ NROM[I] faça
11:    K++
12:  fim enquanto
13:  se (I+1) ≤ TAM então
14:    M ← 1
15:    enquanto CADAUX[M] ≠ NROM[I+1] faça
16:      M++
17:    fim enquanto
18:    se VALAUX[K] < VALAUX[M] então {ou K < M, tanto faz}
19:      SUM ← SUM + VALAUX[M] - VALAUX[K]
20:      I ← I + 2
21:    senão
22:      SUM ← SUM + VALAUX[K]
23:      I ← I + 1
24:    fim se
25:  senão
26:    SUM ← SUM + VALAUX[K]
27:    I ← I + 1
28:  fim se
29: fim enquanto
30: devolva SUM
31: fimfunção
```

Conversão de arábico para romano

```
1: função A2R (inteiro V) : (cadeia RESP[10])
2: inteiro D
3: cadeia AUX[10]
4: AUX ← "
5: E3 ← ⌊ V ÷ 1000
6: D3 ← V - (E3 × 1000)
7: E2 ← ⌊ D3 ÷ 100
8: D2 ← D3 - (E2 × 100)
9: E1 ← ⌊ D2 ÷ 10
10: D1 ← D2 - (E1 × 10)
11: AUX ← AUX + MUITM (E3)
12: AUX ← AUX + ARD (E2 , 'CDM')
13: AUX ← AUX + ARD (E1 , 'XLC')
```

```
14: AUX ← AUX + ARD (D1 , 'IVX')
15: devolva AUX
16: fimfunção
1: função ARD (inteiro A, cadeia LETRAS[3]) : (cadeia RESP[10])
2: cadeia AUX[10]
3: inteiro K
4: se A=0 então
5:   devolva "
6: senão se A=1 então
7:   devolva LETRAS[1]
8: senão se A=2 então
9:   devolva LETRAS[1]+LETRAS[1]
10: senão se A=3 então
11:   devolva LETRAS[1]+LETRAS[1]+LETRAS[1]
12: senão se A=4 então
13:   devolva LETRAS[1]+LETRAS[2]
14: senão se A=5 então
15:   devolva LETRAS[2]
16: senão se A=6 então
17:   devolva LETRAS[2]+LETRAS[1]
18: senão se A=7 então
19:   devolva LETRAS[2]+LETRAS[1]+LETRAS[1]
20: senão se A=8 então
21:   devolva LETRAS[2]+LETRAS[1]+LETRAS[1]+LETRAS[1]
22: senão se A=9 então
23:   devolva LETRAS[1]+LETRAS[3]
24: fim se
25: fimfunção
1: função MUITM (inteiro A) : cadeia RESP[10]
2: K ← 1
3: enquanto K ≤ A faça
4:   AUX[K] ← 'M'
5:   K++
6: fim enquanto
7: devolva AUX
8: fimfunção
Obs: note que + para números é adição (1 + 3 = 4) enquanto + para alfanuméricos (caracteres) significa concatenação ('A'+ 'B'='AB').
```

Exercícios de aquecimento

	arábico		romano
XVI		38	
XXXVII		44	
XLVI		35	
XCVI		104	

Para você fazer

Implemente os programas acima e a seguir responda:

Converte para arábico

MMMDCXXXIV	
DXLIV	
CLIX	
CDXXV	
MMMCXXIV	
DCII	

Converte para romano

1840	
175	
775	
609	
550	
1864	



- 1 - /

Completando algoritmos

1. **X é primo ?** Muitas vezes, é necessário saber dentro de um programa se um determinado inteiro é ou não é primo (Definição: um número inteiro positivo maior do que 1 é primo se e somente se seus únicos 2 divisores são a unidade e ele mesmo).

Há um imenso capítulo na matemática para descobrir a primalidade de um número sem ser necessário procurar seus divisores, já que este procedimento é muito demorado. Como não existe algo grátis, estes métodos (muitos baseados no Pequeno Teorema de Fermat), não são exatos, podendo apresentar falsos positivos (dizer que um *x* é primo, quando ele não o é), mas não falsos negativos (quando ele diz que é composto, *x* é composto!). A solução é chamar muitas vezes o mesmo método para o mesmo número (variando certas condições). A cada resposta positiva, a chance do erro cometido é dividida por 2. Daí poder-se ir tão longe quanto se queira.

Voltando à primalidade, se os números forem pequenos, a busca pode ser feita pelos métodos baseados no crivo de Eratóstenes, que seguem uma otimização crescente:

- 1. Variar um divisor entre 2 e *n* – 1. Examinar o resto em divisões sucessivas. Se alguma divisão deixar resto zero, *x* é composto e sair. Ao final, concluir que *x* é primo.
- 2. Lembrando que o único par primo é o 2, e excluindo-o da busca, concluir que se *n* é par, *n* é composto.
- 3. Outra melhoria é buscar a divisibilidade por 3, e neste caso, concluir que *n* é composto.
- 4. Uma melhoria (a principal) é que como não se quer todos os divisores, (apenas um resolve o problema!), a busca pode se encerrar em $\sqrt{n}$ sem ser necessário chegar até *n*. Isto diminui sobremaneira a carga computacional do algoritmo.
- 5. Finalmente, os primos obedecem à regra $p = (6 \times x) \pm 1$, pelo que o passo na busca pode ser de 6 em 6. (Se não acreditar nesta regra, examine todos os primos entre 5 e 100...)

Assim, por exemplo, 17 é primo, pois os únicos números inteiros que o dividem sem deixar resto são o 1 e o 17. Já 12 não é primo, pois seus divisores são o 1, 2, 3, 4, 6 e 12. A seguir um algoritmo (incompleto) escrito em Python. Você deve completá-lo, testá-lo, implementá-lo e finalmente achar as respostas pedidas.

```
def primo(n):
    if n==1:
        return False
    if n<4:
        -----
    if n%2==0:
        return False
    if n<9:
        -----
    if n%3==0:
        return False
    r=math.floor(math.sqrt(n))
    -----
    while f<=r:
        if (n%f)==0 or (n%(f+2))==0:
            return False
        f=f+6
    return True
```

A seguir, algumas execuções corretas desta função:

primo(123456789012419) é True
primo(1) é False
primo(2) é True

2. **A soma dos divisores de *n*** Defina-se a função *sigma*(*n*) como sendo a soma dos divisores próprios de *n* (ou seja excluindo-se dessa soma o próprio *n*). Assim, como os divisores de 12 são 1,2,3,4,6 e 12, *sigma*(12) = 1+2+3+4+6 = 16. O algoritmo incompleto em Python é

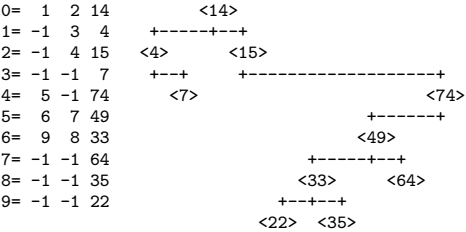
```
def sigma(n):
    -----
    i=2
    while i<n:
        if n%i==0:
            -----
            i=i+1
    -----
    sigma(12)=16
    sigma(123456) = 203696
    sigma(1000000) = 1480437
```

Execuções corretas:

3. **Criação de uma ABP** Uma Árvore Binária de Pesquisa (ABP) é uma interessante estrutura de dados que permite efetuar busca binária em um conjunto de dados SEM que eles precisem estar (ou ser mantidos) em ordem crescente. Nesta implementação não se usarão ponteiros e sim cursores. Conceitualmente é a mesma coisa, e ao custo de uma pequena ineficiência, ganha-se simplicidade, visibilidade, facilidade de depuração e sobretudo tempo de programação. Em uma ABP, a raiz é uma das chaves de busca e ela divide o universo em 2 pedaços: itens menores que a chave estarão vinculados ao filho esquerdo e itens maiores ao filho direito. Acompanhe no exemplo:

Seja incluir os números:
14 4 15 7 74 49 33 64 35 22

Eis como ficará esta ABP:



Depois que a árvore está criada, para verificar a correção da mesma, vis-a-vis o gabarito, ela deve ser percorrida em largura. No exemplo, daria como resposta: 14, 4, 15, 7, 74, 49, 33, 64, 22 e 35. O algoritmo python é

```
import numpy as np
def crarvore(x):
    arv=np.zeros((len(x),3),int)
    arv[0]=[-1,-1,x[0]]
    i=1
    ult=0
    while i<len(x):
        ult=ult+1
        arv[ult]=[-1,-1,x[i]]
        onde=0
        while onde!=-1:
            -----
            if x[i]>arv[onde,2]:
                esqdir=1
            -----
            else:
                esqdir=0
            -----
            arv[pai,esqdir]=ult
            i=i+1
        return arv
```

Para obter a visitação em largura basta percorrer o desenho da árvore em fatias horizontais. Mais alguns exemplos:

15 68 79 14 74 67 43 87 39 96 32 92 27 76,
deu como décimo = 76
76 8 3 80 58 24 32 13 19 36 51 47 38,
deu como décimo = 36
50 66 76 9 81 8 93 67 39 35 31,
deu como décimo = 31

4. **Egípcios** Os egípcios antigos, há mais de 5.000 anos, já conheciam o conceito de π – não com esse nome, claro, mas como a constante que iguala o comprimento de uma circunferência ao seu raio. Como eles não dominavam a aritmética real que temos hoje, π era representado por uma fração ordinária, nomeadamente 22/7. Seu problema aqui é receber um número real qualquer com bastantes casas decimais e localizar qual a fração de números inteiros que chega mais perto do valor do número real original. Ambos - numerador e denominador - devem ser menores do que 100 (ou seja, com 2 dígitos). Eis o código Python

```
def egipcio(nr,lim): # lim deve ser 100
    minimo=999999
    i=1
    while i<lim:
        -----
        while j<lim:
            ca=i/j
            if (abs(nr-ca))<minimo:
                -----
            mini=i
            minj=j
        -----
        i=i+1
    return [mini,minj]
```

Eis alguns exemplos:

9.333090, teve como fração mnemônica 28 / 3
1.23456, deu 79 / 64
6.10987, deu 55 / 9

📖 Para você fazer

A seguir, duas instâncias de cada exercício. Você deve executar o algoritmo sobre estes valores e achar a resposta pedida:

1. **X é primo ?** Responda 1 se *X* for primo e responda 0 senão.

1972131
1924619

2. **Soma divisores** Ache a soma dos divisores de

1971767
1474319

3. **ABP** Responda qual o décimo elemento na visitação em largura da árvore resultado da criação com estes números na ordem em que eles aparecem

33 26 31 12 4 50 3 37 97 90 87 36 51 8
85 52 29 35 14 3 79 93 40 63 37 13 32

4. **Egípcios** Ache o numerador e denominador inteiros, menores que 100 que mais se aproximam do número dado.

2.6333333333333333
4.2777777777777778

1.1	1.2	2.1	2.2
3.1	3.2	4.1	4.2



Comprimindo objetos binários com Huffman

O algoritmo de Huffman foi proposto em 1952, (*A Method for the Construction of Minimum-Redundancy Codes*) e é um marco na compressão de dados. Necessário não apenas pela quantidade imensa de dados digitais gerados pela humanidade, como pelo fato de que eles tendem a ser muito redundantes (música, vídeo, imagens, mesmo texto). Veja uma conta simples: Um vídeo HDTV tem 1920×1080 pixels, e uma nova imagem é gerada a cada $1/30$ de segundo e cada pixel ocupa 3 bytes (*RGB*), ou seja, sem compressão, um vídeo HDTV ocupa 186.624.000 bytes/segundo. 1 hora, 671.846.400.000 bytes, ou o conteúdo de 133 DVDs. Que você possa ver um filme de mais de 1 hora em um único DVD deve-se à compressão.

Dados não comprimidos usam uma tabela de possíveis valores existentes no material e atribuem a cada valor uma certa configuração de bits. Por exemplo, no código ASCII, a letra *A* maiúscula tem configuração 0100.0001. A letra *Z* é 0101.1010. Ambas ocupam 8 bits. O que Huffman intuiu é que a letra *A* é muito mais frequente do que a letra *Z*, e que, se ela tivesse um comprimento menor, haveria um ganho considerável de espaço. Vale aqui a regra de Pareto: em qualquer objeto binário poucos valores ocupam a maior parte das ocorrências. Isto caracteriza a redundância.

Como não existe almoço grátis, o que se ganha em espaço se perde em facilidade: agora não existe um comprimento padrão em bits (8 no caso do código ASCII, mais de 20 no caso de Unicode), e cada caractere tem um comprimento diferente. Os mais frequentes são mais curtos que os menos frequentes. O que Huffman fez foi bolar um algoritmo que trata este fato e permite facilmente comprimir e descomprimir objetos digitais.

A idéia de Huffman é:

1. Contar os caracteres e ordená-los
2. Construir uma árvore binária
3. Construir um dicionário a partir da árvore
4. Percorrer a árvore (ou consultar o dicionário) na compressão
5. Percorrê-la em sentido inverso na descompressão

1. Contagem

Eis o algoritmo

```
def huf2(uni):
    ctd=[]
    for i in range(len(uni)):
        j=0
        while uni[i]!=ctd[j][0]:
            ctd[j][1]=ctd[j][1]+1
            j+=1
        if j>len(ctd):
            ctd.append([uni[i],1])
        cts=sorted(ctd,key=getkey)
    return cts
```

2. Montagem da rvore

Eis o algoritmo

```
def ca1(x):
    for j in range(len(x)):
        x[j][0]=ord(x[j][0])
        arv=[]
        i=1
        while x[j][0]>=x[j][1]:
            x=sorted(x,key=getkey)
            w=x[0][1]+x[1][1]
            alfa=x[0][0]
            if alfa>0:
                alfa=alfa-1
            beta=x[1][0]
            if beta>0:
                beta=beta-1
            arv.append([i-1,alfa,beta])
            x.pop(0)
            x.pop(0)
            i=i+1
    return arv
```

3. Montagem do dicionário

Com a árvore acima, monta-se o dicionário

```
global dicionario
dicionario=[]
def cdic1(len(bb)-1,'',bb) # bb é a árvore
    ...
def cdic1(raiz,ate aqui,arv):
    if arv[raiz][1]<0:
        dicionario.append([chr(-arv[raiz][1]),xica])
    else:
        cdic1(arv[raiz][1],ate aqui+'0',arv)
    if arv[raiz][2]<0:
        xica=ate aqui+'1'
        dicionario.append([chr(-arv[raiz][2]),xica])
    else:
        ...
```

4. Compressão

Eis o algoritmo

```
def comprime1(texto,dicionario):
    res=''
    for i in range(len(texto)):
        for j in range(len(dicionario)):
            res=res+dicionario[j][1]
    return res
```

5. Descompressão

```
def descomprime(tc,arvore):
    td=''
    raiz=len(arvore)-1
    while i<len(tc):
        if tc[i]=='0':
            raiz=arvore[raiz][1]
            if raiz<0:
                td=td+chr(-raiz)
                raiz=len(arvore)-1
            else:
                if raiz<0:
                    td=td+chr(-raiz)
                    raiz=len(arvore)-1
                i=i+1
    return(td)
```

6. Para gravar

```
def gravac(texto,onde):
    f=open(onde,"wb") ; tt=''
    tamu=8*(math.ceil(tamr/8))
    a1=tamr%256
    tamr=tamr//256
    a2=tamr%256
    tamr=tamr//256
    a3=tamr%256
    tamr=tamr//256
    texto=texto+'0000000'
    tt=bytes([a4])+bytes([a3])+bytes([a2])+bytes([a1])
    i=0
    while i<tamu:
        tt=tt+bytes([int(texto[i+0:i+8],2)])
        i=i+8
    f.write(tt)
    f.close()
```

7. Para ler

```
def lec(onde):
    t=f.read()
    tam=(t[0]*16777216)+(t[1]*65536)+(t[2]*256)+t[3]
    tt=''
    while i<(len(t)):
        tt=tt+bin(t[i][2:]).rjust(8,'0')
        i=i+1
    return tt[0:tam]
```

O conjunto

```
aa=huf2('IVO/VIU/A/UVA')
bb=ca1(aa)
print('Lista de caracteres')
print(aa)
print('Arvore de Huffman')
print(bb)
global dicionario
dicionario=[]
```

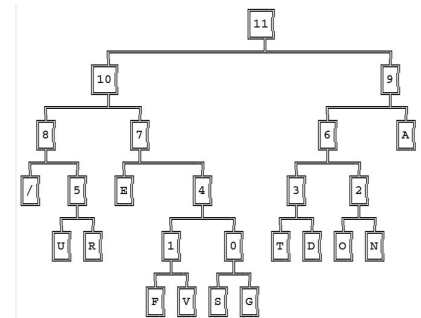
```
cdic1(len(bb)-1,'',bb)
print('Dicionário')
print(dicionario)
tc=comprime1('IVO/VIU/A/UVA',dicionario)
print('Resultado da compressão')
print(tc)
td=descomprime(tc,bb)
gravac(tc,'c:/p/n/451/arquivo1')
lido=lec('c:/p/n/451/arquivo1')
td=descomprime(lido,bb)
print('Resultado da descompressão')
print(td)
```

Exemplos

Seja a frase:

E/NAO/DE/AGRESTE/AVENA/OU/FRAUTA/RUDA - / é 000

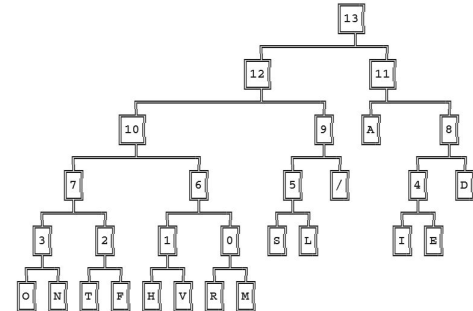
A árvore gerada é



Mais um exemplo:

MARAVILHA/FATAL/DA/NOSSA/IDADE - / é 011

Neste último exemplo, a árvore é



Para você fazer

Suponha que – usando os algoritmos acima – precisa comprimir a frase a seguir, extraída d'Os Lusíadas:

CESAREA/OU/CRISTIANISSIMA/CHAMADA

Qual será a configuração do espaço (caracter /) ?



1. MMC e MDC Nos fundamentos da matemática aprende-se o conceito de Máximo Divisor Comum (MDC) e Mínimo múltiplo comum (MMC). Dados dois inteiros, o MDC é o maior número menor ou igual a ambos que divide os dois sem deixar resto. Por exemplo, o MDC de 10 e 14 é 2. Os divisores de 10 são: 1, 2, 5 e 10. Os divisores de 14 são 1, 2, 7 e 14. O maior número que está em ambas as listas é o 2 que é o MDC. Já o MMC de dois números inteiros é o menor múltiplo de ambos. Os múltiplos de 10 são: 10, 20, 30, 40, 50, 60, 70, ... Os múltiplos de 14 são: 14, 28, 42, 56, 70, 84, ... O menor número em ambas as listas é 70. Então: MDC(10,14)=2 e MMC(10,14)=70. O detalhe é que ambos (o MMC e o MDC) estão relacionados pela expressão:

$$MMC(a,b) = \frac{a \times b}{MDC(a,b)}$$

Como se viu no exemplo acima: $70 = \frac{140}{2}$. Desta maneira, para obter um deles, basta obter o outro. Começar-se-á pelo MDC e usaremos um algoritmo devido a Euclides, cerca de 400 aC. Tem como interessante característica ser recursivo. Abaixo dois algoritmos: o primeiro (completo) é o MDC recursivo e o segundo é o MDC e o MMC iterativos.

```
def mdc(a,b):
    if b==0:
        return a
    else:
        return mdc(b,(a%b))
def mdci(a,b):
    p=a*b
    sa=b
    a=sa
    return [a,(p//a)]
```

Exemplos

x	y	mdc	mmc
123	456	3	18696
1208	4096	8	618496
102	204	102	204

2. Multiplicação de matrizes

```
import numpy as np
def marra(a,b):
    tam=len(a)
    r=np.zeros((tam,tam),float)
    for i in range(tam):
        for j in range(tam):
            olha=0
            for k in range(tam):
                r[i][j]=olha
            return(r)
    #/////////////////////////////////
def strassen(x,y):
    n=len(x)
    if n>4: # ou n>1
        me=int(n/2)
        a1=x[0:me,0:me]
        a12=x[0:me,me:n]
        a21=x[me:n,0:me]
        a22=x[me:n,me:n]
        b11=y[0:me,0:me]
        b12=y[0:me,me:n]
        b21=y[me:n,0:me]
        b22=y[me:n,me:n]
        m1=strassen((a11+a22),(b11+b22))
        m2=strassen((a21+a22),b11)
        m3=strassen(a11,(b12-b22))
        m4=strassen(a22,(b21-b11))
        m5=strassen((a11+a12),b22)
        m6=strassen((a21-a11),(b11+b12))
        m7=strassen((a12-a22),(b21+b22))
        c[0:me,0:me]=m1+m4-m5+m7
        c[0:me,me:n]=m3+m5
        c[me:n,0:me]=m2+m4
        c[me:n,me:n]=m1-m2+m3+m6
    else:
        return marra(x,y) #x*y
```

Exemplos Sejam as matrizes

45	25	42	154	152	121
78	68	36	102	125	176
59	40	58	173	195	109

A soma de todos os elementos da multiplicação matricial é 196185.

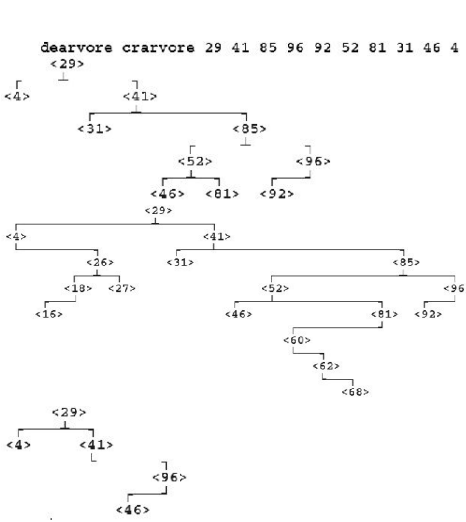
3. Inclusão em ABP

```
def incabp(arv,x):
    arv1=np.resize(arv,(1+tam,3))
    arv1[tam]=[-1,-1,x]
    term=0
    while term != -1:
        adr=term
        esqdir=0
        term=arv1[term,0]
        else:
            term=arv1[term,1]
        arv1[adr,esqdir]=tam
        return arv1
```

4. Exclusão em ABP

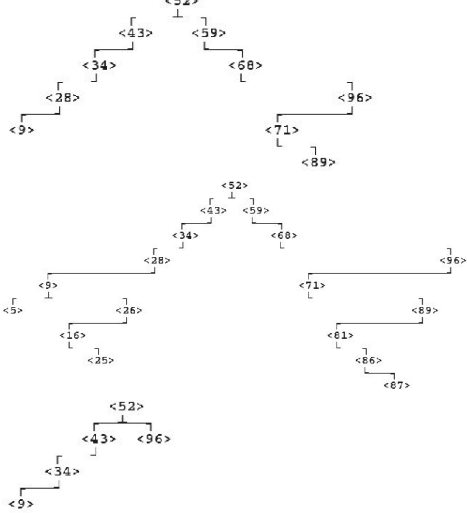
```
def excabp(raiz,x):
    global arv
    term=raiz
    pai=term
    if x<arv[term,2]:
        esqdir=0
        term=arv[term,0]
        else:
            term=arv[term,1]
    if arv[term,0]==-1 and arv[term,1]==-1:
        arv[pai,esqdir]=-1
    else:
        if arv[term,0]==-1 and arv[term,1]==-1:
            arv[pai,esqdir]=arv[term,1]
            return term
        else:
            if arv[term,0]!=-1 and arv[term,1]==-1:
                arv[pai,esqdir]=arv[term,0]
                return term
            else:
                elex=arv[term,1]
                while elex!=-1:
                    save=elex
                    elex=arv[elex,0]
                tempo=excabp(raiz,arv[save,2])
                arv[term,2]=arv[tempo,2]
                return term
```

Exemplos Seja a árvore formada por 29 41 85 96 92 52 81 31 46 4. Depois inclui-se 60 26 62 18 68 16 27. Nesta hora o décimo elemento na busca em largura é 96. Excluindo da árvore original os números 85, 92, 52, 81 e 31 e mantendo 29 41 96 46 4, o quarto em largura também é o 96 (mera coincidência).



Outro exemplo: Inicial: 52 43 34 28 59 68 96 71 9 89. Inclusões 81 26 86 87 5 16 25. Décimo é

o 5. Exclusões: 28, 59, 68, 71 e 89. Ficaram 52 43 34 96 9. Quarto é o 34.



Para você fazer

1. Qual o MMC e o MDC de

9802 , 46338

2. Multiplicação matricial Multiplique as matrizes a seguir e ache a soma de todos os elementos do resultado

23	48	68	46	66	18	2	14	49	7	10	42	58	14	57	27
54	49	9	40	67	57	26	41	39	45	60	4	54	41	5	16
22	55	16	62	38	31	24	37	24	8	29	13	9	15	59	34
4	69	61	56	13	20	19	70	62	61	3	11	69	51	50	66
58	50	5	52	21	64	1	60	2	19	65	25	23	30	38	56
36	47	53	25	6	35	43	10	67	55	35	48	26	46	40	43
42	44	7	65	32	30	63	59	12	31	18	20	32	21	68	70
45	3	27	15	33	12	51	34	36	63	17	37	33	44	6	28

ABP Primeiro crie uma ABP com os seguintes números (em ordem)

56 23 41 68 44 57 66 72 10 47

3. Inclusões Inclua na ABP acima os seguintes números

14 7 28 48 18 92 86

Responda qual o décimo elemento na busca em largura

4. Exclusões Considere a ABP original (sem as inclusões acima) e exclua

23 41 66 44 47

Responda qual o quarto elemento na busca em largura.

mmc	mdc	soma	décimo	quarto

O mundo descentralizado

Começando com a Arpanet, filha direta da guerra fria, (suprema ironia: como a humanidade progride quando precisa matar mais a custos menores...) o mundo caminhou nos últimos 50 anos em direção à descentralização. Aqui a lei fundamental é que não há um lugar privilegiado, todos são iguais, e sobretudo, todos são descartáveis. Repare na disponibilidade desta arquitetura e consequentemente na sua confiabilidade. É tudo o que a gente precisa.

Uma belíssima construção teórico/prática é a do Bitcoin: é uma arquitetura admirável, trouxe inúmeras contribuições brilhantes, e deu origem a dezenas de tecnologias poderosas ainda mais quando usadas em conjunto. Glória e exaltação a Satoshi Nakamoto, seja ele quem for. Como diz a wikipedia:

Estima-se que Nakamoto tenha uma fortuna de aproximadamente um milhão de bitcoins. A verdadeira identidade de Nakamoto permanece desconhecida, e tem sido objeto de muita especulação. Não se sabe se o nome "Satoshi Nakamoto" é real ou um pseudônimo, ou se o nome de uma pessoa ou representa um grupo de pessoas.

Talvez a principal contribuição seja a do Blockchain, um conjunto de certificações descentralizadas e rastreáveis. À medida em que as transações não sendo efetuadas elas são registradas em um bloco por milhares de gestores/mineradores. Quando o bloco for encerrado, mineradores entram em mineração deste bloco e quando o primeiro minerador for bem sucedido, o bloco é disseminado e incluído na blockchain por todos os participantes.

Este exercício vai se concentrar numa tecnologia básica do Blockchain que é a certificação: como garantir que os (pedaços de) blocos trocados na rede sejam todos válidos e não resultado de algum pilantra tentando iludir a rede e ganhar algum. Como os blocos tendem a crescer em tamanho, necessita-se um algoritmo de certificação eficiente e que cresça menos em exigência computacional do que o tamanho dos blocos. Quem faz isso é o algoritmo da árvore de Merkle. Ao invés de crescer em proporção ao tamanho *n* do bloco, ele cresce em proporção a $\log_2 n$, o que é muito melhor. Acompanhe: $\log_2$ de 10 = 3.3, o $\log_2$ de 100 = 6.6, e o $\log_2$ de 1000000 = 19.9.

Árvores de Merkle

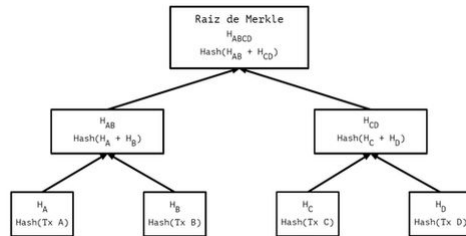
Inventadas em 1979 por Ralph Merkle, e devidamente patenteadas (patente US4309569 A) é uma estrutura de dados que serve para garantir a integridade de um bloco maior de dados. É uma extensão do conceito de Hash.

O hash é uma assinatura binária, obtida a partir de aplicação de um algoritmo a um objeto binário. É um cálculo simples (!) em uma direção, do blob → hash, mas impossível na direção oposta: hash → blob.

Neste contexto, as árvores de Merkle são usadas para garantir que blocos de dados recebidos de outros pares, em uma rede P2P, são recebidos intactos e inalterados. É usada para verificar se outros pares não mentem e enviam blocos falsos.

A principal diferença do uso simples de um hash para verificar a integridade de um bloco de dados é que cada ramo (pedaço) pode ser verificado imediatamente, embora toda a árvore não esteja disponível. Esta separação é eficiente pois permite que apenas pequenos blocos sejam baixados novamente caso estejam danificados.

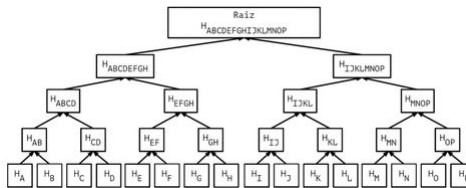
No bloco do bitcoin (que lembrando é formado pelas transações aceitas no período de tempo do bloco) vai a raiz da árvore de Merkle. Todas as transações são dispostas em uma árvore binária. Cada transação tem o seu *id* que é o hash da transação concatenado com *id* da transação irmã. veja-se numa figura



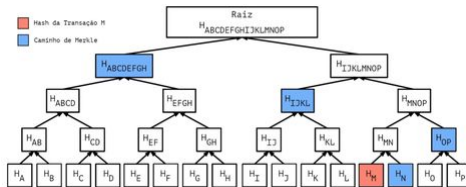
Para isso, todas as transações devem ser colocadas no mesmo nível como na imagem e formarem duplas para criarem os níveis acima sucessivamente. Em caso do bloco não ter um número par de transações, tudo que se faz é repetir a última transação.

Obviamente, no Bitcoin, estas árvores são muito maiores e proporcionais ao número de transações de cada bloco e em cenários desta magnitude que esta estrutura se torna uma solução para comprovação eficiente da existência e integridade de uma transação num dado bloco. Cada hash deste tem o tamanho de 32 bytes e a complexidade de busca na árvore de Merkle cresce $O(\log_2(N))$ na notação "Big-O" com *N* sendo o número de transações.

Vejamos um exemplo um pouco maior de uma árvore de Merkle criada a partir de 16 transações:



Caso precisemos comprovar que a transação M está incluída no bloco, precisamos de apenas 4 hashes de 32 bytes num total de 128 bytes para formar o nosso caminho de Merkle. Veja como o caminho de Merkle junto com a raiz de Merkle é tudo que precisamos para comprovar que um bloco inclui a transação M:



A eficiência da árvore de Merkle para este objetivo vai se tornando mais óbvia à medida em que aumenta-se o número de transações e comparamos com o número de bytes necessários para comprovar a existência de uma transação nestes números maiores:

Transações	Tamanho Aproximado do Bloco	Caminho de Merkle (hashes)	Caminho de Merkle (bytes)
16	4 K	4	128
512	128 K	9	288
2048	512 K	11	352
65.525	16 M	16	512

O que se observa é que com poucas transações não parece fazer muito sentido o uso da árvore de Merkle, mas logo que o número de transações começa a saltar podemos ver claramente a otimização que esta estrutura traz ao sistema do Bitcoin.

Para você fazer Vamos simular uma Árvore de Merkle usando versos de Os Lusíadas, ao invés de blocos de transações. A ideia entretanto, permanece a mesma. Considere primeiro definir uma função simples de *hashing* usando 5 bytes de resultado e usando também o sistema de numeração de base 58, formado pelos seguintes valores: (Porquê 58?: 26+26+10=62, menos zero, I, O e l).

1 2 3
123456789012345678901234567890123456789
123456789ABCDEFGHIJKLMNPQRSTUVWXYZabcde
f
4 5
0123456789012345678
ghijklmnopqrstuvwxyz

Considere a seguinte função de numeração em base 58:

```
def n58(x):  
    print('----',x)  
    b58="123456789ABCDEFGHJKLMNPQRST  
        UVWXYZabcdefghijklmnopqrstuvwxyz"  
    a1=x//(58**4)  
    c5=b58[a1]  
    a2=(x-((58**4)*a1))//(58**3)  
    c4=b58[a2]  
    a3=(x-(((58**4)*a1)+((58**3)*a2)))/(58**2)  
    c3=b58[a3]  
    a4=(x-(((58**4)*a1)+((58**3)*a2)+((58**2)*a3))/  
        //58  
    c2=b58[a4]  
    a5=x-(((58**4)*a1)+((58**3)*a2)+((58**2)*a3))/  
        +58*a4  
    c1=b58[a5]  
    return c5+c4+c3+c2+c1
```

Para testar sua função veja lá:
hs(ASARMASEOSBAROESASSINALADOS) é 1kBUZ
hs(QUEDAOCIDENTALPRAIALUSITANA) é 8QQC1
hs(PORMARESNUCADEANTESNAVEGADOS) é 695rw

Eis aqui a função que calcula o hash e depois converte-o em um número de base 58:

```
def hs(x):  
    ta=len(x)  
    -----  
    i=0  
    a=656356701  
    -----  
    z=ord(x[i])*(i+1)  
    k=k+z+a  
    k=k%656356767 #=(58^5)-1  
    a=k  
    -----  
    return n58(k)
```

1. Agora, imagine que você quer saber se a frase DAQUELESREISQUEFORAMDILATANDO faz parte ou não do bloco de controle. Para sua informação ela deve estar na árvore no nodo: 40 e os hash dos nodos irmão que interessam são 41=1MEp3, 21=12ZNH, 11=11b19, 4=11jhw, 3=11srf, 1=11dQs

2. Frase QUENAOTENHAMINVEJAASDEHIPOCRENE faz parte ou não do bloco de controle? Para sua informação ela deve estar na árvore no nodo: 44 e os hash dos nodos irmão que interessam são 45=13nka, 23=136iw, 10=11mVu, 4=11jhw, 3=11srf, 1=11dQs

3. Frase NAQUALVOSDEUPORARMASEDEIXOU faz parte ou não do bloco de controle? Para sua informação ela deve estar na árvore no nodo: 53 e os hash dos nodos irmão que interessam são 52=4m8aN, 27=11uYH, 12=11s7X, 7=11has, 2=12ACD, 1=11dQs

4. Frase QUENESSETENROGESTOVOSCONTEMPLO faz parte ou não do bloco de controle? Para sua informação ela deve estar na árvore no nodo: 47 e os hash dos nodos irmão que interessam são 46=vshMa, 22=12S8P, 10=11mVu, 4=11jhw, 3=11srf, 1=11dQs

Responda aqui (não: responda 0 e sim: responda 1)

1	2	3	4



Sistemas Lineares - solução ingênua

É muito comum em engenharia e ciências exatas, o surgimento de sistemas (muitas equações) lineares (de grau 1 no máximo), como por exemplo

$$\begin{matrix} 3x & + & 8y & + & 9z & = & 18 \\ -2x & + & y & - & z & = & -11 \\ 5x & - & y & + & 3z & = & 25 \end{matrix}$$

Que tem como resposta $x = 2, y = -3$ e $z = 4$.

Existem inúmeras estratégias para a solução destes sistemas, aqui vai se ver o algoritmo (ingênuo) de escalonamento de Gauss. A idéia é aplicar operações elementares nas linhas, gerando sistemas equivalentes (sistemas equivalentes=têm a mesma solução) de tal maneira que cada linha comece com zeros de maneira escalonada. Acompanhe

- 1. Multiplicando a segunda equação por 5 e a terceira por 2 e somando ambos e substituindo a terceira equação pelo resultado, fica $3y + z = -4$
- 2. Multiplicando a primeira equação por -5 e a terceira por 3 e somando ambos e substituindo a terceira equação pelo resultado fica $-43y - 36z = -15$
- 3. Agora multiplica-se a segunda equação por 43 e a terceira por 3, soma-se e a nova terceira fica: $65z = 260$
- 4. Daqui, o primeiro resultado é $z = 4$. Levando este valor na segunda equação ($3y + z = -5$), fica $y = -3$ e finalmente, levando estes 2 valores na primeira equação ($3x + 8y + 9z = 18$), tem-se $x = 2$, concluindo-se a solução do sistema

O algoritmo que acha esta solução (para qualquer sistema linear onde número de equações = número de incógnitas) é

```
import numpy as np
def fsl(x):
    t=len(x)
    aux=[0.0]*(t+1)
    i=0

    j=t-1
    while j>i:
        if x[j-1][i]==0:
            for k in range(t+1):
                aux[k]=x[j][k]
                x[j][k]=x[j-1][k]
                x[j-1][k]=aux[k]
        if x[j-1][i]==0 and x[j][i]==0:
            ola=0.0
        else:
            ola=(-x[j][i])/x[j-1][i]
        for k in range(t+1):
            x[j][k]=x[j][k]+(x[j-1][k]*ola)

    i=i+1
    i=t-1
    r=[0.0]*t
    while i>=0:
        r[i]=x[i][i+1]/x[i][i]
        for k in range(t):
            x[k][i]=x[k][i+1]-(x[k][i]*r[i])

    print(r)
z=np.array([[3,8,9,18],[-2,1,-1,-11],[5,-1,3,25.0]])
fsl(z)
```

Caixeiro Viajante

Este é um problema célebre na ciência da Computação. Talvez seja o mais famoso problema cuja complexidade (demora) seja $O(n!)$ onde n é o número de cidades que devem ser percorridas. Imagine um vendedor que tem um carro e que deve percorrer um determinado conjunto de cidades. Deve sair da cidade onde reside visitar 1 vez cada uma

das demais cidades e voltar à cidade original, percorrendo a menor distância possível entre elas. Vamos ver o que acontece medida em que o conjunto de cidades cresce.

- 1. Para 1 cidade, não há problema e a resposta é zero km.
- 2. Para 2 cidades, a solução é o dobro da distância entre elas.
- 3. Para 3 cidades, a solução é a distância triangular entre elas.
- 4. Para 4 cidades, digamos: A,B,C e D, começando em A, as opções são ABCDA, ABDCA, ACBDA, ACDBA, ADBCA e ADCBA, num total de 6 trajetos. Olhando a tabela acima, 2 simplificações podem ser notadas:
 - (a) A primeira, é que a cidade origem=destino (A no exemplo) pode ser descartada, já que é fixa.
 - (b) A segunda, é que a lista apresenta resultados duplicados, já que - por exemplo, o caminho ABCDA é o mesmo que ADCBA, só que em ordem inversa.

Neste caso foram 3 caminhos a estudar.

- 5. Para 5 cidades, serão 12 caminhos (Desafio: escreva eles...)
- 6. Para 6 cidades, serão 60 caminhos.
- 7. Para 7 cidades, serão 360 e para 8 cidades, 2520.
- 8. Para n cidades serão $\frac{(n-1)!}{2}$ trajetos.

Para 10 cidades são 181.440 trajetos e para 15 cidades são 43589145600 trajetos. Para 40 cidades serão 10198941040598721679320140869951448678400000 000 trajetos.

Antes de executar o algoritmo necessita-se uma tabela de distâncias entre todas as cidades. (Nada muda no algoritmo, se em vez de distâncias, se usar custo de pedágio, horas de voo, paradas para abastecimento, ..., o que se quiser). Neste caso, como não se tem as distâncias rodoviárias, o que seria o mais correto, e sim as coordenadas X e Y da cidade, há que se construir a tal matriz de distâncias. Isto é feito usando o Teorema de Pitágoras.

```
Eis o algoritmo

# caixeiro viajante
import numpy
no=["Campo Mourao","Curitiba","Fco Beltrao", "Guaira","Irat",
    "Londrina","Ortigueira","Palmas","Paranagua","Paranavai"]
lx=[164,467,91,-25,338,291,310,200,543,158]
ly=[216,63,-8,213,62,297,202,-54,51,323]
mat=numpy.zeros((len(no),len(no)))
for i in range(len(no)):
    for j in range(len(no)):
        mat[i][j]=((lx[i]-lx[j])**2)+((ly[i]-ly[j])**2)**0.5
def cv(lja,lfa,km,mat):
    global minimo
    global caminho
    if len(lfa)+1==0:
        ori=lja[len(lja)-1]
        for i in range(len(lfa)):
            des=lfa[i]
            alfa=lfa[i]
            alfa.remove(des)
            alja=lja[i:]
            akm=km+mat[ori][des]
            cv(alja,alfa,akm,mat)
        else:
            lja.append(lja[0])
            if km<minimo:
                caminho=lja
                minimo=99999999
                caminho=[]
                cv([9],[0,1,2,3,4,5,6,7,8],0,mat)
                print(minimo,' Km')
            for i in range(len(caminho)-1):
                print('de ',no[caminho[i]],' para ',no[caminho[i+1]],
                    ' com ',mat[caminho[i],caminho[i+1]],' Km.')
```

O algoritmo acima é o que dá a resposta correta de maneira absoluta. Só que ele pode demorar demais. E, se você tiver que resolver o problema para (digamos) 40 cidades ? Obviamente não dá para esperar o tempo do algoritmo exato. Precisa-se fazer um abrandamento nos requisitos, para poder obter alguma resposta. Uma boa resposta, ainda que não haja certeza de que ela é a ótima, é melhor do que nada.

Guloso

Uma possível solução é usar um algoritmo guloso e neste caso ele é bem simples. A regra agora é: **visite a cidade mais próxima ainda não visitada**. Agora o algoritmo é muito rápido não importando quantas cidades são. Eis como ficou o algoritmo

```
import numpy as np
def gulosotsp():
    ci=['cascavel','guaira','londrina','maringa',
        'palmas','tel. borba','umarama']
    t=len(ci)
    dfinal=0
    inicio=1
    onde=[inicio]
    dx=np.array([[53,118],[-25,213],[291,297],
        [211,287],[200,-54],[341,185],[72,246]])
    di=np.zeros((t,t),float)
    for i in range(t):
        for j in range(t):
            di[i,j]=(((dx[i,0]-dx[j,0])**2)+((dx[i,1]-dx[j,1])**2)**0.5
    dz=di.copy()
    for i in range(t):
        di[i,inicio]=99999
        qual=inicio

        jj=np.argmax(di[qual,:])
        dfinal=dfinal+min(di[qual,j])
        for k in range(t):
            onde.append(jj)
            qual=jj
        dfinal=dfinal+dz[jj,inicio]
        onde.append(inicio)
        print(dfinal,onde)
    gulosotsp()
```

Para você fazer

- 1. Aplique o algoritmo de escalonamento de Gauss ao sistema abaixo e descubra o valor de x_1 e de x_{10} , (a primeira e a última raízes)

-3	-6	13	29	37	19	27	60	10	12	1465
5	1	17	-18	69	-12	47	27	-28	51	1480
5	-19	17	39	15	58	-1	-24	-27	23	-208
-26	42	2	48	67	-19	33	67	14	55	2001
45	-6	-9	40	25	-23	51	-10	66	5	1250
16	-8	0	-1	62	6	-9	8	14	59	1072
-25	30	44	58	16	49	25	59	-20	-9	904
50	66	43	69	8	-10	-17	-25	46	5	361
-28	28	68	47	50	38	44	31	59	59	2242
-2	-1	-10	18	6	-27	2	63	-26	54	628

- 2. Aplique o algoritmo do caixeiro viajante para o seguinte problema e depois aplique o algoritmo guloso para o mesmo conjunto de cidades.

1-CAMPO MOURAO	164 216
2-CASCABEL	53 118
3-CURITIBA	467 63
4-MARINGA	211 287
5-ORTIGUEIRA	310 202
6-PALMAS	200 -54
7-TELEM BORBA	341 185

Começando e terminando na cidade de CAMPO MOURAO .

Responda aqui:

x_1	x_{10}	cv exato	guloso



Desenvolvendo a lógica de programação

Nesta folha são apresentados 3 conceitos: ABP=Árvore Binária de Pesquisa, MQ=matriz quadrada crescente e PL=Ponto de Lombada. O aluno deve responder a 3 instâncias, uma de cada conceito.

Árvores binárias de pesquisa

Uma ferramenta importante no acesso binário a dados desordenados. Lembrando, a busca sequencial a dados desordenados, custa $O(n)$ onde n é a quantidade de chaves, enquanto a busca binária custa $O(\log n)$, mas exige que os dados sejam mantidos ordenados. A ABP, permite – em tese – um custo $O(\log n)$ em dados desordenados: o melhor de 2 mundos. Na sua modalidade mais simples, uma ABP é uma árvore (raiz, folhas, pai-filho, irmão...) de grau 2, na qual para todos os nodos, vale a regra: *Se a raiz é x, todos os valores menores que x estarão à esquerda e os maiores à direita.* Examine um exemplo de construção de uma ABP. Suponha que queremos incluir os valores

8 7 9 5 3 2 4 1

a) raiz = 8, e a árvore fica

<8>

b) o próximo é 7. Como é menor que 8: à esquerda

<8>

+++++

<7>

c) o próximo é 9

<8>

+++++

<7> <9>

d) agora 5

<8>

+++++

<7> <9>

+++

<5>

e) próximo é 3

<8>

+++++

<7> <9>

+++

<5>

+++

<3>

f) agora é o 2

<8>

+++++

<7> <9>

+++

<5>

+++

<3>

+++

<2>

g) próximo é o 4

<8>

+++++

<7> <9>

+++

<5>

+++++

<3>

+++++

<2>

<4>

h) finalmente o último: 1

<8>

+++++

<7> <9>

+++

<5>

+++++

<3>

+++++

<2> <4>

+++

<1>

A representação espacial da árvore apresenta problemas. De fato, um programa que desenhe uma ABP é bastante complexo. Usam-se diversas representações (mais simples, aqui vai-se usar a representação parentizada: A árvore acima é

(8(7(5(3(2(1())(4))())(9))

Esta representação usa a regra: (raiz(fe(fd)) O exercício aqui é receber uma árvore parentizada, desenhá-la e descobrir quantos nodos estão no maior nível da árvore. Veja um exemplo: Seja a árvore (Nota: a árvore está em 2 linhas apenas por facilidade de caber na margem. Considere-a escrita em uma única linha).

(61(42(19(10(7(16)))(39(21() (25() (31(29() ()))) (41)))(51(44() ())(97(74() (79(76(92)))(99)))

Formada por 20 nodos. O nodo 29 está no nível 7, que é o maior. Logo a resposta ao exercício é 1 (1 nodo).

Mais um exemplo. A árvore

(13(4() (10)) (68(40(21(16(14) (19)) (23)) (59(49(44() ())(76(70) (90(85(82) (87)) (98)))))

Tem cinco nodos no nível 5, que é o maior. Os nodos são 14, 19, 44, 82 e 87.

Matriz quadrada Suponha a matriz que vai sendo construída

8

9

8

10 9

8

10 9

11 8

10 9

11 8

12 13

10 9 16

11 8 15

12 13 14

Neste caso, $a_1 = 8$ e a razão de crescimento $r = 1$.

Se a matriz crescer mais um nível ela vai ficar como

20 19 18 17 32

21 10 9 16 31

22 11 8 15 30

23 12 13 14 29

24 25 26 27 28

Para os valores $a_1 = 11$ e $r = 2$, para 5 níveis, a matriz fica

35 33 31 29 59

37 15 13 27 57

39 17 11 25 55

41 19 21 23 53

43 45 47 49 51

Exemplo Considere uma matriz quadrada com $ordem = 17$, $a_1 = 8$, $r = 8$. Neste caso o elemento da $linha = 6$, $coluna = 3$ é igual a 1952.

Mais um: Considere a matriz com $ordem = 15$, $a_1 = 8$, $r = 4$. Neste caso o elemento da $linha = 1$, $coluna = 8$ é igual a 1500.

Ponto de Lombada Dada uma matriz M de ordem l, c , define-se um possível ponto de lombada (*saddle point*) como sendo um valor $M_{i,j}$ que é ao mesmo tempo

- O menor valor da linha i e
- O maior valor da coluna j

[Horo84].

Exemplo Considere a matriz

174	98	20	107	109	271	13	99	45	44
79	84	184	165	237	173	24	26	292	95
138	100	190	178	48	235	35	282	276	61
53	39	209	38	71	57	27	136	264	239
215	80	208	62	81	140	124	83	290	101
199	167	231	273	288	231	180	318	188	260
34	8	150	245	295	42	262	243	87	289
117	66	64	41	279	157	151	50	118	270

O ponto de lombada está na linha 6, coluna 2.

Mais um exemplo
Seja a matriz

205	205	191	166	130	199	138	190	136	255	135
28	2	165	243	27	252	185	85	222	96	212
131	46	95	44	78	134	119	224	221	61	99
217	187	250	92	94	157	152	143	234	74	149
182	203	210	107	55	248	18	26	204	67	72
69	219	174	93	71	155	29	245	249	123	65
63	87	213	16	47	49	53	42	201	79	122
41	235	162	161	48	9	104	125	244	11	75
89	251	12	206	124	168	110	184	108	140	113

O ponto de lombada está na posição 1,5.

Para você fazer

1. Desenhe a ABP representada por

(54(22(21(6(3() ())(47(30() (40)) (49(48() ()))) (57() (86(80(75(64() (68)) ())(94() (97)))))

Considere a matriz acima escrita em uma única linha. Ela só aparece em 2 linhas por questões de espaço.

Descubra quantos nodos existem no maior nível.

2. Construa uma matriz quadrada de ordem 17 e com termo inicial (a_1) igual a 15 e com razão de crescimento (r) igual a 3 Descubra qual o valor registrado na posição da matriz de linha 2 e coluna 14 sendo ambas medidas com origem dos índices igual a 1.

3. Dada a matriz abaixo

111	245	21	340	25	7	89	158	283	77	225
180	292	308	326	198	331	306	344	325	248	217
92	84	262	31	159	28	205	121	99	33	201
130	160	214	29	261	302	279	93	70	321	43
95	203	289	195	282	320	334	299	273	4	1
169	14	229	134	57	122	127	193	86	171	162
118	304	170	191	146	107	129	102	120	207	114

Localize qual a linha e coluna de um possível ponto de lombada. Se não houver, responda a dupla 0,0. A origem dos índices é 1.

Responda aqui:

1.nodos	2.valor	3.lin	3.col



- 1 - /

Prática em matrizes em C++

Suponha uma matriz em C++ gerada através do seguinte código

```
#include<iostream>
using namespace std;
int criamat(int M[x][y], int vi, int incr, int decap) {
    i=0;
    while (i<x) {
        j=0;
        while (j<y) {
            M[i][j]=vi
            vi=(vi+incr)%decap;
            j=j+1;
        }
        i=i+1;
    }
}
int main() {
    // chama criamat, com os parâmetros dados
    // imprime M (para fazer à mão...) ou
    // programa os valores perdidos...
}
```

Para cada exercício a seguir, os 5 valores importantes serão fornecidos e a seguir 5 características numéricas da matriz terão que ser calculadas. São elas:

- 1. Qual a coluna que contém a maior soma ?
- 2. Qual a linha que contém o valor da menor soma ?
- 3. Quantos primos menores do que 100 existem na matriz ?
- 4. Quantos múltiplos de um dado número existem na matriz ?
- 5. Quantos números na matriz são menores do que os seus 4 vizinhos diretos ? Por óbvio, os elementos das primeira e últimas linhas e primeira e última colunas não devem ser pesquisados, pois eles deixam de ter um ou mais vizinhos.

Para sua referência os primos menores do que 100 são:

2 3 5 7 11 13 17 19 23 29 31 37
41 43 47 53 59 61 67 71 73 79 83 89 97

Exemplo Seja uma instância com os valores $x = 7, y = 9, vi = 14, incr = 17$ e $decap = 89$ com múltiplos de 13 (pergunta 4). A matriz gerada é

14 31 48 65 82 10 27 44 61
78 6 23 40 57 74 2 19 36
53 70 87 15 32 49 66 83 11
28 45 62 79 7 24 41 58 75
3 20 37 54 71 88 16 33 50
67 84 12 29 46 63 80 8 25
42 59 76 4 21 38 55 72 0

- 1. A soma das colunas é
285 315 345 286 316 346 287 317 258
e portanto o maior valor está na linha 6.
- 2. Já a soma das linhas é
382 335 466 419 372 414 367
e o menor valor está na coluna 2.
- 3. Os primos nesta matriz são:
31 61 23 2 19 53 83 11
79 7 41 3 37 71 67 29 59
que totalizam 17
- 4. Os múltiplos de 13 são
65 78
- 5. Finalmente, os elementos menores do que seus 4 vizinhos estão a seguir marcados
14 31 48 65 82 10 27 44 61
78 (6)23 40 57 74 (2)19 36
53 70 87(15)32 49 66 83 11
28 45 62 79 (7)24 41 58 75
3 20 37 54 71 88(16)33 50
67 84(12)29 46 63 80 (8)25
42 59 76 4 21 38 55 72 0

Consolidando as 5 respostas, para a instância do exemplo as respostas são:
6 2 17 2 7

🔗 Para você fazer

1. Lembrando que x é o número de linhas da matriz, y é o número de colunas, vi é o valor inicial da posição (0,0) da matriz, $incr$ é o incremento que deve ser aplicado ao valor da célula anterior para obter a nova célula e finalmente $decap$ é o valor do módulo que deve ser aplicado a todas as células. Na dúvida, olhe o código C++ no início desta folha.

Suponha uma instância onde

x	y	vi	$incr$	$decap$
8	9	16	14	67

Para a pergunta 4, considere múltiplos de 13

2. Suponha uma instância onde

x	y	vi	$incr$	$decap$
7	8	12	17	87

Para a pergunta 4, considere múltiplos de 13

3. Suponha uma instância onde

x	y	vi	$incr$	$decap$
6	11	10	14	103

Para a pergunta 4, considere múltiplos de 19

4. Suponha uma instância onde

x	y	vi	$incr$	$decap$
7	9	11	11	103

Para a pergunta 4, considere múltiplos de 3

5. Suponha uma instância onde

x	y	vi	$incr$	$decap$
7	8	16	15	69

Para a pergunta 4, considere múltiplos de 18 .

Responda aqui:

Ex.	número da maior coluna	número da menor linha	quantos primos < 100	múltiplos do valor dado	menores que os vizinhos
1					
2					
3					
4					
5					



Prática em matrizes em C++ - II

Suponha uma matriz em C++ gerada através do seguinte código

```
#include<iostream>
using namespace std;
int criamat(int M[x][x], int vi, int incr, int inlin, int decap) {
    i=0;
    while (i<x) {
        j=0;
        while (j<x) {
            M[i][j]=vi
            vi=(vi+incr)%decap;
            j=j+1;
        }
        i=i+1;
        vi=(vi+inlin)%decap;
    }
}
int main() {
    // chama criamat, com os parâmetros dados
    // imprime M (para fazer à mao...) ou
    // programa os valores pedidos...
}
```

Para cada exercício a seguir, os 5 valores importantes serão fornecidos e a seguir 5 características numéricas da matriz terão que ser calculadas. São elas:

- 1. Qual é o maior valor da diagonal principal ?
- 2. Qual é o menor valor da diagonal secundária ?
- 3. Quantos números da matriz têm mais do que 4 divisores ?
- 4. Quantos números da matriz têm mais de uma instância na matriz ? (cópia)
- 5. Quantos números na matriz são maiores do que os seus 3 vizinhos da esquerda OU são menores do que seus 3 vizinhos da direita ? Por óbvio, os elementos das primeira e últimas linhas e primeira e última colunas não devem ser pesquisados, pois eles deixam de ter um ou mais vizinhos.

Exemplo Seja uma instância na qual os valores gerados foram x = 8, vi = 13, incr = 9, inlin = 17 e decap = 83, o que gerou a seguinte matriz:

13	22	31	40	49	58	67	76
19	28	37	46	55	64	73	82
25	34	43	52	61	70	79	5
31	40	49	58	67	76	2	11
37	46	55	64	73	82	8	17
43	52	61	70	79	5	14	23
49	58	67	76	2	11	20	29
55	64	73	82	8	17	26	35

- 1. A diagonal principal é

13 28 43 58 73 5 20 35

O maior valor é 73.
- 2. A diagonal secundária é

76 73 70 67 64 61 58 55

e o menor valor é 55.
- 3. Os números da matriz que têm mais do que 4 divisores são:

40 76 28 64 52 70 40 76 64 52 70 76 20 64

e o total deles é 14.
- 4. Os números duplicados (ou mais) na matriz são

31 40 49 58 67 76 76 37 46 55 64 73
82 43 52 61 70 79 5 31 40 49 58
67 76 2 11 37 46 55 64 73 82 8
17 43 52 61 70 79 5 49 58 67 76
2 11 55 64 73 82 8 17

No total de 52 números. Note que quando um número é duplicado ele é contado 2 vezes.
- 5. Finalmente, os que são maiores que seus vizinhos da esquerda OU menores que os vizinhos da direita são os números (na ordem em que eles foram achados)

em número de 30.

Então a resposta a esta instância é

73 55 14 52 30

Para você fazer

1. Lembrando que x é a ordem da matriz, vi é o valor inicial da posição (0,0) da matriz, incr é o incremento que deve ser aplicado ao valor da célula anterior para obter a nova célula, inlin é o incremento após preencher cada linha e finalmente decap é o valor do módulo que deve ser aplicado a todas as células. Na dúvida, olhe o código C++ no início desta folha.

Suponha uma instância onde

x	vi	incr	inlin	decap
17	14	9	11	75

2. Suponha uma instância onde

x	vi	incr	inlin	decap
13	11	13	11	105

3. Suponha uma instância onde

x	vi	incr	inlin	decap
15	12	13	11	75

4. Suponha uma instância onde

x	vi	incr	inlin	decap
8	15	17	15	67

Responda aqui:

Ex.	maior na d.p.	menor na diag, sec.	quantos com + 4 divs	quantos com cópia	men ou mai que os vizinhos
1					
2					
3					
4					

Prática em matrizes em Python

Suponha uma matriz em Python gerada através do seguinte código

```
import numpy as np
def criamat(M, vi, incr, decap):
    i=0
    xx=len(M)
    yy=len(M[0])
    while i<xx:
        j=0
        while j<yy:
            M[i,j]=vi
            vi=(vi+incr) % decap
            j=j+1
        i=i+1
    return M
M=np.zeros((x,y))
print (criamat(M,vi,incr,decap))
```

Para cada exercício a seguir, os 5 valores importantes serão fornecidos e a seguir 5 características numéricas da matriz terão que ser calculadas. São elas:

- 1. Qual a coluna que contém a maior soma ?
- 2. Qual a linha que contém o valor da menor soma ?
- 3. Quantos primos menores do que 100 existem na matriz ?
- 4. Quantos múltiplos de um dado número existem na matriz ?
- 5. Quantos números na matriz são menores do que os seus 4 vizinhos diretos ? Por óbvio, os elementos das primeira e últimas linhas e primeira e última colunas não devem ser pesquisados, pois eles deixam de ter um ou mais vizinhos.

Para sua referência os primos menores do que 100 são:

2 3 5 7 11 13 17 19 23 29 31 37
41 43 47 53 59 61 67 71 73 79 83 89 97

Exemplo Seja uma instância com os valores $x = 7$, $y = 9$, $vi = 14$, $incr = 17$ e $decap = 89$ com múltiplos de 13 (pergunta 4). A matriz gerada é

14 31 48 65 82 10 27 44 61
78 6 23 40 57 74 2 19 36
53 70 87 15 32 49 66 83 11
28 45 62 79 7 24 41 58 75
3 20 37 54 71 88 16 33 50
67 84 12 29 46 63 80 8 25
42 59 76 4 21 38 55 72 0

- 1. A soma das colunas é

285 315 345 286 316 346 287 317 258

e portanto o maior valor está na linha 6.
- 2. Já a soma das linhas é

382 335 466 419 372 414 367

e o menor valor está na coluna 2.
- 3. Os primos nesta matriz são:

31 61 23 2 19 53 83 11
79 7 41 3 37 71 67 29 59

que totalizam 17
- 4. Os múltiplos de 13 são

65 78
- 5. Finalmente, os elementos menores do que seus 4 vizinhos estão a seguir marcados

14 31 48 65 82 10 27 44 61
78 (6)23 40 57 74 (2)19 36
53 70 87(15)32 49 66 83 11
28 45 62 79 (7)24 41 58 75
3 20 37 54 71 88(16)33 50
67 84(12)29 46 63 80 (8)25
42 59 76 4 21 38 55 72 0

Consolidando as 5 respostas, para a instância do exemplo as respostas são:

6 2 17 2 7

🔗 Para você fazer

1. Lembrando que x é o número de linhas da matriz, y é o número de colunas, vi é o valor inicial da posição (0,0) da matriz, $incr$ é o incremento que deve ser aplicado ao valor da célula anterior para obter a nova célula e finalmente $decap$ é o valor do módulo que deve ser aplicado a todas as células. Na dúvida, olhe o código Python no início desta folha.

Suponha uma instância onde

x	y	vi	$incr$	$decap$
8	8	11	14	95

Para a pergunta 4, considere múltiplos de 13

2. Suponha uma instância onde

x	y	vi	$incr$	$decap$
6	8	12	9	109

Para a pergunta 4, considere múltiplos de 13

3. Suponha uma instância onde

x	y	vi	$incr$	$decap$
7	10	14	19	69

Para a pergunta 4, considere múltiplos de 4

4. Suponha uma instância onde

x	y	vi	$incr$	$decap$
9	9	9	19	65

Para a pergunta 4, considere múltiplos de 7

5. Suponha uma instância onde

x	y	vi	$incr$	$decap$
7	8	9	17	99

Para a pergunta 4, considere múltiplos de 10 .

Responda aqui:

Ex.	número da maior soma das cols	número da menor soma das lins	quantos primos < 100	múltiplos do valor dado	menores que os vizinhos
1					
2					
3					
4					
5					



- 1 - /

Prática em matrizes em Python - II

Suponha uma matriz em Python gerada através do seguinte código

```
import numpy as np
def criamat(M, vi, incr, decap):
    i=0
    xx=len(M)
    yy=len(M[0])
    while i<xx:
        j=0
        while j<yy:
            M[i,j]=vi
            vi=(vi+incr) % decap
            j=j+1
        i=i+1
        vi=(vi+inlin) % decap
    return M
M=np.zeros(x,y)
print criamat(M,vi,incr,decap)
```

Para cada exercício a seguir, os 5 valores importantes serão fornecidos e a seguir 5 características numéricas da matriz terão que ser calculadas. São elas:

- 1. Qual é o maior valor da diagonal principal ?
- 2. Qual é o menor valor da diagonal secundária ?
- 3. Quantos números da matriz têm mais do que 4 divisores ?
- 4. Quantos números da matriz têm mais de uma instância na matriz ? (cópia)
- 5. Quantos números na matriz são maiores do que os seus 3 vizinhos da esquerda OU são menores do que seus 3 vizinhos da direita ? Por óbvio, os elementos das primeira e últimas linhas e primeira e última colunas não devem ser pesquisados, pois eles deixam de ter um ou mais vizinhos.

Exemplo Seja uma instância na qual os valores gerados foram $x = 8$, $vi = 13$, $incr = 9$, $inlin = 17$ e $decap = 83$, o que gerou a seguinte matriz:

13	22	31	40	49	58	67	76
19	28	37	46	55	64	73	82
25	34	43	52	61	70	79	5
31	40	49	58	67	76	2	11
37	46	55	64	73	82	8	17
43	52	61	70	79	5	14	23
49	58	67	76	2	11	20	29
55	64	73	82	8	17	26	35

- 1. A diagonal principal é

13 28 43 58 73 5 20 35

O maior valor é 73.
- 2. A diagonal secundária é

76 73 70 67 64 61 58 55

e o menor valor é 55.
- 3. Os números da matriz que têm mais do que 4 divisores são:

40 76 28 64 52 70 40 76 64 52 70 76 20 64

e o total deles é 14.
- 4. Os números duplicados (ou mais) na matriz são

31 40 49 58 67 76 76 37 46 55 64 73
82 43 52 61 70 79 5 31 40 49 58
67 76 2 11 37 46 55 64 73 82 8
17 43 52 61 70 79 5 49 58 67 76
2 11 55 64 73 82 8 17

No total de 52 números. Note que quando um número é duplicado ele é contado 2 vezes.
- 5. Finalmente, os que são maiores que seus vizinhos da esquerda OU menores que os vizinhos da direita são os números (na ordem em que eles foram achados)

28 37 46 55 64 73 34 43 52 61
70 79 40 49 58 67 76 46 55 64
73 82 52 61 70 79 58 67 76 20

em número de 30.

Então a resposta a esta instância é

73 55 14 52 30

🔗 Para você fazer

1. Lembrando que x é a ordem da matriz, vi é o valor inicial da posição (0, 0) da matriz, $incr$ é o incremento que deve ser aplicado ao valor da célula anterior para obter a nova célula, $inlin$ é o incremento após preencher cada linha e finalmente $decap$ é o valor do módulo que deve ser aplicado a todas as células. Na dúvida, olhe o código Python no início desta folha.

Suponha uma instância onde

x	vi	$incr$	$inlin$	$decap$
15	13	19	14	115

2. Suponha uma instância onde

x	vi	$incr$	$inlin$	$decap$
9	14	19	14	91

3. Suponha uma instância onde

x	vi	$incr$	$inlin$	$decap$
9	11	13	14	75

4. Suponha uma instância onde

x	vi	$incr$	$inlin$	$decap$
8	11	14	7	65

Responda aqui:

Ex.	maior na d.p.	menor na diag. sec.	quantos com + 4 divs	quantos com cópia	men ou mai que os vizinhos
1					
2					
3					
4					



Matrizes e Equações Lineares

O sistema

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n &= b_1 \\ a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n &= b_2 \\ \dots & \\ a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n &= b_m \end{cases}$$

é equivalente à seguinte equação matricial

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix}$$

no sentido de que toda solução do primeiro é igualmente solução do segundo. Dessa maneira pode-se resumir qualquer dos sistemas acima na forma matricial escrevendo

$$Ax = B$$

Matrizes Inversíveis

Diz-se que uma matriz quadrada é inversível se existe uma matriz B com a propriedade de que AB = BA = I onde I é a matriz identidade.

A matriz B inversa de A é única se existir. A matriz inversa de A é indicada por A⁻¹. Note-se que a relação acima é simétrica, pois se B é a inversa de A então A também é inversa de B.

Mecanismos de cálculo da inversa

Usando a teoria: Usando a teoria, dada uma matriz como por exemplo $\begin{pmatrix} 3 & 5 \\ 2 & 3 \end{pmatrix}$ Uma possibilidade é querer encontrar escalares x, y, z e w para os quais $\begin{pmatrix} 3 & 5 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x & y \\ z & w \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ou $\begin{pmatrix} 3x + 5z & 3y + 5w \\ 2x + 3z & 2y + 3w \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ou ainda equações que satisfaçam $\begin{cases} 3x + 5z = 1 \\ 2x + 3z = 0 \end{cases}$ e $\begin{cases} 3y + 5w = 0 \\ 2y + 3w = 1 \end{cases}$ Resolvendo estas equações, acha-se a matriz inversa que é $\begin{pmatrix} -3 & 5 \\ -2 & 3 \end{pmatrix}$.

Usando determinantes: Outros métodos (extraídos do excelente site <http://mathworld.wolfram.com>) vão a seguir descritos: Dada uma matriz 2 x 2 representada por $A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$, a inversa é dada por $A^{-1} = \frac{1}{|A|} \begin{vmatrix} d & -b \\ -c & a \end{vmatrix} = \frac{1}{ad-bc} \begin{vmatrix} d & -b \\ -c & a \end{vmatrix}$.

Para uma matriz 3 x 3 veja lá no site, que o espaço aqui é muito pequeno.

Usando Gauss-Jordan: Neste método escreve-se a matriz da qual se quer achar a inversa ao lado esquerdo da matriz identidade, formando este arranjo (matriz escalonada reduzida por linhas).

$$[AI] = \left[\begin{array}{ccc|ccc} a_{11} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & \dots & a_{2n} & 0 & 1 & \dots & 0 \\ \dots & & & \dots & & & \\ a_{n1} & \dots & a_{nn} & 0 & 0 & \dots & 1 \end{array} \right]$$

Aplica-se agora a este arranjo as técnicas de eliminação de Gauss, sobre o conjunto todo, de maneira a ficar com o seguinte arranjo $\left[\begin{array}{ccc|ccc} 1 & 0 & \dots & 0 & b_{11} & \dots & b_{1n} \\ 0 & 1 & \dots & 0 & b_{21} & \dots & b_{2n} \\ \dots & & & & & & \\ 0 & 0 & \dots & 1 & b_{n1} & \dots & b_{nn} \end{array} \right]$

Neste ponto, a matriz (b_{ij}) é a matriz inversa de A. Ou seja A⁻¹ = B e também AB = I.

A vantagem deste último método é que ele vale para qualquer dimensão de matriz. É este método que vai ser programado em C++ a seguir.

Para que serve a matriz inversa ?

Inúmeras aplicações, mas vamos focar aqui na solução de sistemas lineares. Dado o sistema Ax = B, a matriz A é a dos coeficientes. Se você calcular a inversa A⁻¹, poderá resolver inúmeras instâncias do mesmo sistema, sem outra preocupação, já que A⁻¹ x B = x. Vamos ver um exemplo do que se diz aqui. Seja um sistema, dado por

$$\begin{cases} 3x + 2y + z &= \\ x - y + z &= \\ 5x + z &= \end{cases}$$

Escrito na forma matricial, o sistema fica Ax = B, onde A é a matriz

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & -1 & 1 \\ 5 & 0 & 1 \end{pmatrix}$$

Calculando a matriz inversa desta, obtém-se:

$$\begin{pmatrix} -0.1 & -0.2 & 0.3 \\ 0.4 & -0.2 & -0.2 \\ 0.5 & 1 & -0.5 \end{pmatrix}$$

Agora, a solução para a instância onde B = {10, 2, 8} é obtida fazendo-se A⁻¹ x B que dá os valores x = 1, y = 2, z = 3. Para outra instância cujo B seja {30, -1, 26}, ao fazer a mesma multiplicação, obtém-se x = 5, y = 7, z = 1 e assim por diante.

Este exercício pede que você implemente o algoritmo de Gauss-Jordan para cálculo da matriz inversa e depois o aplique a uma matriz de ordem suficientemente alta para desencorajar o cálculo manual da mesma. Relembrando as etapas, você deve:

- 1. Escrever a matriz A e ao seu lado a matriz I.
- 2. Aplicar operações elementares com linhas a A visando transformá-la em I.
- 3. Aplicar rigorosamente as mesmas operações sobre a matriz ao lado.
- 4. Quando A tiver se transformado em I, a matriz que era originalmente I se transformou em A⁻¹.
- 5. Para conferir, faça a multiplicação matricial entre A e A⁻¹. Tem que dar I.

Vamos aplicar este processo ao sistema acima descrito. O objetivo é produzir um zero no coeficiente do x na segunda equação: Para isso: a[1][1] = a[1][1] + a[0][1] x (-a[1][0]/a[0][0]) onde a notação a[1][1] deve ser entendida como a aplicação da operação à toda a linha [1] da matriz a. Mas, atenção: isto não vale no compilador C++. Lá, tem que ser por extensão.

Note-se também que o fator (-a[1][0]/a[0][0]) é o mesmo para toda a linha e portanto deve ser calculado antes e preservado durante todo o ciclo. O algoritmo vai chamar este fator de pivot e ele vai ser previamente calculado e guardado.

para calcular o zero no coeficiente de x na terceira linha, o comando é a[2][1] = a[2][1] + a[0][1] x (-a[2][0]/a[0][0]).

Para a segunda coluna, começa-se achando o zero na primeira linha. O comando agora é a[0][2] = a[0][2] + a[1][2] x (-a[0][1]/a[1][1]) e depois a[2][2] = a[2][2] + a[1][2] x (-a[2][1]/a[1][1]). A terceira coluna é a[0][3] = a[0][3] + a[2][3] x (-a[0][2]/a[2][2]) e a[1][3] = a[1][3] + a[2][3] x (-a[1][2]/a[2][2]). Com estas operações, os zeros foram alcançados. Agora basta transformar os elementos da diagonal principal na unidade. As operações são: a[0][0] = a[0][0]/a[0][0], a[1][1] = a[1][1]/a[1][1] e a[2][2] = a[2][2]/a[2][2].

Note que não houve preocupação em determinar se a matriz A é inversível. Isto foi deixado para não complicar demais o algoritmo. Fica como desafio para os mais corajosos.

Eis o algoritmo:

```
#include<iostream>
#include<iomanip>
#define ta 5
using namespace std;
int cami(float x[ta][ta], float y[ta][ta]){
// x é a matriz A e y é a matriz A^-1
int i,j,k;
float pivot,alvo;
for (i=0;i<ta;i++){
for (j=0;j<ta;j++){
if (i==j) {
y[i][j]=1;

```

```

}
else {
y[i][j]=0;
}
}
}
for (i=0;i<ta;i++){
for (j=0;j<ta;j++){
if (j!=i){
pivot=-x[j][i]/x[i][i];
for (k=0;k<ta;k++){
x[j][k]=x[j][k]+x[i][k]*pivot;
y[j][k]=y[j][k]+y[i][k]*pivot;
}
}
}
alvo=x[i][i];
for (k=0;k<ta;k++){
y[i][k]=y[i][k]/alvo;
x[i][k]=x[i][k]/alvo;
}
}
return 0;
}
int main() {
float mat[ta][ta]={{{...},{...},{...}}};
float res[ta][ta];
int i,j;
cami(mat,res);
for (i=0;i<ta;i++){
for (j=0;j<ta;j++){
cout<<fixed<<setprecision(5)
<<res[i][j]<<" ";
}
cout<<endl;
}
}
```

Um exemplo Para você testar seu algoritmo, use-o nesta matriz:

59	57	44	42	65	23	47	66	78	71
94	36	83	17	12	86	4	92	43	95
12	61	18	10	94	45	94	81	30	76
9	49	92	22	90	22	27	6	21	26
31	45	96	26	26	75	59	39	47	98
33	27	9	94	29	5	49	68	9	33
61	81	17	18	61	74	16	37	34	45
64	85	46	74	49	81	69	20	18	59
17	54	66	65	94	54	76	54	30	76
68	91	61	89	6	57	35	92	52	80

Aqui a resposta (sexta linha, sétima coluna, vezes 1000) deverá ser 151.989, já devidamente multiplicada por 1000.

Para você fazer

A seguir uma matriz A de dimensão 10 x 10 (o que corresponderia a um sistema linear de 10 equações e 10 incógnitas). Você deve calcular a matriz inversa, seguindo o algoritmo acima, e depois responder qual o valor que está localizado na sexta linha e sétima coluna de A⁻¹, (em C: a[5][6]) multiplicado por 1000. Eis a matriz:

75	18	11	56	81	82	14	3	89	99
79	30	87	16	25	17	9	30	60	81
96	54	11	10	7	94	8	55	57	24
91	25	21	76	92	79	56	20	50	24
21	49	51	67	50	16	82	2	16	86
74	87	7	19	59	81	47	53	49	8
98	20	75	9	94	11	64	82	91	46
38	49	90	10	48	95	64	38	29	54
25	34	7	45	17	57	68	44	64	8
6	65	85	57	77	58	93	46	76	5

Responda aqui:

A⁻¹[5][6] x 1000 =



Matrizes e Equações Lineares

O sistema

{ a11 x1 + a12 x2 + ... + a1n xn = b1
a21 x1 + a22 x2 + ... + a2n xn = b2
...
am1 x1 + am2 x2 + ... + amn xn = bm

é equivalente à seguinte equação matricial

(a11 a12 ... a1n
a21 a22 ... a2n
...
am1 am2 ... amn) (x1
x2
.
.
xn) = (b1
b2
.
.
bm)

no sentido de que toda solução do primeiro é igualmente solução do segundo. Dessa maneira pode-se resumir qualquer dos sistemas acima na forma matricial escrevendo

Ax = B

Matrizes Inversíveis

Diz-se que uma matriz quadrada é inversível se existe uma matriz B com a propriedade de que AB = BA = I onde I é a matriz identidade. A matriz B inversa de A é única se existir. A matriz inversa de A é indicada por A^-1. Note-se que a relação acima é simétrica, pois se B é a inversa de A então A também é inversa de B.

Mecanismos de cálculo da inversa

Usando a teoria: Usando a teoria, dada uma matriz como por exemplo (3 5 / 2 3)

Uma possibilidade é querer encontrar escalares x, y, z e w para os quais (3 5 / 2 3) (x y / z w) = (1 0 / 0 1) ou (3x+5z 3y+5w / 2x+3z 2y+3w) = (1 0 / 0 1) ou ainda equações que satisficam { 3x+5z=1 3y+5w=0 / 2x+3z=0 2y+3w=1 Resolvendo estas equações, acha-se a matriz inversa que é (-3 5 / -2 3).

Usando determinantes: Outros métodos (extraídos do excelente site http://mathworld.wolfram.com) vão a seguir descritos: Dada uma matriz 2 x 2 representada por A = | a b / c d |, a inversa é dada por A^-1 = 1/|A| | d -b / -c a | = 1/(ad-bc) | d -b / -c a |.

Para uma matriz 3 x 3 veja lá no site, que o espaço aqui é muito pequeno. Usando LU: Decompondo A = L.U como já estudado, multiplica-se L por um vetor coluna {c1, c2, ..., cn} igual ao vetor coluna {1, 0, 0...}. A seguir faz-se U.x = {c1, c2, ..., cn}, e obtem-se a primeira coluna da matriz inversa. Acompanhe um exemplo:

(5 -8 2 / 3 -1 4 / 5 7 1) = (1 0 0 / 0.6 1 0 / 1 3.9 1) (5 -8 2 / 0 3.8 2.8 / 0 0 -12.0)
(1 0 0 / 0.6 1 0 / 1 3.9 1) . (c1 / c2 / c3) = (1 / 0 / 0)
c1 = 1, c2 = -0.6 e c3 = 1.36841. Agora

(5 -8 2 / 0 3.8 2.8 / 0 0 -12.0) = (x1 / x2 / x3) = (1 / -0.6 / 1.3)

e daqui: x3 = -0.11353, x2 = -0.07423 e x1 = 0.12663 que é a primeira coluna da matriz inversa. A seguir, deve-se repetir o ciclo para o vetor coluna {0, 1, 0...} e se obterá a segunda coluna da matriz inversa. Segue-se até o final. A matriz inversa completa é

array([[0.12663755, -0.09606987, 0.13100437],
[-0.07423581, 0.02183406, 0.06113537],
[-0.11353712, 0.32751092, -0.08296943]])

Usando Gauss-Jordan: Neste método escreve-se a matriz da qual se quer achar a inversa ao lado esquerdo da matriz identidade, formando este arranjo (matriz escalonada reduzida por linhas).

[AI] = | a11 ... a1n 1 0 ... 0
a21 ... a2n 0 1 ... 0
...
an1 ... ann 0 0 ... 1 |
Aplica-se agora a este arranjo as técnicas de eliminação de Gauss, sobre o conjunto todo, de maneira a ficar com o seguinte arranjo | 1 0 ... 0 b11 ... b1n
0 1 ... 0 b21 ... b2n
...
0 0 ... 1 bn1 ... bnn |

Neste ponto, a matriz (bij) é a matriz inversa de A. Ou seja A^-1 = B e também AB = I. A vantagem deste último método é que ele vale para qualquer dimensão de matriz. É este método que vai ser programado em Python a seguir.

Para que serve a matriz inversa ?

Inúmeras aplicações, mas vamos focar aqui na solução de sistemas lineares. Dado o sistema Ax = B, a matriz A é a dos coeficientes. Se você calcular a inversa A^-1, poderá resolver inúmeras instâncias do mesmo sistema, sem outra preocupação, já que A^-1 x B = x. Vamos ver um exemplo do que se diz aqui. Seja um sistema, dado por { 3x+2y+z = / x-y+z = / 5x+z = }

Escrito na forma matricial, o sistema fica Ax = B, onde A é a matriz (3 2 1 / 1 -1 1 / 5 0 1) Calculando a matriz inversa desta, obtém-se: (-0.1 -0.2 0.3 / 0.4 -0.2 -0.2 / 0.5 1 -0.5)

Agora, a solução para a instância onde B = {10, 2, 8} é obtida fazendo-se A^-1 x B que dá os valores x = 1, y = 2, z = 3. Para outra instância cujo B seja {30, -1, 26}, ao fazer a mesma multiplicação, obtém-se x = 5, y = 7, z = 1 e assim por diante.

Este exercício pede que você implemente o algoritmo de Gauss-Jordan para cálculo da matriz inversa e depois o aplique a uma matriz de ordem suficientemente alta para desencorajar o cálculo manual da mesma. Relembrando as etapas, você deve:

- 1. Escrever a matriz A e ao seu lado a matriz I.
- 2. Aplicar operações elementares com linhas a A visando transformá-la em I.
- 3. Aplicar rigorosamente as mesmas operações sobre a matriz ao lado.
- 4. Quando A tiver se transformado em I, a matriz que era originalmente I se transformou em A^-1.
- 5. Para conferir, faça a multiplicação matricial entre A e A^-1. Tem que dar I.

Vamos aplicar este processo ao sistema acima descrito. O objetivo é produzir um zero no coeficiente do x na segunda equação: Para isso: a[1][] = a[1][] + a[0][] x (-a[1][0]/a[0][0]) onde a notação a[1][] deve ser entendida como a aplicação da operação à toda a linha [1] da matriz a. Mas, atenção: isto não vale no Python. Lá, tem que ser por extenso.

Note-se também que o fator (-a[1][0]/a[0][0]) é o mesmo para toda a linha e portanto deve ser calculado antes e preservado durante todo o ciclo. O algoritmo vai chamar este fator de pivot e ele vai ser previamente calculado e guardado.

para calcular o zero no coeficiente de x na terceira linha, o comando é a[2][] = a[2][] + a[0][] x (-a[2][0]/a[0][0]).

Para a segunda coluna, começa-se achando o zero na primeira linha. O comando agora é a[0][] = a[0][] + a[1][] x (-a[0][1]/a[1][1]) e depois a[2][] = a[2][] + a[1][] x (-a[2][1]/a[1][1]). A terceira coluna é a[0][] = a[0][] + a[2][] x (-a[0][2]/a[2][2]) e a[1][] = a[1][] + a[2][] x (-a[1][2]/a[2][2]). Com estas operações, os zeros foram alcançados. Agora basta transformar os elementos da diagonal

principal na unidade. As operações são: a[0][] = a[0][]/a[0][0], a[1][] = a[1][]/a[1][1] e a[2][] = a[2][]/a[2][2]. Note que não houve preocupação em determinar se a matriz A é inversível. Isto foi deixado para não complicar demais o algoritmo. Fica como desafio para os mais corajosos. Eis o algoritmo:

```
import numpy as np
def cami(x):
    # x e a matriz de ordem ta de quem
    # se quer obter a inversa
    ta=len(x[0])
    y=np.identity(ta,float) #tem 0, na dp tem 1
    i=0
    while(i<ta):
        j=0
        while(j<ta):
            if (j!=i):
                pivot=x[j][i]/x[i][i]
                k=0
                while k<ta:
                    x[j][k]=x[j][k]+x[i][k]*pivot
                    y[j][k]=y[j][k]+y[i][k]*pivot
                    k=k+1
                j=j+1
            alvo=x[i][i]
            k=0
            while k<ta:
                y[i][k]=y[i][k]/alvo
                x[i][k]=x[i][k]/alvo
                k=k+1
            i=i+1
        return y;
a=np.array([[2.1, 3.2],[4.3,5.4]],float)
b=cami(a)
print(b)
```

Um exemplo Para você testar seu algoritmo, use-o nesta matriz:

59	57	44	42	65	23	47	66	78	71
94	36	83	17	12	86	4	92	43	95
12	61	18	10	94	45	94	81	30	76
9	49	92	22	90	22	27	6	21	26
31	45	96	26	26	75	59	39	47	98
33	27	9	94	29	5	49	68	9	33
61	81	17	18	61	74	16	37	34	45
64	85	46	74	49	81	69	20	18	59
17	54	66	65	94	54	76	54	30	76
68	91	61	89	6	57	35	92	52	80

Aqui a resposta (sexta linha, sétima coluna, vezes 1000) deverá ser 151.989, deverá ser 151.989, já devidamente multiplicada por 1000.

Para você fazer

A seguir uma matriz A de dimensão 10 x 10 (o que corresponderia a um sistema linear de 10 equações e 10 incógnitas). Você deve calcular a matriz inversa, seguindo o algoritmo acima, e depois responder qual o valor que está localizado na sexta linha e sétima coluna de A^-1, (em Python: a[5][6]) multiplicado por 1000. Eis a matriz:

18	90	50	73	47	74	35	90	65	44
18	31	34	36	73	60	29	95	87	33
28	21	59	90	56	53	80	33	71	46
35	27	24	66	9	20	47	43	88	84
6	9	98	77	80	31	98	73	46	84
27	63	87	1	90	47	58	74	31	31
45	69	84	62	5	31	8	91	17	95
12	82	27	54	79	79	72	42	59	11
30	72	11	24	60	26	96	84	61	45
69	3	73	63	34	66	35	87	67	78

Responda aqui:

A^-1[5][6] x 1000 =



Matriz Inversa

A definição matemática da matriz inversa é

A.A^{-1} = I

Ou seja, ao multiplicar uma matriz pela sua inversa, obtem-se a matriz identidade I de mesma ordem. Nesse sentido a matriz inversa joga o mesmo papel do inverso no produto comum: Dado um número, digamos 7, seu inverso (1/7) é aquele que quando multiplicado pelo número original reproduz a unidade: 7 x 1/7 = 1. Uma matriz quadrada pode não ser inversível, isto é não ter inversa. (Como o zero não tem inverso). Ela será não inversível quando seu determinante for igual a zero.

Propriedades

- 1. só existe A^{-1} se det(A) != 0
- 2. A inversa da inversa é a própria matriz (A^{-1})^{-1} = A
- 3. (n . A)^{-1} = n^{-1} . A^{-1}
- 4. (A_1 A_2 A_3 ... A_n)^{-1} = A_n^{-1} ... A_3^{-1} A_2^{-1} A_1^{-1}
- 5. (A^t)^{-1} = (A^{-1})^t
- 6. A.A^{-1} = A^{-1}.A = I (em geral A.B != B.A)
- 7. I^{-1} = I
- 8. (A.B)^{-1} = B^{-1}.A^{-1}

Formas de Calcular

Usando a teoria Usando a teoria, dada uma matriz como por exemplo (1 2 / 5 9)

Uma possibilidade é querer encontrar escalares x, y, z e w para os quais (1 2 / 5 9) (x y / z w) = (1 0 / 0 1) ou (x + 2z 5y + 9w / x + 2z 5y + 9w) = (1 0 / 0 1) ou ainda equações que satisfaçam { x + 2z = 1 / 5x + 9z = 0 } e { y + 2w = 0 / 5y + 9w = 1 } Resolvendo estas equações, acha-se a matriz inversa que é (-9 2 / 5 -1).

Usando determinantes Outros métodos (extraídos do excelente site http://mathworld.wolfram.com) vão a seguir descritos: Dada uma matriz 2 x 2 representada por

A = | a b / c d |, a inversa é dada por A^{-1} = 1/|A| | d -b / -c a | = 1/(ad-bc) | d -b / -c a |.

Para uma matriz 3 x 3 veja lá no site, que o espaço aqui é muito pequeno.

Usando Gauss-Jordan Neste método escreve-se a matriz da qual se quer achar a inversa ao lado esquerdo da matriz identidade, formando este arranjo (matriz escalonada reduzida por linhas).

[AI] = | a11 ... a1n | 1 0 ... 0 / | a21 ... a2n | 0 1 ... 0 / | ... | ... / | an1 ... ann | 0 0 ... 1 | Aplicam-se agora a este arranjo as técnicas de eliminação de Gauss, sobre o conjunto todo, de maneira a ficar com o seguinte arranjo | 1 0 ... 0 | b11 ... b1n / | 0 1 ... 0 | b21 ... b2n / | ... | ... / | 0 0 ... 1 | bn1 ... bnn |

Neste ponto, a matriz (bij) é a matriz inversa de A. Ou seja A^{-1} = B e também AB = I.

Usando cofatores Veja em /p/n/159/matinv.pdf

Usando LU Relembrando a decomposição LU, para um exemplo já feito em exercício passado:

A = L.U = (3 4 -9 / 2 1 2 / 7 -1 1) = (1 0 0 / 0.666 1 0 / 2.333 6.2 1) . (3 4 -9 / 0 -1.666 8 / 0 0 -27.6)

A inversa pode ser calculada coluna a coluna, fazendo-se a multiplicação de L por um vetor coluna {d1, d2, d3} obtendo como resultado a matriz coluna {1, 0, 0}. Depois, seguindo-se o método, multiplica-se a incógnita por U obtendo este resultado (Lembre-se do método LU). O resultado final deste sistema será a primeira coluna da matriz inversa procurada. Acompanhe: L.{d1, d2, d3} = {1, -0.6666, 1.79999} Agora, deve-se calcular U.x = d o que resulta {0.02173913, 0.08695652, -0.06521739} que é a primeira coluna da matriz inversa procurada.

A segunda coluna é obtida fazendo-se L.d = {0, 1, 0} e fica d = {0, 1, -6.2} Agora U.x = d e fica x = {0.03620637, 0.47828000, 0.22463768} que é a segunda coluna da matriz inversa. Finalmente, a terceira coluna L.d = {0, 0, 1} e fica d = {0.12319768, -0.17392000, -0.03623188} e a matriz inversa é

array([[0.02173913, 0.03623188, 0.12318841], [0.08695652, 0.47826087, -0.17391304], [-0.06521739, 0.22463768, -0.03623188]])

o que pode ser confirmado multiplicando-se esta matriz por A o que resultará na matriz identidade.

Usando pacotes computacionais

APL Dada a matriz A, sua inversa é facilmente calculável. Acompanhe

A←3 3p3 4 9 2 1 2 7 1 1 / MA 0.02173913043 0.03623188406 0.1231884058 / 0.08695652174 0.4782608696 -0.1739130435 / -0.0652173913 0.2246376812 -0.03623188406

MATLAB A função é Y=inv(X). Acompanhe

X=[1 0 2; -1 5 0; 0 3 -9] Y=inv(X) Y 0.8824 -0.1176 0.1961 / 0.1765 0.1765 0.0392 / 0.0588 0.0588 -0.0980

Python-Numpy O pacote Numpy é o de computação numérica no Python.

>>> import numpy as np >>> A=np.array([[1,0,2],[-1,5,0],[0,3,-9]]) >>> A array([[1, 0, 2], [-1, 5, 0], [0, 3, -9]]) >>> B=np.linalg.inv(A) >>> B array([[0.88235294, -0.11764706, 0.19607843], [0.17647059, 0.17647059, 0.03921569], [0.05882353, 0.05882353, -0.09803922]])

Python-Sympy O pacote Sympy é o de computação simbólica.

>>> import sympy as sp >>> a=sp.Matrix([[1,0,2],[-1,5,0],[0,3,-9]]) >>> a Matrix([[1, 0, 2], [-1, 5, 0], [0, 3, -9]]) >>> a.inv() Matrix([[15/17, -2/17, 10/51], [3/17, 3/17, 2/51], [1/17, 1/17, -5/51]])

>>> a.inv().n() Matrix([[0.882352941176471, -0.1176470588235294, 0.196078431372549], [0.176470588235294, 0.176470588235294, 0.0392156862745098], [0.0588235294117647, 0.0588235294117647, -0.0980392156862745]])

Usos 1. Ajuda a resolver sistemas lineares de maneira bastante elegante. De fato, a maioria dos sistemas lineares reais têm uma matriz A mais ou menos fixa (já que ela reflete as interdependências das variáveis na obtenção do resultado numérico final) e uma matriz B que varia de caso a caso. Nos métodos anteriores, (Gauss, jacobi, LU...) este fato era desconsiderado e a cada nova instância do sistema tudo era recalculado, incluindo-se aí a matriz A.

Agora, A gera a sua inversa A^{-1} uma única vez. E daí, a cada novo B apenas este é recalculado, usando-se o seguinte desenvolvimento: A.x = B, multiplicando-se ambos os lados por A^{-1} fica A^{-1}.A.x = A^{-1}.B e lembrando que A.A^{-1} = I, fica x = A^{-1}.B

2. Em um sistema linear, informa as respostas a determinados estímulos, ou seja funciona para entender as inter-relações de partes componentes. Sempre que um sistema linear for derivado de uma lei de conservação (massa, força, calor, momento, carga, quantidade de bits, ...). Neste caso, uma única equação pode ser escrita em cada ponto do sistema, resultando em um conjunto de equações que que define as propriedades do sistema. Em A.x = B, x são os níveis da propriedade sendo balanceada em cada parte do sistema. B contém os elementos do balanço que são independentes do comportamento do sistema - são constantes. Em muitos casos, são os estímulos externos. A matriz A contém os parâmetros que expressam como as partes do sistema interagem ou são acopladas. Como se viu, a solução de A.x = B pode ser escrita como x = A^{-1}.B. Da multiplicação de matrizes, pode-se dizer: x_1 = a_{11}^{-1}b_1 + a_{12}^{-1}b_2 + a_{13}^{-1}b_3 (com formulações semelhantes para os demais x_i). Daqui vê-se que a matriz inversa, além de fornecer uma solução, mostra a resposta de uma parte do sistema a um estímulo unitário de qualquer outra parte do sistema. A conclusão é que a_{ij}^{-1} representa o valor de x_i causado por uma quantidade unitária de b_j.

Para você fazer

1. A seguir, uma matriz A de ordem 4:

99	26	10	58
96	51	84	9
95	45	75	70
91	68	42	84

Agora, uma possível matriz inversa de A, A^{-1}:

0.01554	-0.00485	0.00615	-0.00735
-0.01119	-0.02622	0.01052	0.02844
-0.01071	0.01968	0.00012	-0.00902
-0.00243	0.01664	-0.01524	0.00135

A pergunta: A e A^{-1} são realmente inversas? Responda 1 se sim e 0 senão.

2. Dada a matriz M também de ordem 4, calcule a sua matriz inversa M^{-1} e responda qual o terceiro valor da terceira linha da matriz inversa, MULTIPLICADO POR 1000, com 3 casas decimais.

80	38	68	44
89	60	12	67
13	75	97	23
78	70	21	38

3. Abaixo, a matriz inversa P^{-1}.

0.00675	0.02245	-0.02644	0.00213
-0.00810	0.01629	-0.00064	0.00025
0.00929	-0.01015	0.01793	-0.00965
-0.00322	-0.01096	-0.00033	0.01585

E também os valores dos termos independentes R obtidos em P.x = R.

18202 14270 10764 15680

Ache os 4 valores (INTEIROS) de {x} neste sistema e responda aqui

0/1	13;c3	x	y	z	t



Números Primos

Numa manhã de agosto de 1900, David Hilbert da Universidade de Gottingen foi fazer sua aguardada palestra no Congresso Internacional de Matemáticos na Sorbonne em Paris. Ele era um dos maiores matemáticos do mundo e havia preparado uma palestra ousada. Ao invés de falar do passado ou do presente, resolveu falar do futuro. Mostraria o desconhecido e não o que já fora provado. Para anunciar o novo século desafiou a platéia com 23 problemas que segundo ele ditariam o rumo da *rainha das ciências* durante o século XX. Durante as décadas seguintes encontraram-se as respostas para muitos desses problemas. A propósito, a resposta ao décimo problema de Hilbert: *haverá um algoritmo capaz de decidir se uma equação diofantina tem ou não solução*, solucionado de forma independente por Alonso Church (cálculo λ) e por Alan Turing (Máquina Universal de Turing) ambos em 1936. Tangencialmente este último trabalho deu origem ao computador.

Voltando aos problemas de Hilbert, o oitavo, pedia a comprovação da Hipótese de Riemann, mas para poder trabalhar nela, precisa-se de alguma preparação.

Número Primo

Define-se um inteiro primo como aquele número que só tem 2 divisores inteiros: a unidade e o próprio número. Por exemplo, 5 é primo (divisores: 1 e 5) enquanto 6 não é primo, também chamado composto, (já que seus divisores são 1, 2, 3 e 6). Algumas curiosidades:

- O número 1 não é nem primo nem composto.
- 2 é o único par que também é primo. Todos os demais pares são divisíveis também por 2, logo compostos.
- O universo de divisores de n sempre existe em pares, cujo produto é igual ao número n .
- Se n é quadrado perfeito, sua raiz deve ser contada duas vezes para formar o par. Por exemplo, os divisores de 100 são: 1, 2, 4, 5, 10, 20, 25, 50 e 100. Neste caso, o 10 deve ser contado duas vezes.

Serão infinitos os primos ?

Eis uma demonstração do fato, devida a Euclides (há várias outras demonstrações). Quer-se provar o Teorema que diz que os primos são infinitos. A demonstração é por *redução ao absurdo*. Então, suponhamos que existem r números primos apenas, a saber: $p_1, p_2, \dots, p_r$. Façamos o conjunto $P = \{p_1, p_2, \dots, p_r\}$. Imagine o número N , definido por $N = p_1 \times p_2 \times \dots \times p_r$, obviamente composto. Agora, faça $M = N + 1$. De cara, M não é divisível por nenhum $p \in P$ pois o resto desta divisão é sempre 1. Agora vamos especular sobre a natureza de M . Ele pode ser primo ou composto. Se for primo, achamos uma contradição (o “último” primo era r) e está demonstrado o teorema original. Se for composto, então existe um primo p que não pertence ao conjunto P , pois se pertencesse, ele dividiria N e também M e ao fazê-lo, deveria também dividir a diferença $M - N$ que como se sabe vale 1. Mas, não existe um primo que divida 1. Então, achou-se um novo primo p que não pertence a P , uma contradição o que demonstra o teorema original.

Distribuição dos números primos

Os primos são infinitos, embora eles venham se tornando mais raros à medida em que cresce o limite superior da contagem. Por exemplo, entre 1 e 10 há 4 primos (40%), entre 1 e 100, há 25 (25%), entre 1 e 1000, há 168 (16%), entre 1 e 10.000, há 1.229 (12%), entre 1 e 100.000 há 9.529 (9%) e assim por diante. A primeira descoberta na tentativa de entender a distribuição dos primos veio de Gauss,

quando ele afirmou que o número de primos entre 1 e N é aproximadamente igual a

$$\frac{N}{\log_e(N)}$$

O número real é um pouco menor do que isso. A próxima etapa é o surgimento da função zeta, originalmente proposta nos tempos de Euler, cuja definição é

$$\zeta(x) = \frac{1}{1^x} + \frac{1}{2^x} + \dots \frac{1}{n^x} + \dots$$

A origem do interesse dos matemáticos por esta soma infinita veio da música e tem origem numa descoberta dos gregos (Pitágoras) associando as oitavas da escala musical aos números 1, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$,...

Os matemáticos sabem há muito tempo que ao substituir $x = 1$ na função ζ , o resultado da soma, embora converja muito lentamente, tende a ∞ . Números menores que 1, sempre convergem, eventualmente mais rapidamente, para infinito.

Euler, um dos maiores (senão o maior) torturadores de números da história resolveu estudar o que aconteceria se $x = 2$ na função ζ . A função fica

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \dots$$

e agora este número é menor, pois não inclui todas as parcelas de antes. A melhor estimativa era de $\frac{8}{5}$, mas Euler, *num dos cálculos mais intrigantes de toda a matemática* (Sato, 2004) chegou a

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \dots = \frac{1}{6} \pi^2$$

Depois disso, Euler lembrou da descoberta grega de que qualquer número pode ser expresso como um produto de primos, e reescreveu ζ de outra maneira. Antes, veja-se a base do raciocínio $60 = 2 \times 2 \times 3 \times 5$ e daqui $\frac{1}{60} = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} \times \frac{1}{5} = (\frac{1}{2})^2 \times \frac{1}{3} \times \frac{1}{5}$ e agora

$$\zeta(x) = (1 + \frac{1}{2^x} + \frac{1}{4^x} + \dots) \times (1 + \frac{1}{3^x} + \frac{1}{9^x} + \dots) \times \dots \times (1 + \frac{1}{p^x} + \frac{1}{(p^2)^x} + \dots) \times \dots$$

Os gregos já haviam provado há 2000 anos que os números primos eram infinitos, mas a expressão acima apresentava nova prova do mesmo fato. Estamos por volta de 1850 em Gottingen, quando Dirichlet foi trabalhar lá, entusiasmado com a função ζ . Lá já estava um aluno de Gauss, Riemann por sua vez entusiasmado com os números complexos de Cauchy. O pulo do gato agora foi dado por Riemann que foi introduzir números imaginários na função ζ . Recém eleito para a Academia de Berlim, Riemann aproveitou o costume (novos associados deveriam apresentar um relato de suas pesquisas recentes) e em novembro de 1859 ele apresentou um artigo de 10 páginas sobre os números primos. Aqui Riemann herdou de seu professor Gauss (*pauca sed madura*) o costume de ao apresentar um bela construção retirar antes todos os andaimes e ferramentas usadas na construção. Quase perdido no texto está declarado o problema cuja solução hoje possui uma etiqueta de 1.000.000 de dólares: a hipótese de Riemann. Junto uma observação: *É claro que gostaríamos de ter uma prova rigorosa, mas deixei de lado essa busca após breves tentativas fracassadas, pois ela não é necessária para o objetivo imediato de minha investigação*. O principal objetivo do artigo em Berlim era confirmar que a função de Gauss fornecia uma aproximação cada vez melhor do número de primos à medida em que se atinge contagens cada vez mais elevadas. A recompensa por este artigo foi a cadeira universitária que Gauss havia ocupado e que a morte de Dirichlet deixara vaga.

Baseado no trabalho de Cauchy (funções podem ser representadas por equações) e estas desenhadas em um gráfico ele estudou o gráfico da função ζ . O problema aqui é que os reais usam um eixo e a função o outro. Ao passar para complexos (que exigem 2 eixos para o número) são necessários 4 eixos: 2 para o número e mais dois para a função. A chave para pensar neste mundo é pensar na sombra: ela é a imagem bidimensional de algo que tem 3 dimensões. Em alguns ângulos ela fornece informação pobre, mas em outros isso muda: ao olhar o perfil de alguém eventualmente pode-se reconhecê-lo.

Hipótese de Riemann

No artigo de 1859, Riemann obteve a fórmula (exata) para $\pi(x)$, na qual aparecem 2 parcelas: primeiro uma excelente estimativa para $\pi(x)$ e depois uma subtração do erro cometido através de uma parcela de correção. Esta correção exige saber onde estão os zeros da equação ζ . Há dois tipos de zeros: os triviais e os não triviais. Se $\Re(s) > 1$ então $\zeta(x) \neq 0$, isto é não há raízes nessa porção do plano complexo. Se $\Re(s) < 0$, a própria equação dá os zeros triviais. A afirmação de Riemann, feita em 1859 e que hoje vale 1 milhão de dólares diz que todos os zeros da função ζ se encontram sobre a reta $\Re(s) = \frac{1}{2}$. A validade deste resultado implica que não há surpresas no comportamento da sequência de primos, isto é que os primos têm padrão de regularidade e que não há surpresas lá adiante, onde a vista não alcança. Uma consequência prática importante deste estudo é a garantia de integridade do método RSA de criptografia na Internet.

Função Möebius É uma função cuja imagem comporta apenas 3 valores: 0, -1 e 1, a depender da seguinte regra. $\mu(n) = 0$ se n tem como divisor outro natural ao quadrado. Senão, $\mu(n) = 1$ se n é decomposto em uma quantidade par de primos e $\mu(n) = -1$ se n é decomposto em quantidade ímpar de primos.

Função Totiente de Euler É uma função, denotada $\phi(n)$ que conta a quantidade de números x , tais que $x \leq n$ e que são co-primos em relação a n (Ou seja, o $\text{mdc}(x, n) = 1$). Conhecida também como função totiente.

Exemplo

- Seja o seguinte enunciado
1. Ache o maior fator primo de 1.894.383 (R=210.487).
 2. Ache o k-ésimo número primo. k=137.255 (R=1.831.187).
 3. Ache o primo anterior a 1.618.439 (R=1.618.433).
 4. Teste de primalidade. Informe 0=não e 1=sim se 1.431.559 for primo (R=0).
 5. Quantos primos p há, tal que $p \leq 947.085$? (R=74.702).
 6. Ache a função contador de divisores de 18.387 (R=10).
 7. Ache a função soma de divisores de 14.396 (R=26.040).
 8. Ache a função de Möebius de 10.616 (R=0).
 9. Ache a função totiente de Euler de 14.971 (R=13.600).

🔗 Para você fazer

1. Ache o maior fator primo de 1.844.360	
2. Ache o k-ésimo número primo. k= 114.258	
3. Ache o primo anterior a 1.974.500	
4. Teste de primalidade: Informe 0=não e 1=sim se 1.889.101 for primo	
5. Quantos primos p há, tal que $p \leq 972.073$?	
6. Ache a função contador de divisores de 19.144	
7. Ache a função soma de divisores de 11.163	
8. Ache a função de Möebius de 15.465	
9. Ache a função totiente de Euler de 10.430	



Otimização de algoritmos em C++

Seja o problema 12 do magnífico site de matemática e programação projecteuler.net. Ele diz: A sequência dos números triangulares é gerada pela adição dos números naturais. Assim, o 7º número triangular será 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. Os primeiros 10 números triangulares são

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Vai-se listar aqui os divisores dos primeiros sete números triangulares:

1: 1
3: 1,3
6: 1,2,3,6
10: 1,2,5,10
15: 1,3,5,15
21: 1,3,7,21
28: 1,2,4,7,14,28

Pode-se ver que 28 é o primeiro número triangular que tem mais do que 5 divisores. Qual é o valor do primeiro número triangular que tem mais do que 500 divisores ?

Primeira estratégia

Sem pensar muito, pode-se resolver a questão usando a força bruta. Aqui, geram-se os números triangulares como descrito e depois acham-se todos seus divisores. A busca acaba quando se acha o primeiro com mais do que 500 divisores. (Uma informação: inteiros de 32 bits são suficientes para este problema). Uma solução em C++:

```
#include<iostream>
using namespace std;
int main() {
    int t,a,cnt,i;
    t=1;
    a=1;
    cnt=0;
    while (cnt<=500){
        cnt=0;
        a=a+1;
        t=t+a;
        i=1;
        while (i<=t){
            if ((t%i)==0){
                cnt++;
            }
            i++;
        }
        cout<<t;
    }
}
```

A implementação acima é muito demorada (17.75 minutos em uma CPU média, com 2 processadores).

Segunda estratégia

Uma melhora evidente é interromper a busca quando o divisor alcança a raiz quadrada do número triangular: Para cada divisor abaixo da raiz quadrada existe um acima dela. O código fica:

```
#include<iostream>
#include<cmath>
using namespace std;
int main() {
    int t,a,cnt,i,ttx;
    t=1;
    a=1;
    cnt=0;
    while (cnt<=500){
        cnt=0;
        a=a+1;
        t=t+a;
        i=1;
        ttx=sqrt(t);
        while (i<=ttx){
            if ((t%i)==0){
                cnt=cnt+2;
            }
            if (t==(ttx*ttx)){
                cnt=cnt-1; // correcao quadrado perfeito
            }
            i++;
        }
        cout<<t;
    }
}
```

Ainda assim, esta segunda estratégia continua demorada.

Terceira estratégia

Vamos nos socorrer de um pouco de matemática. Sabe-se que qualquer inteiro *N* pode ser expresso de maneira única como

$$N = p_1^{a_1} \times p_2^{a_2} \times p_3^{a_3} \times \dots$$

onde *p_i* é um número primo e *a_n* seu expoente. Por exemplo, 28 = 2² × 7¹. Daqui, o número de divisores de *N*, *D(N)* de qualquer inteiro pode ser computado como

$$D(N) = (a_1 + 1) \times (a_2 + 1) \times \dots$$

onde os *a_n* são os expoentes dos fatores primos que compõe o número *N*. No exemplo, os divisores de 28 são *D*(28) = (2 + 1) × (1 + 1) = 3 × 2 = 6. Uma tabela de primos é necessária para aplicar esta relação. Uma estratégia para criá-la pode ser (até 65500):

```
#include<iostream>
#include<cmath>
#include<fstream>
#include <stdlib.h>
using namespace std;
int pri[7000];
int primo(int n){
    int f,i,r,j;
    if (n==1){
        return 0;
    }
    if (n<4){
        return 1;
    }
    if (0==(n%2)){
        return 0;
    }
    if (n<9){
        return 1;
    }
    if (0==(n%3)){
        return 0;
    }
    r=floor(sqrt(n));
    f=5;
    while (f<=r){
        if ((n%f)==0){
            return 0;
        }
        if ((n%(f+2))==0){
            return 0;
        }
        f=f+6;
    }
    return 1;
}
int criatabprimos(){
    int i,j;
    pri[0]=2;
    i=1;
    j=3;
    while (j<65500){
        if (primo(j)==1){
            pri[i]=j;
            i=i+1;
        }
        j=j+2;
    }
    ofstream outfile;
    outfile.open("c:/p/n/162/primos.dat");
    j=0;
    while (j<i){
        outfile<<pri[j]<<endl;
        j=j+1;
    }
    outfile.close();
}
int main() {
    int t,a,cnt,tt,i,expoente;
    string valor;
    // criatabprimos();
    ifstream infile;
    infile.open("c:/p/n/162/primos.dat");
    i=0;
    while (!infile.eof()){
        getline(infile,valor);
        pri[i]=atoi(valor.c_str());
        i=i+1;
    }
    infile.close();
    t=1;
    a=1;
    cnt=1;
    while (cnt<=500){
        cnt=1;
        a=a+1;
        t=t+a;
        tt=t;
        i=0;
        while (i<6540){
            if (pri[i]*pri[i]>tt){
                cnt=cnt*2;
                break;
            }
            expoente=1;
            while ((tt%pri[i])==0){
                expoente=expoente+1;
                tt=tt/pri[i];
            }
            if (expoente>1){
                cnt=cnt*expoente;
            }
            if (tt==1){
                break;
            }
            i=i+1;
        }
        cout<<t;
    }
}
```

Quarta estratégia

Ainda se pode melhorar mais, sabendo que os números triangulares também podem ser obtidos de acordo com

$$t = \frac{n \times (n + 1)}{2}$$

onde os componentes *n* e *n* + 1 são necessariamente co-primos (isto é, não têm qualquer fator primo comum nem divisor comum). Aqui, o número de divisores *D(t)* pode ser obtido

$$D(t) = D(n/2) \times D(n + 1) \text{ se } n \text{ par}$$

ou

$$D(t) = D(n) \times D((n + 1)/2) \text{ se } n + 1 \text{ par}$$

Cada componente é muito menor do que o número triangular. A tabela de primos também é muito menor (contém apenas primos menores do que 1000). Mais, o resultado do componente *n* + 1 pode ser reaproveitado como *n* no próximo número triangular, sem ser necessário recalculá-lo. Fica:

```
#include<iostream>
#include<stdlib.h>
#include<fstream>
using namespace std;
int pri[168];
int main(){
    int i,n,Dn,xnt,Dn1,n1,expoente,cnt;
    string valor;
    ifstream infile;
    infile.open("c:/p/n/162/primos.dat");
    i=0;
    while (i<169){
        getline(infile,valor);
        pri[i]=atoi(valor.c_str());
        i=i+1;
    }
    infile.close();
    n=3;
    Dn=2;
    cnt=0;
    while (cnt<=500){
        n=n+1;
        n1=n;
        if ((n1%2)==0) {
            n1=n1/2;
        }
        Dn1=1;
        i=0;
        while (i<169){
            if ((pri[i]*pri[i])>n1){
                Dn1=2*Dn1;
                break;
            }
            expoente=1;
            while ((n1%pri[i])==0){
                expoente=expoente+1;
                n1=n1/pri[i];
            }
            if (expoente>1){
                Dn1=Dn1*expoente;
            }
            if (n1==1){
                break;
            }
            i++;
        }
        cnt=Dn*Dn1;
        Dn=Dn1;
    }
    cout<<((n*(n-1))/2)<<endl;
}
```

Esta implementação demorou perto de 1/10 de segundo na mesma CPU lá do começo. (0.152 seg para ser exato). Ou seja, de 1065 segundos para 0.152, ou uma melhora de cerca de 7000 vezes. Veja-se como vale a pena estudar matemática e algoritmos.

👉 Para você fazer

O objetivo deste exercício é treinar a implementação de algoritmos em C++ e também meditar sobre como modificações na implementação de um mesmo problema a partir de conhecimentos matemáticos (Santa Matemática !) podem reduzir radicalmente os tempos de processamento.

Para ajudar nesta meditação, implemente a alternativa que quiser e informe qual o número inteiro triangular que tem mais do que

195

divisores. Informe o número achado abaixo:



Otimização de algoritmos em C++

Seja o problema 12 do magnífico site de matemática e programação projecteuler.net. Ele diz: A sequência dos números triangulares é gerada pela adição dos números naturais. Assim, o 7º número triangular será 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. Os primeiros 10 números triangulares são

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Vai-se listar aqui os divisores dos primeiros sete números triangulares:

1: 1
3: 1,3
6: 1,2,3,6
10: 1,2,5,10
15: 1,3,5,15
21: 1,3,7,21
28: 1,2,4,7,14,28

Pode-se ver que 28 é o primeiro número triangular que tem mais do que 5 divisores. Qual é o valor do primeiro número triangular que tem mais do que 500 divisores ?

Primeira estratégia

Sem pensar muito, pode-se resolver a questão usando a força bruta. Aqui, geram-se os números triangulares como descrito e depois acham-se todos seus divisores. A busca acaba quando se acha o primeiro com mais do que 500 divisores. (Uma informação: inteiros de 32 bits são suficientes para este problema). Uma solução em C++:

```
#include<iostream>
using namespace std;
int main() {
    int t,a,cnt,i;
    t=1;
    a=1;
    cnt=0;
    while (cnt<=500){
        cnt=0;
        a=a+1;
        t=t+a;
        i=1;
        while (i<=t){
            if ((t%i)==0){
                cnt++;
            }
            i++;
        }
        cout<<t;
    }
}
```

A implementação acima é muito demorada (17.75 minutos em uma CPU média, com 2 processadores).

Segunda estratégia

Uma melhora evidente é interromper a busca quando o divisor alcança a raiz quadrada do número triangular: Para cada divisor abaixo da raiz quadrada existe um acima dela. O código fica:

```
#include<iostream>
#include<cmath>
using namespace std;
int main() {
    int t,a,cnt,i,ttx;
    t=1;
    a=1;
    cnt=0;
    while (cnt<=500){
        cnt=0;
        a=a+1;
        t=t+a;
        i=1;
        ttx=sqrt(t);
        while (i<=ttx){
            if ((t%i)==0){
                cnt=cnt+2;
            }
            if (t==(ttx*ttx)){
                cnt=cnt-1; // correcao quadrado perfeito
            }
            i++;
        }
        cout<<t;
    }
}
```

Ainda assim, esta segunda estratégia continua demorada.

Terceira estratégia

Vamos nos socorrer de um pouco de matemática. Sabe-se que qualquer inteiro *N* pode ser expresso de maneira única como

$$N = p_1^{a_1} \times p_2^{a_2} \times p_3^{a_3} \times \dots$$

onde *p_i* é um número primo e *a_n* seu expoente. Por exemplo, 28 = 2² × 7¹. Daqui, o número de divisores de *N*, *D(N)* de qualquer inteiro pode ser computado como

$$D(N) = (a_1 + 1) \times (a_2 + 1) \times \dots$$

onde os *a_n* são os expoentes dos fatores primos que compõe o número *N*. No exemplo, os divisores de 28 são *D*(28) = (2 + 1) × (1 + 1) = 3 × 2 = 6. Uma tabela de primos é necessária para aplicar esta relação. Uma estratégia para criá-la pode ser (até 65500):

```
#include<iostream>
#include<cmath>
#include<fstream>
#include <stdlib.h>
using namespace std;
int pri[7000];
int primo(int n){
    int f,i,r,j;
    if (n==1){
        return 0;
    }
    if (n<4){
        return 1;
    }
    if (0==(n%2)){
        return 0;
    }
    if (n<9){
        return 1;
    }
    if (0==(n%3)){
        return 0;
    }
    r=floor(sqrt(n));
    f=5;
    while (f<=r){
        if ((n%f)==0){
            return 0;
        }
        if ((n%(f+2))==0){
            return 0;
        }
        f=f+6;
    }
    return 1;
}
int criatabprimos(){
    int i,j;
    pri[0]=2;
    i=1;
    j=3;
    while (j<65500){
        if (primo(j)==1){
            pri[i]=j;
            i=i+1;
        }
        j=j+2;
    }
    ofstream outfile;
    outfile.open("c:/p/n/162/primos.dat");
    j=0;
    while (j<i){
        outfile<<pri[j]<<endl;
        j=j+1;
    }
    outfile.close();
}
int main() {
    int t,a,cnt,tt,i,expoente;
    string valor;
    // criatabprimos();
    ifstream infile;
    infile.open("c:/p/n/162/primos.dat");
    i=0;
    while (!infile.eof()){
        getline(infile,valor);
        pri[i]=atoi(valor.c_str());
        i=i+1;
    }
    infile.close();
    t=1;
    a=1;
    cnt=1;
    while (cnt<=500){
        cnt=1;
        a=a+1;
        t=t+a;
        tt=t;
        i=0;
        while (i<6540){
            if (pri[i]*pri[i]>tt){
                cnt=cnt*2;
                break;
            }
            expoente=1;
            while ((tt%pri[i])==0){
                expoente=expoente+1;
                tt=tt/pri[i];
            }
            if (expoente>1){
                cnt=cnt*expoente;
            }
            if (tt==1){
                break;
            }
            i=i+1;
        }
        cout<<t;
    }
}
```

Quarta estratégia

Ainda se pode melhorar mais, sabendo que os números triangulares também podem ser obtidos de acordo com

$$t = \frac{n \times (n + 1)}{2}$$

onde os componentes *n* e *n* + 1 são necessariamente co-primos (isto é, não têm qualquer fator primo comum nem divisor comum). Aqui, o número de divisores *D(t)* pode ser obtido

$$D(t) = D(n/2) \times D(n + 1) \text{ se } n \text{ par}$$

ou

$$D(t) = D(n) \times D((n + 1)/2) \text{ se } n + 1 \text{ par}$$

Cada componente é muito menor do que o número triangular. A tabela de primos também é muito menor (contém apenas primos menores do que 1000). Mais, o resultado do componente *n* + 1 pode ser reaproveitado como *n* no próximo número triangular, sem ser necessário recalculá-lo. Fica:

```
#include<iostream>
#include<stdlib.h>
#include<fstream>
using namespace std;
int pri[168];
int main(){
    int i,n,Dn,xnt,Dn1,n1,expoente,cnt;
    string valor;
    ifstream infile;
    infile.open("c:/p/n/162/primos.dat");
    i=0;
    while (i<169){
        getline(infile,valor);
        pri[i]=atoi(valor.c_str());
        i=i+1;
    }
    infile.close();
    n=3;
    Dn=2;
    cnt=0;
    while (cnt<=500){
        n=n+1;
        n1=n;
        if ((n1%2)==0) {
            n1=n1/2;
        }
        Dn1=1;
        i=0;
        while (i<169){
            if ((pri[i]*pri[i])>n1){
                Dn1=2*Dn1;
                break;
            }
            expoente=1;
            while ((n1%pri[i])==0){
                expoente=expoente+1;
                n1=n1/pri[i];
            }
            if (expoente>1){
                Dn1=Dn1*expoente;
            }
            if (n1==1){
                break;
            }
            i++;
        }
        cnt=Dn*Dn1;
        Dn=Dn1;
    }
    cout<<((n*(n-1))/2)<<endl;
}
```

Esta implementação demorou perto de 1/10 de segundo na mesma CPU lá do começo. (0.152 seg para ser exato). Ou seja, de 1065 segundos para 0.152, ou uma melhora de cerca de 7000 vezes. Veja-se como vale a pena estudar matemática e algoritmos.

👉 Para você fazer

O objetivo deste exercício é treinar a implementação de algoritmos em C++ e também meditar sobre como modificações na implementação de um mesmo problema a partir de conhecimentos matemáticos (Santa Matemática !) podem reduzir radicalmente os tempos de processamento.

Para ajudar nesta meditação, implemente a alternativa que quiser e informe qual o número inteiro triangular que tem mais do que

195

divisores. Informe o número achado abaixo:



São equivalentes as condicionais?
(em C++)

Nos exercícios a seguir, são apresentados 2 trechos de código envolvendo comandos condicionais. Você deve examiná-los e determinar se eles são equivalentes (respondendo 1) ou se eles não são equivalentes, neste caso respondendo 0.

Lembre que 2 trechos de código são equivalentes se eles sempre produzem exatamente os mesmos resultados, independente dos valores atribuídos a todas e a cada variável envolvida.

 Para você fazer

A seguir, 10 casos de duplas de condições. Para cada caso, deve-se examinar a condição da esquerda e a da direita. Se elas forem equivalentes, responder 1, senão responder 0.

1.-----

```
if (D>6) {      | if ( not D>6){  
    I=R+7;}      |   {}}  
                  | else {  
                  |   I=R+7;}
```

2.-----

```
if (A>5) {      | if (A<=5) {  
    I=M*3;}      |   I=M*3;}  
else {          | else {  
    if (B<6) {   |   if (B<6) {  
        S=L/5;} |   S=L/5;}  
    }           |   }
```

3.-----

```
if (F==5) {      | if ( not F!=5) {  
    if (B>=2) {   |   P=I+5;}  
        K=S-3;}   |   else {  
    else {        |   if ( not B>=2){  
        if (C<=1){ |   if (C>1) {  
            L=J/2;} |   N=M+6;}  
        else {     |   else {  
            N=M+6;} |   L=J/2;}  
        }         |   }  
    }             |   else {  
else {           |   K=S-3;}  
    P=I+5;}       |   }
```

4.-----

```
if (B<1) { | if (B>=1){  
    Q=M+3;} |   Q=M+3;}
```

5.-----

```
if (B<4) {      | if ( not B<4){  
    M=L/4;}      |   M=L/4;}  
else {          | else {  
    if (F>6) {   |   if (F>6) {  
        M=P/8;} |   M=P/8;}  
    }           |   }
```

6.-----

```
if (C>=7) {      | if ( not C>=7) {  
    J=Q/5;}       |   J=Q/5;}  
else {           | else {  
    if (D!=6) {   |   if ( not D!=6){  
        if (A!=7){ |   Q=M-7;}  
            J=P*7;} |   else {  
        }         |   if (A!=7) {  
    else {       |   J=P*7;}  
        Q=M-7;}  |   }  
    }           |   }
```

7.-----

```
if (D!=5){ | if (D!=5){  
    I=Q-7;} |   I=Q-7;}  
else {     | else {  
    N=J*2;} |   N=J*2;}
```

8.-----

```
if (B==6) {      | if (B!=6) {  
    P=R/4;}       |   if ( not F<3){  
else {          |   L=P/8;}  
    if (F>=3){   |   }  
        L=P/8;}  |   else {  
    }           |   P=R/4;}
```

9.-----

```
if (D>=7) {      | if ( not D>=7) {  
    if (B>=2) {   |   if ( not B>=2) {  
        if (B>=1){ |   P=L*8;}  
            M=I*6;} |   else {  
        else {     |   if ( not B>=1){  
            K=R+6;} |   K=R+6;}  
        }         |   else {  
    else {       |   M=I*6;}  
        P=L*8;}  |   }  
    }           |   }  
else {          | else {  
    Q=S*3;}      |   Q=S*3;}
```

10.-----

```
if (D<7) {      | if (D<7) {  
    if (A>=7){   |   S=J/7;}  
        S=J-4;} |   else {  
    }           |   if (A>=7){  
else {          |   S=J-4;}  
    S=J/7;}     |   }
```

Responda aqui:				
1	2	3	4	5
6	7	8	9	10

Mais um

11.-----

```
if (C<=1){      | if ( not C<=1){  
    Q=J/6;}       |   P=K*2;}  
else {          | else {  
    P=K*2;}      |   Q=J/6;}
```

12.-----

```
if (B>=1) {      | if (B<1) {  
    if (D!=7){   |   I=J/4;}  
        Q=S+8;} |   else {  
    }           |   if (D!=7){  
else {          |   Q=S+8;}  
    I=J/4;}     |   }
```

13.-----

```
if (D>4) {      | if ( not D>4) {  
    K=R*6;}       |   if (B>=1) {  
else {          |   if (C<=2){  
    if (B<1) {   |   L=M/5;}  
        N=P/8;} |   }  
    else {       |   else {  
        if (C<=2){ |   N=P/8;}  
            L=M/5;} |   }  
    }           |   else {  
    }           |   K=R*6;}
```

14.-----

```
if (F>=4){ | if (F>=4){  
    S=J*2;} |   K=R-7;}  
else {     | else {  
    K=R-7;} |   S=J*2;}
```

15.-----

```
if (B>=5) {      | if ( not B<5){  
    N=Q-4;}       |   N=Q-4;}  
else {          | else {  
    if (A>=1){   |   if (A>=1){  
        R=S/2;}  |   R=S/2;}  
    }           |   }
```

16.-----

```
if (A>3) {      | if ( not A>3) {  
    N=Q-7;}      |   if (A<=7) {  
else {          |   M=K+4;}  
    if (A>7) {   |   else {  
        if (F>=5){ |   if ( not F<5){  
            K=J*5;} |   K=J*5;}  
        }         |   }  
    else {       |   }  
        M=K+4;}  |   else {  
    }           |   N=Q-7;}
```

17.-----

```
if (A<5) { | if (A<5) {  
    L=Q/4;} |   {}}  
            | else {  
            |   L=Q/4;}
```

18.-----

```
if (C==4) {      | if ( not C==4) {  
    if (C>=6){   |   N=K-2;}  
        Q=M-6;}  |   else {  
    }           |   if ( not C<6) {  
else {          |   {}}  
    N=K-2;}      |   else {  
                |   Q=M-6;}  
                |   }
```

19.-----

```
if (D<4) {      | if ( not D<4) {  
    R=I*3;}       |   if (D>=7) {  
else {          |   if (C<=2){  
    if (D<7) {   |   M=I/6;}  
        N=S+4;}  |   }  
    else {       |   else {  
        if (C<=2){ |   N=S+4;}  
            M=I/6;} |   }  
    }           |   else {  
    }           |   R=I*3;}
```

20.-----

```
if (B!=7) {      | if ( not B==7){  
    M=N-2;}       |   if (C<=3){  
else {          |   I=P+5;}  
    if (C<=3){   |   }  
        I=P+5;}  |   else {  
    }           |   M=N-2;}
```

11	12	13	14	15
16	17	18	19	20

São equivalentes as condicionais?
(em Python)

Nos exercícios a seguir, são apresentados 2 trechos de código envolvendo comandos condicionais. Você deve examiná-los e determinar se eles são equivalentes (respondendo 1) ou se eles não são equivalentes, neste caso respondendo 0.

Lembre que 2 trechos de código são equivalentes se eles sempre produzem exatamente os mesmos resultados, independente dos valores atribuídos a todas e a cada variável envolvida.

👉 Para você fazer

A seguir, 10 casos de duplas de condições. Para cada caso, deve-se examinar a condição da esquerda e a da direita. Se elas forem equivalentes, responder 1, senão responder 0.

1.-----

if C==4: | if C==4:
S=M/7 | S=M/7

2.-----

if B<=3: | if not B<=3:
L=R/5 | if A<3:
else: | I=P-3
if A>=3: | else:
Q=L-5 | Q=L-5
else: | else:
I=P-3 | L=R/5

3.-----

if B==3: | if not B==3:
R=P-6 | R=P-6
else: | else:
if A>3: | if A<=3:
N=S+6 | if C>=2:
else: | M=R-7
if C>=2: | else:
M=R-7 | Q=L-8
else: | else:
Q=L-8 | N=S+6

4.-----

if D!=2: | if D==2:
S=K*2 | S=N*6
else: | else:
S=N*6 | S=K*2

5.-----

if F>2: | if F<=2:
R=K*8 | if not A<=5:
else: | I=R*8
if A<=5: | else:
L=N/2 | L=N/2
else: | else:
I=R*8 | R=K*8

6.-----

if B<7: | if B>=7:
if A<5: | P=Q-5
P=L/4 | else:
else: | if not A>=5:
if C<3: | if C<3:
S=M*2 | S=M*2
else: | else:
R=K*5 | R=K*5
else: | else:
P=Q-5 | P=L/4

7.-----

if C==1: | if not C==1:
S=N*3 | S=N*3

8.-----

if D==2: | if not D==2:
I=K*3 | if C!=4:

else: | J=P-2
if C==4: | else:
J=P-2 | I=K*3

9.-----

if F>=2: | if F>=2:
I=P+6 | if D!=1:
else: | M=N-2
if D==1: | else:
if C>=4: | if not C>=4:
R=M*6 | S=K+2
else: | else:
S=K+2 | R=M*6
else: | else:
M=N-2 | I=P+6

10.-----

if A<7: | if A>=7:
if F==7: | if not F!=7:
I=S-6 | I=S-6
else: | else:
R=N+5 | R=N+5
else: | else:
N=I+6 | N=I+6

Responda aqui:

1	2	3	4	5
6	7	8	9	10

Mais um

11.-----

if B<=7: | if not B<=7:
N=I+3 | J=R+5
else: | else:
J=R+5 | N=I+3

12.-----

if C!=5: | if C==5:
P=J/8 | if B<=5:
else: | pass
if B>5: | else:
J=N/3 | J=N/3
else: | else:
P=J/8

13.-----

if D>=1: | if not D<1:
Q=I/4 | Q=I/4
else: | else:
if F>7: | if not F>7:
if F<=6: | K=L-3
S=P/2 | else:
else: | if F<=6:
K=L-3 | S=P/2

14.-----

if D<=5: | if D<=5:
L=Q/8 | pass
else: | else:
L=Q/8

15.-----

if F<=7: | if not F<=7:
J=N/2 | if not F!=3:
else: | K=S+6
if F!=3: | else:
N=L+7 | N=L+7
else: | else:
K=S+6 | J=N/2

16.-----

if D>7: | if not D>7:
K=S+5 | K=S+5
else: | else:
if C>=6: | if not C>=6:
if D>5: | L=I+6
K=L-2 | else:
else: | if D>5:
K=N/5 | K=L-2
else: | else:
L=I+6 | K=N/5

17.-----

if C>3: | if C>3:
L=M+5 | pass
else: | else:
L=M+5

18.-----

if A==4: | if A==4:
if B<=4: | M=Q/8
Q=P-5 | else:
else: | if not B<=4:
N=L+4 | N=L+4
else: | else:
M=Q/8 | Q=P-5

19.-----

if F<7: | if F>=7:
if A<3: | L=S-2
if B<1: | else:
P=Q-7 | if not A>=3:
else: | if B>=1:
L=I*4 | pass
else: | else:
L=S-2 | P=Q-7
else: | else:
L=I*4

20.-----

if B!=1: | if B==1:
K=L+2 | if C>4:
else: | R=Q/8
if C>4: | else:
R=Q/8 | K=L+2

11	12	13	14	15
16	17	18	19	20

Varejão I - Pérolas de programação

Ordenação de complexos Suponha a necessidade de estabelecer alguma ordenação a um conjunto de números complexos. Como se sabe os complexos são o resultado da necessidade de obter solução para equações tais como $x^2 + 1 = 0$. No campo dos reais, tal equação não tem solução, já que não existe número real que multiplicado por ele mesmo dê -1 . O jeito é criar um novo conjunto de números, os complexos ($\mathbb{C}$) nos quais um número agora é $a + bi$, onde a e b são reais e $i = \sqrt{-1}$. Com estes números a solução da equação $x^2 + 1 = 0$ é $\pm i$. Uma óbvia necessidade dos complexos é para atender ao Teorema Fundamental da Álgebra, que diz que *uma equação de grau n terá n raízes*.

Definidos os complexos como um superconjunto dos reais ($\mathbb{C} \subset \mathbb{R}$), toda a matemática se mantém íntegra. A dificuldade aqui é de outra ordem. Como se sabe, para ordenar um conjunto é vital estabelecer uma relação de ordem, por exemplo $x > y$, que nos reais é óbvia. Como se sabe $5 > 4$ é verdadeiro e $2 > 3$ é falso. Aqui nos complexos a coisa é mais sutil: Vamos procurar saber se $i > 0$ é verdadeira. Supondo isto, tem-se que i é positivo. Multiplicando $i \times i$, (positivo vezes positivo) sabe-se que o resultado deve ser positivo e $i \times i = -1$ e portanto $i > 0$ é falso. Fazendo o outro caminho, suponhamos que $i < 0$ é verdadeiro, ou seja, i é negativo. Novamente fazendo $i \times i$ teríamos negativo vezes negativo, que como sabemos é positivo. Mas, $i \times i = -1$ e portanto $i < 0$ também é falso. A conclusão do fato de que $i < 0$ é falso e $i > 0$ também é falso (além de que obviamente $i = 0$ por óbvio é falso também), nos leva a concluir que a relação maior e a menor, NÃO estão definidas no conjuntos dos complexos, daí que em princípio não se pode ordenar um conjunto de complexos. Para deixar bem claro não se pode dizer, por exemplo, que $10 + 10i$ é maior ou menor do que $2 + 2i$. Até aqui apenas as relações $=$ e $\neq$ persistem, já que $10 + 10i \neq 2 + 2i$.

Este exercício surge aqui para mostrar como é possível generalizar a operação de ordenação. Quando $<$ ou $>$ não estão definidas basta criá-las, dando-lhes um funcionamento lógico que assegure os 2 princípios básicos: não contradição (se A é verdadeiro OU A é falso, A não pode ser as duas coisas ao mesmo tempo) e terceiro excluído (se A é verdadeiro então $\sim A$ é falso e vice versa). Para construir isso nos complexos vai-se comparar PRIMEIRO o valor absoluto ou módulo do complexo ($\sqrt{a^2 + b^2}$ em $a + bi$) e havendo empate comparar o ângulo θ do número no plano de Argand-Gauss via $\tan(\theta) = \frac{b}{a}$ ou $\arctg \frac{b}{a} = \theta$. Agora, no exemplo acima $|\sqrt{10^2 + 10^2} = 10| > |\sqrt{2^2 + 2^2} = 2|$ e $|10| > |2|$ é verdadeiro. Se houvesse empate, o ângulo com o eixo real seria usado para desempatar.

Pode-se usar qualquer algoritmo de ordenação aqui, como a ordenação *quick* (devida a Hoare, 1960), cuja implementação bem simples em python pode ser (uma sugestão: aproveite a oportunidade para admirar a beleza sem igual deste algoritmo e desta linguagem).

```
def quick(lis):
    if len(lis)<2:
        return lis
    else:
        pivot=lis[0]
        menor=[i for i in lis[1:] if i<=pivot]
        maior=[i for i in lis[1:] if i>pivot]
        return quick(menor)+[pivot]+quick(maior)
print(quick([[3,1],[7,2],[1,3],[88,4]]))
# cada elemento é [valor,identificação]
```

Compare agora com o algoritmo de quick (de nome quick2), adaptado para a ordenação de complexos de acordo com a descrição acima

```
def meig(a,b):
    if (((a[0]**2)+(a[1]**2))*0.5)<
        (((b[0]**2)+(b[1]**2))*0.5)):
        return True
    if (((a[0]**2)+(a[1]**2))*0.5)>
        (((b[0]**2)+(b[1]**2))*0.5)):
        return False
    if (math.atan(a[1]/a[0])<=
```

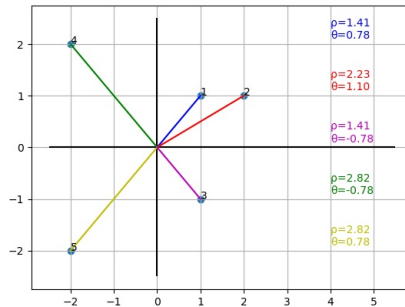
```
math.atan(b[1]/b[0])):
    return True
else:
    return False

def quick2(lis):
    if len(lis)<2:
        return lis
    else:
        pv=lis[0]
        me=[i for i in lis[1:] if meig(i,pv)]
        ma=[i for i in lis[1:] if not meig(i,pv)]
        return quick2(me)+[pv]+quick2(ma)
```

```
print(quick2([[1,1,1],[2,1,2],[1,-1,3],
               [-2,2,4],[-2,-2,5]]))
```

cada ponto tem [real, imaginário, id]

Se você plotar os pontos acima, verá que os mesmos em ordem ascendente são 3,1,2,4,5. Como se esperava.



Fractal de Sierpinski

A geometria fractal é uma consequência interessante do casamento entre matemática e computação, ocorrida em meados do século XX. Ainda assim, inúmeros aspectos da geometria fractal extrapolam a computação e se constituem em legítimos resultados matemáticos de fato e de direito. O fractal de Sierpinski é um desenho de uma figura geométrica (usualmente um triângulo equilátero, mas pode ser qualquer outro) que é dividido em quatro outras figuras congruentes entre si, sendo que o quarto não faz parte do fractal. O processo segue sendo aplicado a cada uma das sub-figuras até o limite ∞ . Algumas das propriedades do triângulo de Sierpinski

- tem tantos pontos quanto o $\mathbb{R}$
- tem área zero
- é auto semelhante (qualquer parte é semelhante ao todo). Esta propriedade também é chamada de Invariância de escala.
- descrito em 1915 por Wacław Sierpinski

Se uma linha tem dimensão 1 e uma área tem dimensão 2, o fractal de Sierpinski tem dimensão de $\log(3)/\log(2) = \log_2(3) = 1,58496\dots$ Isto acontece porque a figura é formada por 3 cópias de si mesma, cada uma reduzida por um fator 2. A área da figura é zero, pois a cada interação a área total é reduzida de 25% e a figura vai ao limite. Antes de ir ao código, vamos estudar um pacote bem interessante do Python, a saber o turtle: ele implementa a tartaruga de Wally Feurzig e Seymour Papert em 1966 através da linguagem Logo. É um ótimo ambiente para ensinar programação para crianças. Trata-se de um agente (a tartaruga) que caminha sobre o papel deixando (ou não) um rastro e consequentemente desenhando sobre o papel. Um exemplo da capacidade de desenho deste pacote está na criação do Pikachu, (trazido da internet)

Python Turtle Graphics



A seguir, um código Python capaz de produzir o fractal de Sierpinski

```
import turtle

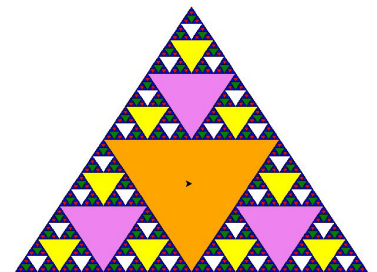
def desenhatri(pontos, cor, tarta):
    tarta.fillcolor(cor)
    tarta.up()
    tarta.goto(pontos[0][0], pontos[0][1])
    tarta.down()
    tarta.begin_fill()
    tarta.goto(pontos[1][0], pontos[1][1])
    tarta.goto(pontos[2][0], pontos[2][1])
    tarta.goto(pontos[0][0], pontos[0][1])
    tarta.end_fill()

def metade(p1, p2):
    return ((p1[0]+p2[0])/2, (p1[1]+p2[1])/2)

def sierpinski(pontos, grau, tarta):
    cormap = ['blue', 'red', 'green',
              'white', 'yellow', 'violet',
              'orange']
    desenhatri(pontos, cormap[grau], tarta)
    if grau > 0:
        sierpinski([pontos[0],
                    metade(pontos[0], pontos[1]),
                    metade(pontos[0], pontos[2])],
                    grau-1, tarta)
        sierpinski([pontos[1],
                    metade(pontos[0], pontos[1]),
                    metade(pontos[1], pontos[2])],
                    grau-1, tarta)
        sierpinski([pontos[2],
                    metade(pontos[0], pontos[2]),
                    metade(pontos[1], pontos[2])],
                    grau-1, tarta)
```

```
def main():
    tarta = turtle.Turtle()
    myWin = turtle.Screen()
    meuspon = [[-300, -150], [0, 300], [300, -150]]
    sierpinski(meuspon, 6, tarta)
    ts = turtle.getscreen()
    ts.getcanvas().postscript(file="c:/.../sir.eps")
    myWin.exitonclick()
main()
```

Eis o resultado do código acima:



Para você fazer

Estude os dois códigos acima (ordenação de complexos e triângulo de Sierpinski) e implemente-os em ambiente Python nativo (do aluno) e faça algumas experiências. Se quiser, baixe da internet o código do pikachu e mude a cor do pikachu para azul.



- 1 - /

Truques de programação I - C++

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando aprendido e apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para você fazer

Os seus dados estão em um arquivo publicado no lugar de sempre com o nome de

F183001.myd

Peso de um trem Um trem é composto por muitos vagões conectados. Neste exercício você é um despachante de trens. Uma composição pode ter dezenas de vagões e locomotivas e o trem precisa passar por pontes e viadutos que tem limites rígidos de peso suportado. Cada vagão tem um certo peso e a composição precisa ser pensada para que num determinado instante, a soma de 5 ou menos carros adjacentes (5 é um número fixo, e ele corresponde ao comprimento da maior ponte na malha ferroviária) não ultrapasse o limite de peso das estruturas pelas quais o trem passará. Os últimos valores do vetor geralmente correspondem ao peso das locomotivas, e elas devem ser tratadas como se vagões fossem. Do ponto de vista deste problema, não há diferença entre vagão e locomotiva, ambos pesam.

Nos limites inicial e final do trem, a soma do peso deve levar em consideração este fato. Assim, se apenas a locomotiva está sobre a ponte, apenas o peso dela deve ser considerado. À medida em que mais vagões vão entrando, a soma deve ser incrementada, até o limite de 5 parcelas, pois quando o sexto vagão entrar, o primeiro terá saído da ponte e deve ser desconsiderado.

Você deve escrever um programa que leia um vetor representando os pesos de cada vagão e locomotivas de um trem e um limite de peso máximo suportado naquele trajeto e deve certificar se aquele trem pode fazer ou não essa viagem.

Havendo ultrapassagem do limite proposto, o programa deve sinalizar qual o vagão central (o terceiro do conjunto de 5) que ocasionou este fato. Para esta sinalização, pode numerar os vagões a partir do zero, tal como o C++ faz.

Se ao final o trem estiver liberado o programa deve mandar a mensagem "Liberado" ou senão a mensagem "O trem nao pode passar".

A seguir, dois exemplos de execução: Para um trem formado por: 100, 3, 3, 3, 2, 5 e 200 e com um limite de 101 o programa deve informar ter ultrapassado o limite nos vagões 0, 1, 2, 4, 5 e 6, e portanto "o trem não pode passar".

Um segundo exemplo é: 10, 20, 30, 40, 50, 40, 30, 20, 10 e 160, com limite de 200. Os vagões 7 e 8 impedem a passagem. Finalmente o mesmo trem com limite de 261, permite a passagem.

No arquivo acima, há 100 trens cada um com 20 vagões/locomotivas. A 21ª coluna do arquivo é o limite de peso que as pontes aguentam. Processe os 100 trens e descubra quantos estão liberados para a viagem.

liberados

Truque: janela deslizante Trata-se de uma janela que percorre os dados e em cada momento faz algo com os dados que estão sob a janela naquele instante. A idéia é muito usada em processamento digital de imagens, para filtros, melhoradores, etc. Aqui a janela é unidimensional correspondendo a 5 vagões. As dificuldades do truque são representadas pelo tratamento dos limites ou das bordas. Uma possibilidade é acrescentar dois zeros antes e depois do vetor e começar a análise pela terceira posição, encerrando-a na antepenúltima. Só para sua ciência, esta operação que acabamos de realizar é uma operação de **convolução**.

```
#include<iostream>
#define N 10
using namespace std;
int main(){
    int v[N]={10,20,30,40,50,40,30,20,10,160};
    int a[N+4]={0};
    int lim,i,simnao=0,som;
    cout<<"Informe o limite: ";
    cin>>lim;
    for (i=0;i<N;i++){
        a[i+2]=v[i];
    }
    for (i=2;i<N+2;i++) {
        som=a[i-2]+a[i-1]+a[i]+a[i+1]+a[i+2];
        if (som>lim){
            cout<<i-2<<" ultrapassou"<<endl;
            simnao=1;
        }
    }
    if (simnao==0){cout<<"Liberado"<<endl;}
    else {cout<<"nao pode passar"<<endl;}
}
```

Porto e Containeres Como se sabe o comércio mundial teve um enorme incremento com o surgimento da idéia de container, por Malcom Mc Lean, em 1956. Trata-se de uma caixa de aço que é embalada na origem, manipulada em diversos modais (caminhão, trem e navio) de forma ágil e desemballada apenas no destino.

Imagine um megaporto com um enorme pátio de containeres. Para nossa facilidade o pátio está dividido em ruas e avenidas e entre elas existem quarteirões onde os containeres são armazenados. Nesta simulação, o pátio é representado por uma matriz retangular. Nela, cada célula contém o número de containeres que existem naquele quarteirão no pátio. Por exemplo, se na matriz, na linha 8 coluna 5 existe o valor 88, isto significa que no quarteirão da avenida 8 com a rua 5 existem 88 containeres guardados.

Escreva um programa que leia essa matriz de M linhas por N colunas (M e N definidas externamente) chamada PORTO contendo a disposição atual dos containeres no pátio e determine o a quantidade e localização do maior e do menor quarteirões e também a distância em linha reta (medida em quarteirões) entre eles. Por característica do problema nunca haverá dois ou mais valores iguais na matriz.

Por exemplo,
18 17 22 31 12 -> maior:90 L1, C4
16 8 2 4 90 -> menor: 1 L1, C2
32 33 34 35 36 -> dist.: 2,82
40 41 1 11 15

1 2 3 4 5 6 -> mai: 24 L3, C5
7 8 9 10 11 12 -> men: 1 L0, C0
13 14 15 16 17 18 -> d: 5,83
19 20 21 22 23 24

Truque: Teorema de Pitágoras

Usado aqui para calcular a distância entre dois pontos. Basta criar um triângulo retângulo no qual a hipotenusa é o valor da distância a descobrir. Agora o problema se transfere para descobrir o valor dos catetos do triângulo. Neste caso é fácil pelo uso de um sistemas de coordenadas ortogonais.

```
int main(){
    int p[M][N]={1,2,3,4,5,6},{7,8,9,10,11,12},
        {13,14,15,16,17,18},{19,20,21,22,23,24}};
    int i,j,ma,me,ima,jma,ime,jme;
    float dist,cat1,cat2;
    ma=-99999999;
    for (i=0;i<M;i++){
        for (j=0;j<N;j++){
            if (p[i][j]>ma){
                ma=p[i][j];
                ima=i;
                jma=j; } } }
    me=-99999999;
    for (i=0;i<M;i++){
        for (j=0;j<N;j++){
            if (p[i][j]<me){
                me=p[i][j];
                ime=i;
                jme=j; } } }
    cout<<"Maior: "<<ma<<" L= "
        <<ima<<" C= "<<jma<<endl;
    cout<<"Menor: "<<me<<" L= "
        <<ime<<" C= "<<jme<<endl;
    cat1=abs((float) ima-ime);
    cat2=abs((float) jma-jme);
```

```
dist=pow(cat1,2)+pow(cat2,2);
dist=sqrt(dist);
cout<<"Distancia entre eles: "<<dist<<endl;
}
```

No arquivo acima há 4 configurações de pátios, cada uma com 22 x 22 containeres. Em cada uma você deve descobrir qual a distância entre o ponto de máximo e o de mínimo em cada pátio.

1	2	3	4

Avaliação de uma mão de poker

Como se sabe no jogo de poker, as possibilidades de pontuação são
* straight flush (5 cartas seguidas do mesmo naipe)
* quadra (4 cartas de mesmo valor)
* full house (3 cartas iguais entre si e outras 2 iguais entre si)
* flush (5 cartas de mesmo naipe)
* sequencia: (5 cartas seguidas)
* trinca (3 cartas iguais)
* jogos menores: 2 pares, 1 par, maior carta...

A próxima discussão é a respeito da representação das cartas. Inúmeras possibilidades ocorrem aqui, mas vai-se usar a seguinte: O naipe da carta é representado pela centena: 100=ouros, 200=espadas, 300=copas e 400=paus. A carta será representada pelas dezenas: 01=ás, 02 a 10=a própria carta, 11=dama, 12=valete e 13=rei. Por exemplo, a carta 408 é um 8 de paus, 113 é um rei de ouros e 201 é o ás de espadas.

Truque: usar e abusar de resto e de divisão inteira

Usar-se-ão estas 2 operações para quebrar um número como 113 em 100 (113 div 100) e em 13 (113 mod 100).

```
int quadra(int c[5]){
    int x[5];
    int i;
    for (i=0;i<5;i++){
        x[i]=c[i]%100;
    }
    sort(x,x+5);
    if ((x[0]==x[1]&&(x[1]==x[2])&&(x[2]==x[3]))
        ||((x[1]==x[2])&&(x[2]==x[3])&&(x[3]==x[4])))
        return 1;
    else {return 0;}
}
```

```
int fullh(int c[5]){
    int x[5];
    int i;
    for (i=0;i<5;i++){
        x[i]=c[i]%100;
    }
    sort(x,x+5);
    if ((x[0]==x[1])&&(x[2]==x[3])&&(x[3]==x[4]))
        ||((x[0]==x[1])&&(x[1]==x[2])&&(x[3]==x[4])))
        return 1;
    else {return 0;}
}
```

```
int flush(int c[5]){
    // desenvolva sua criatividade
}
```

```
int seque(int c[5]){
    // desenvolva sua criatividade
}
```

```
int trinca(int c[5]){
    // desenvolva sua criatividade
}
```

No arquivo, há 200 rodadas, cada uma com 5 mãos de poker, totalizando 1000 mãos. Estude-as e descubra quantos flushes, quadras, full-hands, sequencias e trincas há.

fl	qu	fh	se	tr



Truques de programação I - Python

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando aprendido e apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para você fazer

Os seus dados estão em um arquivo publicado no lugar de sempre com o nome de

F183001.myd

Peso de um trem Um trem é composto por muitos vagões conectados. Neste exercício você é um despachante de trens. Uma composição pode ter dezenas de vagões e locomotivas e o trem precisa passar por pontes e viadutos que tem limites rígidos de peso suportado. Cada vagão tem um certo peso e a composição precisa ser pensada para que num determinado instante, a soma de 5 ou menos carros adjacentes (5 é um número fixo, e ele corresponde ao comprimento da maior ponte na malha ferroviária) não ultrapasse o limite de peso das estruturas pelas quais o trem passará. Os últimos valores do vetor geralmente correspondem ao peso das locomotivas, e elas devem ser tratadas como se vagões fossem. Do ponto de vista deste problema, não há diferença entre vagão e locomotiva, ambos pesam.

Nos limites inicial e final do trem, a soma do peso deve levar em consideração este fato. Assim, se apenas a locomotiva está sobre a ponte, apenas o peso dela deve ser considerado. À medida em que mais vagões vão entrando, a soma deve ser incrementada, até o limite de 5 parcelas, pois quando o sexto vagão entrar, o primeiro terá saído da ponte e deve ser desconsiderado.

Você deve escrever um programa que leia um vetor representando os pesos de cada vagão e locomotivas de um trem e um limite de peso máximo suportado naquele trajeto e deve certificar se aquele trem pode fazer ou não essa viagem.

Havendo ultrapassagem do limite proposto, o programa deve sinalizar qual o vagão central (o terceiro do conjunto de 5) que ocasionou este fato. Para esta sinalização, pode numerar os vagões a partir do zero, tal como o C++ faz.

Se ao final o trem estiver liberado o programa deve mandar a mensagem "Liberado" ou senão a mensagem "O trem nao pode passar".

A seguir, dois exemplos de execução: Para um trem formado por: 100, 3, 3, 3, 2, 5 e 200 e com um limite de 101 o programa deve informar ter ultrapassado o limite nos vagões 0, 1, 2, 4, 5 e 6, e portanto "o trem não pode passar".

Um segundo exemplo é: 10, 20, 30, 40, 50, 40, 30, 20, 10 e 160, com limite de 200. Os vagões 7 e 8 impedem a passagem. Finalmente o mesmo trem com limite de 261, permite a passagem.

No arquivo acima, há 100 trens cada um com 20 vagões/locomotivas. A 21ª coluna do arquivo é o limite de peso que as pontes aguentam. Processe os 100 trens e descubra quantos estão liberados para a viagem.

liberados

Truque: janela deslizante Trata-se de uma janela que percorre os dados e em cada momento faz algo com os dados que estão sob a janela naquele instante. A idéia é muito usada em processamento digital de imagens, para filtros, melhoradores, etc. Aqui a janela é unidimensional correspondendo a 5 vagões. As dificuldades do truque são representadas pelo tratamento dos limites ou das bordas. Uma possibilidade é acrescentar dois zeros antes e depois do vetor e começar a análise pela terceira posição, encerrando-a na antepenúltima. Para seu conhecimento, a operação que acabamos de realizar chama-se **convolução**.

```
def f183():
    f=open("c:/p/n/183/f183001_exemplo.myd","r")
    i=0
    liberado=0
    while(i<100):
        x=f.readline()
        trem = x.split()
        tr=[]
        for j in trem:
            tr.append(int(j))
        # processa tr (que é uma lista de inteiros)
        k=2
        passou=0
        while (k<18):
            if (tr[k-2]+tr[k-1]+tr[k]+tr[k+1]+tr[k+2])
                <=tr[20]:
                passou=passou+1
                k=k+1
            if passou==16:
                liberado=liberado+1
                i=i+1
        print("Trens liberados: ",liberado)
```

Porto e Containeres Como se sabe o comércio mundial teve um enorme incremento com o surgimento da idéia de container inventado por Malcom Mc Lean, em 1956. Trata-se de uma caixa de aço que é embalada na origem, manipulada em diversos modais (caminhão, trem e navio) de forma ágil e desembalada apenas no destino.

Imagine um megaporto com um enorme pátio de containeres. Para nossa facilidade o pátio está dividido em ruas e avenidas e entre elas existem quarteirões onde os containeres são armazenados. Nesta simulação, o pátio é representado por uma matriz retangular. Nela, cada célula contém o número de containeres que existem naquele quarteirão no pátio. Por exemplo, se na matriz, na linha 8 coluna 5 existe o valor 88, isto significa que no quarteirão da avenida 8 com a rua 5 existem 88 containeres guardados.

Escreva um programa que leia essa matriz de M linhas por N colunas (M e N definidas externamente) chamada PORTO contendo a disposição atual dos containeres no pátio e determine o a quantidade e localização do maior e do menor quarteirões e também a distância em linha reta (medida em quarteirões) entre eles. Por característica do problema nunca haverá dois ou mais valores iguais na matriz.

```
Por exemplo,
18 17 22 31 12 -> maior:90 L1, C4
16 8 2 4 90 -> menor: 1 L1, C2
32 33 34 35 36 -> dist.: 2,82
40 41 1 11 15
```

```
1 2 3 4 5 6 -> mai: 24 L3, C5
7 8 9 10 11 12 -> men: 1 L0, C0
13 14 15 16 17 18 -> d: 5,83
19 20 21 22 23 24
```

Truque: Teorema de Pitágoras

Usado aqui para calcular a distância entre dois pontos. Basta criar um triângulo retângulo no qual a hipotenusa é o valor da distância a descobrir. Agora o problema se transfere para descobrir o valor dos catetos do triângulo. Neste caso é fácil pelo uso de um sistemas de coordenadas ortogonais.

```
i=0
while (i<4):
    patio=np.zeros((22,22),int)
    lin=0
    while lin<22:
        x=f.readline()
        pa=x.split()
        p=[]
        for j in pa:
            p.append(int(j))
        xixa=0
        while xixa<22:
            patio[lin][xixa]=p[xixa]
            xixa=xixa+1
            lin=lin+1
        c1x=np.argmax(patio)//22
        c1y=np.argmax(patio)%22
        c2x=np.argmin(patio)//22
        c2y=np.argmin(patio)%22
        print("Max-Min: ",i+1,
            (((c1x-c2x)**2)+((c1y-c2y)**2))**.5)
        i=i+1
```

No arquivo acima há 4 configurações de pátios, cada uma com 22 x 22 containeres. Em cada uma

você deve descobrir qual a distância entre o ponto de máximo e o de mínimo em cada pátio.

1	2	3	4

Avaliação de uma mão de poker

Como se sabe no jogo de poker, as possibilidades de pontuação são
* straight flush (5 cartas seguidas do mesmo naipe)
* quadra (4 cartas de mesmo valor)
* full house (3 cartas iguais entre si e outras 2 iguais entre si)
* flush (5 cartas de mesmo naipe)
* sequencia: (5 cartas seguidas)
* trinca (3 cartas iguais)
* jogos menores: 2 pares, 1 par, maior carta...

A próxima discussão é a respeito da representação das cartas. Inúmeras possibilidades ocorrem aqui, mas vai-se usar a seguinte: O naipe da carta é representado pela centena: 100=ouros, 200=espadas, 300=copas e 400=paus. A carta será representada pelas dezenas: 01=ás, 02 a 10=a própria carta, 11=dama, 12=valeta e 13=rei. Por exemplo, a carta 408 é um 8 de paus, 113 é um rei de ouros e 201 é o ás de espadas.

Truque: usar e abusar de resto e de divisão inteira

Usar-se-ão estas 2 operações para quebrar um número como 113 em 100 (113 div 100) e em 13 (113 mod 100).

```
def quadra(x):
    l=[0]*5
    for i in range(5):
        l[i]=x[i]%100
    l.sort()
    if ((l[0]==l[1] and l[1]==l[2] and l[2]==l[3])
        or (l[1]==l[2] and l[2]==l[3] and l[3]==l[4])):
        return 1
    else:
        return 0
def fullh(x):
    l=[0]*5
    for i in range(5):
        l[i]=x[i]%100
    l.sort()
    if ((l[0]==l[1] and l[1]==l[2] and l[3]==l[4])
        or (l[0]==l[1] and l[2]==l[3] and l[3]==l[4])):
        return 1
    else:
        return 0
def flush(x):
def seque(x):
def trinca(x):
...
i=0 ; ctf1=ctqu=ctfh=ctse=cttr=0
while i<200:
    x=f.readline()
    pa=x.split()
    p=[]
    for j in pa:
        p.append(int(j))
    for k in range(5):
        ctf1=ctf1+flush(p[(k*5):5+(k*5):])
        ctqu=ctqu+quadra(p[(k*5):5+(k*5):])
        ctfh=ctfh+fullh(p[(k*5):5+(k*5):])
        ctse=ctse+seque(p[(k*5):5+(k*5):])
        cttr=cttr+trinca(p[(k*5):5+(k*5):])
    i=i+1
    print("Flush = ",ctf1)
    print("Quadra = ",ctqu)
    print("Full hand = ",ctfh)
    print("Sequencia = ",ctse)
    print("Trinca = ",cttr)
```

No arquivo, há 200 rodadas, cada uma com 5 mãos de poker, totalizando 1000 mãos. Estude-as e descubra quantos flushes, quadras, full-hands, sequencias e trincas há.

fl	qu	fh	se	tr



Truques de programação II - C++

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F184001.myd

publicado no lugar usual.

Números similares Faça um programa para ler 2 números naturais de 3 algarismos e verificar se eles são similares. Para serem similares estes números precisam ter os mesmos algarismos na sua formação.

Exemplo de execução:
Digite 2 números: 231 312
Números similares

Digite 2 números: 452 357
Números não similares

Outro exemplo de execução:
Digite 2 números: 167 167
Números similares

O truque: quebrar um número inteiro qualquer em seus dígitos componentes e a seguir processar tais dígitos isoladamente é tarefa comum em programação. Primeiro, a quantidade de dígitos é obtida pela expressão $ceil(log_{10}(x))$ que só não funciona em números cheios: 1 seguido de n zeros. A separação de dígitos pode ser feita, rodando-se sucessivamente o trecho de programação (em C++)

```
x é o número que se quer quebrar
digitos[n] é um vetor de dígitos
int i,j,dig[...];
i=0;
while (x>0){
    dig[i]=x%10;
    x=x/10;
    i++;
}
for (j=i-1;j>=0;j--){
    cout<<dig[j]<<" ";
}
```

Este trecho só não funciona para o número 0 que deve ser tratado à parte. Para este problema em particular, como se sabe que os números terão 3 dígitos, a conversão pode ser mais direta:

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main() {
    int a,b,d0,d1,d2,e0,e1,e2;
    cout<<"Informe 2 numeros ";
    cin>>a>>b;
    d0=a/100;
    d2=a%10;
    d1=(a/10)%10;
    e0=b/100;
    e2=b%10;
    e1=(b/10)%10;
    if (((d0==e0) || (d0==e1) || (d0==e2)) &&
        // complete aqui.....
        cout<<"similares";}
    else {
        cout<<"nao similares";}
}
```

Para você fazer Leia no arquivo acima descrito, 500 sequencias de 2 números de 3 dígitos e informe quantas dessas sequencias são formadas por números similares.

números similares:

Brinquedos inteligentes A empresa tem muitos brinquedos pequenos sempre guardados em caixas no formato de cubos (paralelepípedos retos com arestas iguais). A empresa comprou muitas esferas de raios 10, 20 e 30 cm. Pretende encerrar cada brinquedo dentro de uma esfera (como se fosse um kinder ovo). Você deve escrever um programa que peça e receba a aresta do cubo de um brinquedo e deve descobrir qual a menor esfera que pode escondê-lo, imprimindo o raio da esfera. Se o brinquedo não couber nem na maior esfera, o programa deve imprimir NAO CABE. O programa deve processar uma quantidade indeterminada de brinquedos, parando ao receber uma aresta negativa, que não deve ser processada e indica final dos dados. Dados: $V_{cubo} = a^3$, $V_{esfera} = 4/3\pi \times r^3$, Diagonal do cubo: $a \times \sqrt{3}$, Diâmetro da esfera: $2 \times r$

Dados de uma execucao real:
Informe a aresta do brinquedo 10
esfera de 10cm de raio
Informe a aresta do brinquedo 12
esfera de 20cm de raio
Informe a aresta do brinquedo 20
esfera de 20cm de raio
Informe a aresta do brinquedo 30
esfera de 30cm de raio
Informe a aresta do brinquedo -1

Um possível gabarito

```
#include<iostream>
#include<cmath>
using namespace std;
int main() {
    float aresta,raio,diagonal;
    while (1 == 1){
        cout<<"Informe a aresta do brinquedo ";
        cin>>aresta;
        if (aresta < 0){
            return (0);
        }
        diagonal=aresta*sqrt(3);
        if (diagonal < 20){
            cout<<"esfera de 10cm de raio" << endl;
        }
        if (diagonal >=20 && diagonal < 40){
            cout<<"esfera de 20cm de raio " << endl;
        }
        if (diagonal >=40 && diagonal < 60){
            cout<<"esfera de 30cm de raio " << endl;
        }
        if (diagonal >=60){
            cout<<"Nao cabe" << endl;
        }
    }
}
```

O truque: Processar objetos com base em suas características geométricas. Aqui, tem-se um cubo inscrito a uma esfera. Seus pontos de contato são os pares de vértices opostos do cubo, que neste ponto coincidem com um diâmetro da esfera.

No arquivo acima, há 1000 brinquedos representados por suas arestas. Processe-os e descubra quantos brinquedos cabem na esferas de 10,20 e 30 cm além dos que não cabem.

10cm	20cm	30cm	não cabem

distância cartesiana Escreva um programa que leia as coordenadas cartesianas (x,y) de 2 pontos, A e B no espaço e imprima na tela a menor distância entre os 2 pontos e também a distância do menor caminho de A até B passando por um ponto intermediário, C com coordenadas (0,0). Deve ser definida e usada a função de nome dist() que retorna a distância entre 2 pontos, cujas coordenadas são passadas como parâmetros. Obs.: Considere que a distância entre 2 pontos de coordenadas (x_a, y_a) e (x_b, y_b) é dada por $distAB = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$ A distância do caminho de A até B passando por C = (0,0) é dado por caminho AB = dist AC + dist CB .

Exemplo de execução:
Digite coordenada de A: 10 50
Digite coordenada de B: 70 35
Menor distância entre A e B: 61.8466
Caminho de A a B, passando por (0,0): 129.253

O truque: Trata-se de associar a objetos do mundo real, suas coordenadas em algum sistema cartesiano. Com a popularização do Google maps e sistemas similares, bem como o barateamento dos dispositivos GPS, é questão de tempo que praticamente todos os itens que entrem em algum meio magnético (computacional) tragam com eles suas coordenadas. Daí a precisar calcular distâncias é um passo bem pequeno.

```
float distab(float xa, float ya,
             float xb, float yb){
    float dx, dy;
    dx=xa-xb;
    dy=ya-yb;
    return sqrt((dx*dx)+(dy*dy));
}
int q2(){
    float xa,xb,ya,yb,dab,dabc;
    cout<<"Digite coordenadas de A: ";
    cin>>xa>>ya;
    cout<<"Digite coordenadas de B: ";
    cin>>xb>>yb;
    dab=distab(xa,ya,xb,yb);
    dabc=distab(xa,ya,0.0,0.0);
    dabc=dabc+distab(0.0,0.0,xb,yb);
    cout<<"Menor distancia A-B: "<<dab<<endl;
    cout<<"caminho A-C-B: "<<dabc<<endl;
}
```

No arquivo há 1000 pares de pontos. As menores distâncias entre eles devem ser somadas, bem como os caminhos de A a B passando por C.

$\sum A - B$	$\sum A - B - C$

Para você fazer

similares	10cm	20cm
30cm	+30cm	$\sum AB$
$\sum ACB$	//// //// //// //// //// //// //// //// //// ////	//// //// //// //// //// //// //// //// //// ////



Truques de programação II - Python

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F184001.myd

publicado no lugar usual.

Números similares Faça um programa para ler 2 números naturais de 3 algarismos e verificar se eles são similares. Para serem similares estes números precisam ter os mesmos algarismos na sua formação.

Exemplo de execução:
Digite 2 números: 231 312
Números similares

Digite 2 números: 452 357
Números não similares

Outro exemplo de execução:
Digite 2 números: 167 167
Números similares

O truque: quebrar um número inteiro qualquer em seus dígitos componentes e a seguir processar tais dígitos isoladamente é tarefa comum em programação. Primeiro, a quantidade de dígitos é obtida pela expressão $ceil(log_{10}(x))$ que só não funciona em números cheios: 1 seguido de n zeros. A separação de dígitos pode ser feita, rodando-se sucessivamente o trecho de programação (em Python)

```
x é o número que se quer quebrar
digitos[n] é um vetor de dígitos
dig=[]
x=...
while x>0:
    dig.append(x%10)
    x=x//10
dig.reverse()
print(dig)
```

Este trecho só não funciona para o número 0 que deve ser tratado à parte. Para este problema em particular, como se sabe que os números terão 3 dígitos, a conversão pode ser mais direta:

```
def f184():
    f=open("c:/p/n/184/f184001_exemplo.myd","r")
    i=0
    ct=0
    while i<500:
        x=f.readline()
        nums=x.split()
        a=int(nums[0])
        b=int(nums[1])
        d0=a//100
        d2=a%10
        d1=(a//10)%10
        e0=b//100
        e2=b%10
        e1=(b//10)%10
        if (((d0==e0) or (d0==e1) or (d0==e2)) and
            ((d1==e0) or (d1==e1) or (d1==e2)) and
            ((d2==e0) or (d2==e1) or (d2==e2))):
            if (((e0==d0) or (e0==d1) or (e0==d2)) and
                ((e1==d0) or (e1==d1) or (e1==d2)) and
                ((e2==d0) or (e2==d1) or (e2==d2))):
                ct=ct+1
        i=i+1
    print(ct)
```

Para você fazer Leia no arquivo acima descrito, 500 sequencias de 2 números de 3 dígitos e informe quantas dessas sequencias são formadas por números similares.

números similares:

Brinquedos inteligentes A empresa tem muitos brinquedos pequenos sempre guardados em caixas no formato de cubos (paralelepípedos retos com arestas iguais). A empresa comprou muitas esferas de raios 10, 20 e 30 cm. Pretende encerrar cada brinquedo dentro de uma esfera (como se fosse um kinder ovo). Você deve escrever um programa que peça e receba a aresta do cubo de um brinquedo e deve descobrir qual a menor esfera que pode escondê-lo, imprimindo o raio da esfera. Se o brinquedo não couber nem na maior esfera, o programa deve imprimir NAO CABE. O programa deve processar uma quantidade indeterminada de brinquedos, parando ao receber uma aresta negativa, que não deve ser processada e indica final dos dados. Dados: $V_{cubo} = a^3$, $V_{esfera} = 4/3\pi \times r^3$, Diagonal do cubo: $a \times \sqrt{3}$, Diâmetro da esfera: $2 \times r$

Dados de uma execucao real:
Informe a aresta do brinquedo 10
esfera de 10cm de raio
Informe a aresta do brinquedo 12
esfera de 20cm de raio
Informe a aresta do brinquedo 20
esfera de 20cm de raio
Informe a aresta do brinquedo 30
esfera de 30cm de raio
Informe a aresta do brinquedo -1

Um possível gabarito

```
a10=a20=a30=anao=0
i=0
p=[]
while i<500:
    x=f.readline()
    nums=x.split()
    a=int(nums[0])
    b=int(nums[1])
    p.append(a)
    p.append(b)
    i=i+1
for i in range(1000):
    diagonal=p[i]*np.sqrt(3)
    if diagonal<=10:
        a10=a10+1
    if diagonal>10 and diagonal <= 20:
        a20=a20+1
    if diagonal>20 and diagonal <=30:
        a30=a30+1
    if diagonal > 30:
        anao=anao+1
print("10=",a10,"20=",a20,"30=",a30,"-=",anao)
```

O truque: Processar objetos com base em suas características geométricas. Aqui, tem-se um cubo inscrito a uma esfera. Seus pontos de contato são os pares de vértices opostos do cubo, que neste ponto coincidem com um diâmetro da esfera.

No arquivo acima, há 1000 brinquedos representados por suas arestas. Processe-os e descubra quantos brinquedos cabem na esferas de 10,20 e 30 cm além dos que não cabem.

10cm	20cm	30cm	não cabem

distância cartesiana Escreva um programa que leia as coordenadas cartesianas (x,y) de 2 pontos, A e B no espaço e imprima na tela a menor distância entre os 2 pontos e também a distância do menor caminho de A até B passando por um ponto intermediário, C com coordenadas (0,0). Deve ser definida e usada a função de nome dist() que retorna a distância entre 2 pontos, cujas coordenadas são passadas como parâmetros. Obs.: Considere que a distância entre 2 pontos de coordenadas (x_a, y_a) e (x_b, y_b) é dada por $distAB = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$ A distância do caminho de A até B passando por C = (0,0) é dado por caminho AB = dist AC + dist CB .

Exemplo de execução:
Digite coordenada de A: 10 50
Digite coordenada de B: 70 35
Menor distância entre A e B: 61.8466
Caminho de A a B, passando por (0,0): 129.253

O truque: Trata-se de associar a objetos do mundo real, suas coordenadas em algum sistema cartesiano. Com a popularização do Google maps

e sistemas similares, bem como o barateamento dos dispositivos GPS, é questão de tempo que praticamente todos os itens que entrem em algum meio magnético (computacional) tragam com eles suas coordenadas. Daí a precisar calcular distâncias é um passo bem pequeno.

```
import numpy as np
def dcart(p1,p2):
    return (((p1[0]-p2[0])**2)+
            ((p1[1]-p2[1])**2))**0.5
...
i=0
dab=dacb=0
while i<1000:
    x=f.readline()
    nums=x.split()
    p1=[int(nums[0]),int(nums[1])]
    p2=[int(nums[2]),int(nums[3])]
    dab=dab+dcart(p1,p2)
    dacb=dacb+dcart(p1,[0,0])+dcart([0,0],p2)
    i=i+1
print("D1 = ",dab," D2=",dacb)}
```

No arquivo há 1000 pares de pontos. As menores distâncias entre eles devem ser somadas, bem como os caminhos de A a B passando por C.

$\sum A - B$	$\sum A - B - C$

Para você fazer

similares	10cm	20cm
30cm	+30cm	$\sum AB$
$\sum ACB$	////// ////// ////// ////// ////// ////// //////	////// ////// ////// ////// ////// ////// //////



Truques de programação III - C++

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F185001.myd

publicado no lugar usual.

Números crescentes Vamos chamar de crescente um número natural $n = d_1d_2...d_k$ cujos dígitos d_i estão em ordem crescente, isto é, tal que $d_1 < d_2 < ... < d_k$. Faça um programa em C++ que leia um número inteiro e positivo n de 3 dígitos e verifique se n é crescente. Seu programa deve também verificar se n possui exatamente 3 dígitos e imprimir mensagens adequadas em cada caso.

```
Exemplo de execução:
Entre com um inteiro positivo:
3416
valor inválido.

Entre com um inteiro positivo: 152
152 não é crescente.

Entre com um inteiro positivo: 378
378 é crescente.
```

O truque: quebrar um número inteiro qualquer em seus dígitos componentes e a seguir processar tais dígitos isoladamente é tarefa comum em programação. Primeiro, a quantidade de dígitos é obtida pela expressão $ceil(log_{10}(x))$ que só não funciona em números cheios: 1 seguido de n zeros. Neste exemplo, como a demanda é direta (3 dígitos) basta testar o número entre 101 e 999.

A separação de dígitos pode ser feita, rodando-se sucessivamente o trecho de programação (em C++)

```
// x é o número que se quer quebrar
// digitos[n] é um vetor de dígitos
int i,j,dig[...];
i=0;
while (x>0){
    dig[i]=x%10;
    x=x/10;
    i++;
}
for (j=i-1;j>=0;j--){
    cout<<dig[j]<<" ";
}
```

Este trecho só não funciona para o número 0 que deve ser tratado à parte. Para este problema em particular, como se sabe que os números terão 3 dígitos, a conversão pode ser mais direta:

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main() {
    int a,d0,d1,d2;
    cout<<"Informe o numero ";
    cin>>a;
    d0=a/100;
    d2=a%10;
    d1=(a/10)%10;
    if ((d0>d1) && (d1>d2))
        cout<<"crescente";
    else {
        cout<<"nao crescente";
    }
}
```

Para você fazer Leia no arquivo acima descrito, 500 números e indique quantos são crescentes

números crescentes:

Números perfeitos, deficientes e abundantes Um inteiro n é dito um número perfeito se ele é igual a soma de todos os seus divisores positivos, excluindo ele mesmo. Por exemplo, 6 é um número perfeito, pois seus divisores são 1, 2 e 3 e $1+2+3 = 6$. O número 28 também é perfeito, pois $1+2+4+7+14=28$. Faça um programa em C++ que leia uma sequência de números inteiros $n_1, n_2, ..., n_k$ e informe ao usuário, para cada n_i lido, se ele é um número perfeito ou não. Aproveitando o desenvolvimento, se um número é menor que a soma dos divisores próprios ele é deficiente e se é maior ele é abundante. (por exemplo, 12 é abundante pois: $1+2+3+4+6 = 16 > 12$). Note que a origem dos índices é zero.

```
Exemplo de execução:
Entre com um inteiro: 2
2 não é um número perfeito.
Entre com um inteiro: 28
28 é um número perfeito.
Entre com um inteiro: 79
79 não é um número perfeito.
Entre com um inteiro: 8128
8128 é um número perfeito.

#include<iostream>
using namespace std;
int main(){
    int num,dii,som;
    while(1==1){
        cout<<"Entre com um inteiro: ";
        cin>>num;
        if (num==0){
            break;
        }
        dii=1+(num/2);
        som=0;
        while(dii>0){
            if (num%dii==0){
                som=som+dii;
            }
            dii--;
        }
        if ((num==som)&&(num!=1)){
            cout<<num<<" é perfeito."<<endl;
        }
        else {
            cout<<num<<" não é perfeito."<<endl;
        }
    }
}
```

O truque: Achar os divisores de um número inteiro. Note-se que se está no âmbito da matemática inteira, razão da divisão inteira e da operação resto.

No arquivo acima, há 500 valores e você deve contar quantos são perfeitos, abundantes e deficientes

perfeitos	abundantes	deficientes

Localização de mercado Uma empresa de supermercados pretende abrir uma filial em alguma região de uma cidade absolutamente regular, já que todos os seus quarteirões são quadrados de 100x100m e as ruas sempre se interceptam em ângulos retos.

Seu programa deve ler uma matriz quadrada de ordem N (N é definido via #define no começo do programa) que contém a quantidade de habitantes da cidade em cada quarteirão. Assim, se a posição 2,2 da matriz tem 50, significa que no quarteirão da linha 3, coluna 3 há 50 clientes. Note que a numeração começa em 0, assim 2 significa linha ou coluna 3.

Deve ler também 3 valores: as coordenadas x,y da possível localização do supermercado (começando em 0) e também um parâmetro denominado aa = área de abrangência do mercado. aa é sempre um número ímpar, isto não precisa ser testado. x,y nunca vai estar nos limites da cidade, assim sempre haverá espaço para acomodar a submatriz de dimensão $aa \times aa$.

O programa deve totalizar quantos clientes serão atingidos pelo mercado naquela localização e com aquela área de abrangência, somando todas as células da matriz que estejam incluídas na submatriz quadrada centralizada em x,y e com dimensões aa .

Deve imprimir também a quantidade de clientes atingidos no maior quarteirão (aquele que tem mais clientes).

Ao final a matriz deve ser impressa.

```
Exemplos de execução:
Seja a matriz (N=6)
1 1 3 0 5 3
7 2 9 1 1 2
3 4 5 6 0 6
1 2 1 2 3 4
5 2 2 8 9 3
1 2 3 3 5 3
com x=3, y=3 e aa=3, R=36 e 9 respectivamente.
com x=2, y=2 e aa=5, R=83 e 9.
com x=1, y=4 e aa=1, R=1 e 1.
com x=1, y=4 e aa=3, R=24 e 6.
```

```
#include<iostream>
#include<iomanip>
#define N 6
using namespace std;
int main(){
    int m[N][N]={...};
    int i,j,x,y,t,aa,som,maximo;
    cout<<"Informe x,y e aa: ";
    cin>>x>>y>>aa;
    t=aa/2;
    som=0;
    maximo=-999999;
    for (i=x-t;i<=x+t;i++){
        for(j=y-t;j<=y+t;j++){
            som=som+m[i][j];
            if (m[i][j]>maximo){
                maximo=m[i][j];
            }
        }
    }
    cout<<"0 total de clientes e: "<<som<<endl;
    cout<<"MQ="<<maximo<<endl;
    for (i=0;i<N;i++){
        for(j=0;j<N;j++){
            cout<<setw(4)<<m[i][j];
        }
        cout<<endl;
    }
}
```

No arquivo há 50 matrizes de 10 x 10. Após cada cidade, há 3 números que são as coordenadas da localização do mercado e também a sua abrangência. Calcule a soma das 10 somas de clientes e a soma dos 10 maiores quarteiros em cada caso.

$\sum \text{clientes}$	$\sum \text{maiores}$

Leitura de dados Para ler os dados de entrada de um arquivo (ao invés do teclado) em C++ deve-se: i.incluir o pacote <fstream>. ii.definir uma variável (digamos f) de tipo ifstream. iii. Abrir o arquivo passando a localização do mesmo. iv. fazer f>>variável (ao invés de cin>>variável). v. Ao final, fechar o arquivo (f.close()). Acompanhe

```
#include<fstream>
ifstream f;
f.open("c:/p/apl/...")
...
f>>v[i];
...
f.close()
```

🔗 Para você fazer

crescentes	perfeitos	abundantes
deficientes	$\sum \text{clientes}$	$\sum \text{maiores}$



Truques de programação III - Python

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando aprendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F185001.myd

publicado no lugar usual.

Números crescentes Vamos chamar de crescente um número natural $n = d_1d_2...d_k$ cujos dígitos d_i estão em ordem crescente, isto é, tal que $d_1 < d_2 < ... < d_k$. Faça um programa em C++ que leia um número inteiro e positivo n de 3 dígitos e verifique se n é crescente. Seu programa deve também verificar se n possui exatamente 3 dígitos e imprimir mensagens adequadas em cada caso.

```
Exemplo de execução:
Entre com um inteiro positivo:
3416
valor inválido.

Entre com um inteiro positivo: 152
152 não é crescente.

Entre com um inteiro positivo: 378
378 é crescente.
```

O truque: quebrar um número inteiro qualquer em seus dígitos componentes e a seguir processar tais dígitos isoladamente é tarefa comum em programação. Primeiro, a quantidade de dígitos é obtida pela expressão $ceil(log_{10}(x))$ que só não funciona em números cheios: 1 seguido de n zeros. Neste exemplo, como a demanda é direta (3 dígitos) basta testar o número entre 101 e 999.

A separação de dígitos pode ser feita, rodando-se sucessivamente o trecho de programação (em Python)

```
digs=[] # conterà os dígitos
x=... # o valor a quebrar
while x>0:
    digs.append(x%10)
    x=x//10
digs.reverse()
print(digs)
```

Este trecho só não funciona para o número 0 que deve ser tratado à parte. Para este problema em particular, como se sabe que os números terão 3 dígitos, a conversão pode ser mais direta:

```
def f185():
    f=open("c:/p/n/185/f185001_exemplo.myd","r")
    i=0
    p=[]
    while i<50:
        x=f.readline()
        nums=x.split()
        for j in range(10):
            p.append(int(nums[j]))
        i=i+1
    ct=0
    for i in range(len(p)):
        x=p[i]
        dig=[]
        while x>0:
            dig.append(x%10)
            x=x//10
        sentinela=0
        for j in range(1,len(dig)):
            if dig[j]>dig[j-1]:
                sentinela=1
            if sentinela==0:
                ct=ct+1
    print("Crescentes=",ct)
```

Para você fazer Leia no arquivo acima descrito, 500 números e indique quantos são crescentes

números crescentes:

Números perfeitos, deficientes e abundantes Um inteiro n é dito um número perfeito se ele é igual a soma de todos os seus divisores positivos, excluindo ele mesmo. Por exemplo, 6 é um número perfeito, pois seus divisores são 1, 2 e 3 e $1+2+3 = 6$. O número 28 também é perfeito, pois $1+2+4+7+14=28$. Faça um programa em Python que leia uma sequência de números inteiros $n_1, n_2, ..., n_k$ e informe ao usuário, para cada n_i lido, se ele é um número perfeito ou não. Aproveitando o desenvolvimento, se um número é menor que a soma dos divisores próprios ele é deficiente e se é maior ele é abundante. (por exemplo, 12 é abundante pois: $1+2+3+4+6 = 16 > 12$). Note que a origem dos índices é zero.

```
Exemplo de execução:
Entre com um inteiro: 2
2 não é um número perfeito.
Entre com um inteiro: 28
28 é um número perfeito.
Entre com um inteiro: 79
79 não é um número perfeito.
Entre com um inteiro: 8128
8128 é um número perfeito.
```

```
import numpy as np
def divs(x):
    soma=1
    for i in range(2,1+x//2):
        if x%i==0:
            soma=soma+i
    return soma
...
i=0
p=[]
while i<50:
    x=f.readline()
    nums=x.split()
    for j in range(10):
        p.append(int(nums[j]))
    i=i+1
cper=cabu=cdef=0
for i in range(len(p)):
    xx=divs(p[i])
    if xx==p[i]:
        cper=cper+1
    elif xx>p[i]:
        cabu=cabu+1
    else:
        cdef=cdef+1
print("Perf=",cper," Abu=",cabu," Def=",cdef)
```

O truque: Achar os divisores de um número inteiro. Note-se que se está no âmbito da matemática inteira, razão da divisão inteira e da operação resto.

No arquivo acima, há 500 valores e você deve contar quantos são perfeitos, abundantes e deficientes

perfeitos	abundantes	deficientes

Localização de mercado Uma empresa de supermercados pretende abrir uma filial em alguma região de uma cidade absolutamente regular, já que todos os seus quarteirões são quadrados de 100x100m e as ruas sempre se interceptam em ângulos retos.

Seu programa deve ler uma matriz quadrada de ordem N (N é definido no começo do programa) que contém a quantidade de habitantes da cidade em cada quarteirão. Assim, se a posição 2,2 da matriz tem 50, significa que no quarteirão da linha 3, coluna 3 há 50 clientes. Note que a numeração começa em 0, assim 2 significa linha ou coluna 3.

Deve ler também 3 valores: as coordenadas x,y da possível localização do supermercado (começando em 0) e também um parâmetro denominado aa = área de abrangência do mercado. aa é sempre um número ímpar, isto não precisa ser testado. x,y nunca vai estar nos limites da cidade, assim sempre haverá espaço para acomodar a submatriz de dimensão $aa \times aa$.

O programa deve totalizar quantos clientes serão atingidos pelo mercado naquela localização e com aquela área de abrangência, somando todas as células da matriz que estejam incluídas na submatriz quadrada centralizada em x, y e com dimensões aa .

Deve imprimir também a quantidade de clientes atingidos no maior quarteirão (aquele que tem mais clientes).

Ao final a matriz deve ser impressa.

```
Exemplos de execução:
Seja a matriz (N=6)
1 1 3 0 5 3
7 2 9 1 1 2
3 4 5 6 0 6
1 2 1 2 3 4
5 2 2 8 9 3
1 2 3 3 5 3
com x=3, y=3 e aa=3, R=36 e 9 respectivamente.
com x=2, y=2 e aa=5, R=83 e 9.
com x=1, y=4 e aa=1, R=1 e 1.
com x=1, y=4 e aa=3, R=24 e 6.
```

```
...
i=tudo=tudomax=0
while i<50:
    m=np.zeros((10,10))
    for j in range(10):
        x=f.readline()
        nums=x.split()
        for k in range(10):
            m[j,k]=int(nums[k])
    x=f.readline()
    nums=x.split()
    lin=int(nums[0])
    col=int(nums[1])
    tam=int(nums[2])
    tt=tam//2
    somsom=0
    maximo=-999999
    for ii in range(lin-tt,lin+tt+1):
        for jj in range(col-tt,col+tt+1):
            somsom=somsom+m[ii,jj]
            if m[ii,jj]>maximo:
                maximo=m[ii,jj]
    tudo=tudo+somsom
    tudomax=tudomax+maximo
    i=i+1
print(tudo,tudomax)
```

No arquivo há 50 matrizes de 10×10 . Após cada cidade, há 3 números que são as coordenadas da localização do mercado e também a sua abrangência. Calcule a soma das 10 somas de clientes e a soma dos 10 maiores quarteiros em cada caso.

$\sum \text{clientes}$	$\sum \text{maiores}$

Leitura de dados Para ler os dados de entrada de um arquivo (ao invés do teclado) em Python deve-se: i. abrir o arquivo, ii. ler seus registros (via `readline()`) iii. Transformar a linha lida em um vetor de caracteres (via `x.split()`), iv. Transformar cada número desses em um inteiro, colocando-o em uma lista (via `append(int(nums[k]))`). Veja no código acima como isso foi feito.

📖 Para você fazer

crescentes	perfeitos	abundantes
deficientes	$\sum \text{clientes}$	$\sum \text{maiores}$



Truques de programação IV - C++

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F186001.myd

publicado no lugar usual.

Matrizes simétricas Escreva um programa que analise uma matriz e informe se ela é ou não é simétrica. Para esta verificação, os elementos simétricos em relação à diagonal principal devem ser iguais. A matriz precisa ser quadrada e nela todos elementos $m[i][j]$ precisam ser iguais a $m[j][i]$. Por abuso, esta regra pode ser aplicada aos elementos que estão na diagonal principal, já que quem está nela tem $i = j$ e portanto $m[i][j]$ sempre será igual a $m[j][i]$ quando $i = j$. Mas, nada se perde se elementos da diagonal principal simplesmente não forem testados. De acordo com a regra, basta que um único elemento simétrico seja diferente para que a matriz toda o seja. Um possível gabarito:

```
#include<iostream>
#define N ?
using namespace std;
int imp_mat(int mat[N][N]){...}
int main(){
    int mat[N][N];
    int i,j,sentinel=1;
    le_mat(mat);
    for (i=0;i<N;i++){
        for(j=0;j<N;j++){
            if (mat[i][j]!=mat[j][i])
                sentinel=0;
        } } }
    if (sentinel==1){
        cout<<"simetrica"; }
    else {
        cout<<"nao simétrica"; }
}
```

O truque: **Sentinela** é uma variável que sinaliza uma determinada condição ao longo de um processamento completo. Usada muitas vezes, como neste contexto, no qual uma única condição perdida em um volume grande de situações e lugares (como um único elemento simétrico diferente) pode determinar uma condição mais geral (toda a matriz deixou de ser simétrica).

Para otimizar o processamento, ao invés de usar os dois ciclos *for* poderíamos ter usado ciclos *while*, incluindo neles (com AND) a condição *sentinela* = 0, para que, uma vez descoberto um elemento diferente a busca fosse encerrada.

Para você fazer Leia no arquivo acima descrito, 10 matrizes numéricas de 10 linhas por 10 colunas estão registradas. O objetivo do aluno deve ser contar quantas são simétricas.

matrizes simétricas:

Vulcão no Havai Como sabemos o Havai está neste momento em processo de manifestação vulcânica intensa. O vulcão Saidebaixo, numa ilha lá, tem algumas características interessantes:

- a) fica numa ilha que tem platôs quadrados, de altura regular e é deserta.
- b) sua lava é muito líqüefeita, quase líquida, saindo da cratera e escorrendo, se possível, para as 8 direções vizinhas.
- c) não há explosões, a lava sai e escorre apenas pela ação da gravidade quando possível.
- d) a cada erupção, a cratera aparece em um local distinto da ilha.

Você, como estagiário de geologia, recebeu a incumbência de escrever um programa que:

a) leia uma matriz NxN, que representa as alturas médias dos platôs da ilha (cada platô tem 1

hectometro², 1 hm=100m) em hm, em relação ao nível do mar. Assim, se na posição 2,2 houver o valor 3, isto indicará que neste hectômetro quadrado a altura média é de 300m em relação ao mar.

b) leia uma posição x,y representando a localização da cratera, nesta erupção. Essa localização já está na modalidade desejada, ou seja, variando entre 0 e N-1. Especial cuidado deve ser tomado para o caso em que a cratera já esteja na margem da ilha. Neste caso ela poderá não ter alguns dos 8 vizinhos.

Sabe-se que a lava flui apenas do platô cratera para os platôs vizinhos se estes tiverem uma quota menor (forem menos altos), e que a lava vaza para os 8 vizinhos (nas direções N, NE, E, SE, S, SO, O e NO) à velocidade de 1 platô/hora.

Quando a lava chega nos limites da ilha, ela cai no mar e deixa de ter relevância para este problema.

Supondo que a erupção ocorre às 14hr, seu programa deve calcular e imprimir quantos hectômetros quadrados foram cobertos de lava após 1 hora da erupção, ou seja às 15h00. Deve imprimir também qual a altura mínima da lava nesta hora, em relação ao nível do mar. Note que a quantidade de platôs da resposta deve variar entre 1 (só a cratera) e 9 (A cratera e seus 8 vizinhos).

Para seu entendimento, suponha uma ilha com a configuração

1	1	7	8	2	2	Aqui:			
2	2	3	4	1	9	crat=4.0 =>	p=1	m=3	
3	3	5	5	6	7		2,2	5	2
4	5	6	7	8	2		3,3	8	4
3	3	4	5	6	7		5,5	1	1
3	4	5	2	2	1				

```
#include<iostream>
#define N 6
using namespace std;
int main(){
    int vu[N][N];
    int x,y,i,j,qtd,minimo;
    cout<<"Informe o vulcao: "<<endl;
    for (i=0;i<N;i++){
        for (j=0;j<N;j++){
            cin>>vu[i][j];
        }
    }
    cout<<"Informe a cratera: ";
    cin>>x>>y;
    // 14hr
    qtd=1;
    minimo=999999;
    // 15hr
    for (i=x-1;i<=x+1;i++){
        for (j=y-1;j<=y+1;j++){
            if ((i>=0)&&(i<N)&&(j>=0)&&(j<N)) {
                if (vu[i][j]<vu[x][y]){
                    qtd++;
                }
                if (vu[i][j]<minimo){
                    minimo=vu[i][j];
                }
            }
        }
    }
    cout<<"Platos atingidos: "<<qtd<<endl;
    cout<<"Quota minima: "<<minimo<<endl;
}
```

O truque: Achar os vizinhos de um determinado elemento em uma matriz. Basta subtrair e acrescentar 1 unidade (ou mais, se forem vizinhos mais distantes) nos índices dos elementos a processar. Especial cuidado deve ser tomado nos limites da matriz. O número de vizinhos só é total nos elementos centrais da matriz, ou seja, aqueles que não estão em nenhuma borda.

Para você fazer: No arquivo acima, há 20 vulcões de 10 x 10, abaixo de cada um há uma coordenada da cratera. Você deve processar os 20 vulcões e para cada um deve-se achar a soma dos platôs atingidos e a soma das quotas mínimas em cada vulcão.

$\sum$ platos	$\sum$ quotas _{min}

Sequencia de Fibonacci Os historiadores da matemática quase garantem que a disseminação dos algarismos arábicos (e também do zero, este de origem hindu) na cultura ocidental, vem do livro de Fibonacci, *Liber Abacci* publicado na Itália por volta de 1220.

Na nossa matemática contemporânea, o Fibonacci está associado à célebre sequência de números: 1,1,2,3,5,8,13,21,34,55,... na qual os dois primeiros termos valem 1 e 1 e a partir do terceiro, cada elemento é igual à soma dos dois anteriores.

Neste exercício, você está sendo convidado a desenvolver um gerador de sequências de Fibonacci genérico. A diferença é que os 2 primeiros termos da sequência serão variáveis, embora a regra de construção dos elementos além do segundo permaneça a mesma, isto é, a soma dos dois anteriores.

Eis alguns exemplos destas sequências genéricas:

2,7,9,16,25,41,...
3,7,10,17,27,44,71,115,...
100,200,300,500,800,1300,2100,3400,...
123,456,579,1035,1614,2649,4263,...

Você deve escrever um programa e nele deve escrever uma função de nome fibonacci(), que receba 3 parâmetros: a saber (a,b,q) a é o primeiro termo, b o segundo termo, e q é a ordem do elemento que deve ser calculado. Por exemplo, se q=3 então é o terceiro a ser calculado, se q=7 é o sétimo e em termos genéricos é o q-ésimo termo.

O programa principal deve receber uma quantidade indeterminada de triplas a,b,q e para cada uma calcular os termos de fibonacci pedidos através do parâmetro q.

Note que além de devolver o q-ésimo termo, a função retornar também o termo anterior (o de ordem q-1), já que ambos deverão ser impressos.

Exemplos de uso:

```
Informe a, b e q: 1 1 7
Termo q-1: 8 - Termo q: 13
Informe a, b e q: 2 7 6
Termo q-1: 25 - Termo q: 41
Informe a, b e q: 3 7 7
Termo q-1: 44 - Termo q: 71
Informe a, b e q: 123 456 5
Termo q-1: 1035 - Termo q: 1614
Informe a, b e q: 0,0,0
```

```
#include<iostream>
using namespace std;
int fibo(int &a, int &b, int qual){
    int i,res;
    i=3;
    while (i<=qual){
        res=a+b;
        a=b;
        b=res;
        i++;
    }
    return 0;
}

int main() {
    int aa,bb,qual;
    while (1==1){
        cout<<"Informe a, b e q: ";
        cin>>aa>>bb>>qual;
        if ((aa==0) && (bb==0)){
            break;
        }
        fibo(aa,bb,qual);
        cout<<"X q-1: "<<aa<<" - X q: "<<bb<<endl;
    }
}
```

No arquivo acima, há 20 triplas a,b,q, devendo-se calcular os termos X_{q-1} e X_q e somar os 20 resultados.

$\sum X_{q-1}$	$\sum X_q$

Para você fazer

simétricas	$\sum$ platos	$\sum$ min
X_{q-1}	X_q	////// ////// ////// //////



Truques de programação IV - Python

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F186001.myd

publicado no lugar usual.

Matrizes simétricas Escreva um programa que analise uma matriz e informe se ela é ou não é simétrica. Para esta verificação, os elementos simétricos em relação à diagonal principal devem ser iguais. A matriz precisa ser quadrada e nela todos elementos $m[i][j]$ precisam ser iguais a $m[j][i]$. Por abuso, esta regra pode ser aplicada aos elementos que estão na diagonal principal, já que quem está nela tem $i = j$ e portanto $m[i][j]$ sempre será igual a $m[j][i]$ quando $i = j$. Mas, nada se perde se elementos da diagonal principal simplesmente não forem testados. De acordo com a regra, basta que um único elemento simétrico seja diferente para que a matriz toda o seja. Um possível gabarito:

```
import numpy as np
def f186():
    f=open("c:/p/n/186/f186001_exemplo.myd","r")
    i=ct=0
    m=np.zeros((10,10))
    while i<10:
        for ii in range(10):
            x=f.readline()
            nums=x.split()
            for jj in range(10):
                m[ii,jj]=int(nums[jj])
        sentinela=0
        for ii in range(10):
            for jj in range(10):
                if m[ii,jj]!=m[jj,ii]:
                    sentinela=1
        if sentinela==0:
            ct=ct+1
            i=i+1
    print("Simetricas = ",ct)
```

O truque: Sentinela é uma variável que sinaliza uma determinada condição ao longo de um processamento completo. Usada muitas vezes, como neste contexto, no qual uma única condição perdida em um volume grande de situações e lugares (como um único elemento simétrico diferente) pode determinar uma condição mais geral (toda a matriz deixou de ser simétrica).

Para otimizar o processamento, ao invés de usar os dois ciclos *for* poderíamos ter usado ciclos *while*, incluindo neles (com AND) a condição *sentinela* = 0, para que, uma vez descoberto um elemento diferente a busca fosse encerrada.

Para você fazer Leia no arquivo acima descrito, 10 matrizes numéricas de 10 linhas por 10 colunas estão registradas. O objetivo do aluno deve ser contar quantas são simétricas.

matrizes simétricas:

Vulcão no Havai Como sabemos o Havai está neste momento em processo de manifestação vulcânica intensa. O vulcão Saidebaixo, numa ilha lá, tem algumas características interessantes: a) fica numa ilha que tem platôs quadrados, de altura regular e é deserta. b) sua lava é muito líqüefeita, quase líqüida, saindo da cratera e escorrendo, se possível, para as 8 direções vizinhas. c) não há explosões, a lava sai e escorre apenas pela ação da gravidade quando possível. d) a cada erupção, a cratera aparece em um local

distinto da ilha. Você, como estagiário de geologia, recebeu a incumbência de escrever um programa que: a) leia uma matriz NxN, que representa as alturas médias dos platôs da ilha (cada platô tem 1 hectometro², 1 hm=100m) em hm, em relação ao nível do mar. Assim, se na posição 2,2 houver o valor 3, isto indicará que neste hectômetro quadrado a altura média é de 300m em relação ao mar. b) leia uma posição x,y representando a localização da cratera, nesta erupção. Essa localização já está na modalidade desejada, ou seja, variando entre 0 e N-1. Especial cuidado deve ser tomado para o caso em que a cratera já esteja na margem da ilha. Neste caso ela poderá não ter alguns dos 8 vizinhos. Sabe-se que a lava flui apenas do platô cratera para os platôs vizinhos se estes tiverem uma quota menor (forem menos altos), e que a lava vaza para os 8 vizinhos (nas direções N, NE, E, SE, S, SO, O e NO) à velocidade de 1 platô/hora. Quando a lava chega nos limites da ilha, ela cai no mar e deixa de ter relevância para este problema. Supondo que a erupção ocorre às 14hr, seu programa deve calcular e imprimir quantos hectômetros quadrados foram cobertos de lava após 1 hora da erupção, ou seja às 15h00. Deve imprimir também qual a altura mínima da lava nesta hora, em relação ao nível do mar. Note que a quantidade de platôs da resposta deve variar entre 1 (só a cratera) e 9 (A cratera e seus 8 vizinhos).

Para seu entendimento, suponha uma ilha com a configuração

Aqui:

1	1	7	8	2	2
2	2	3	4	1	9
3	3	5	5	6	7
4	5	6	7	8	2
3	3	4	5	6	7
3	4	5	2	2	1

Para seu entendimento, suponha uma ilha com a configuração

1	1	7	8	2	2
2	2	3	4	1	9
3	3	5	5	6	7
4	5	6	7	8	2
3	3	4	5	6	7
3	4	5	2	2	1

```
i=0
platos=sommin=0
while i<20:
    for ii in range(10):
        x=f.readline()
        nums=x.split()
        for jj in range(10):
            m[ii,jj]=int(nums[jj])
        x=f.readline()
        nums=x.split()
        xx=int(nums[0])-1
        yy=int(nums[1])-1
        qtd=0
        minimo=+999999
        for ii in range(xx-1,xx+2):
            for jj in range(yy-1,yy+2):
                if ii>0 and ii<10 and jj>0 and jj<10:
                    if m[ii,jj]<m[xx,yy]:
                        qtd=qtd+1
                    if (ii!=xx or jj!=yy) and m[ii,jj]<minimo:
                        minimo=m[ii,jj]
        platos=platos+qtd
        sommin=sommin+minimo
        i=i+1
print("Vulcao = ",platos," Minimos = ",sommin)
```

O truque: Achar os vizinhos de um determinado elemento em uma matriz. Basta subtrair e acrescentar 1 unidade (ou mais, se forem vizinhos mais distantes) nos índices dos elementos a processar. Especial cuidado deve ser tomado nos limites da matriz. O número de vizinhos só é total nos elementos centrais da matriz, ou seja, aqueles que não estão em nenhuma borda.

Para você fazer: No arquivo acima, há 20 vulcões de 10 x 10, abaixo de cada um há uma coordenada da cratera. Você deve processar os 20 vulcões e para cada um deve-se achar a soma dos platos atingidos e a soma das quotas mínimas em cada vulcão.

$\sum$ platos	$\sum$ quotasmin

Sequencia de Fibonacci Os historiadores da matemática quase garantem que a disseminação dos algarismos arábicos (e também do zero, este de origem hindu) na cultura ocidental, vem do livro de Fibonacci, *Liber Abacci* publicado na Itália por volta de 1220.

Na nossa matemática contemporânea, o Fibonacci está associado à célebre sequência de números: 1,1,2,3,5,8,13,21,34,55,... na qual os dois pri-

meiros termos valem 1 e 1 e a partir do terceiro, cada elemento é igual à soma dos dois anteriores. Neste exercício, você está sendo convidado a desenvolver um gerador de sequências de Fibonacci genérico. A diferença é que os 2 primeiros termos da sequência serão variáveis, embora a regra de construção dos elementos além do segundo permaneça a mesma, isto é, a soma dos dois anteriores. Eis alguns exemplos destas sequências genéricas:

2,7,9,16,25,41,...
3,7,10,17,27,44,71,115,...
100,200,300,500,800,1300,2100,3400,...
123,456,579,1035,1614,2649,4263,...

Você deve escrever um programa e nele deve escrever uma função de nome fibonacci(), que receba 3 parâmetros: a saber (a,b,q) a é o primeiro termo, b o segundo termo, e q é a ordem do elemento que deve ser calculado. Por exemplo, se q=3 então é o terceiro a ser calculado, se q=7 é o sétimo e em termos genéricos é o q-ésimo termo.

O programa principal deve receber uma quantidade indeterminada de triplas a,b,q e para cada uma calcular os termos de fibonacci pedidos através do parâmetro q.

Note que além de devolver o q-ésimo termo, a função retornar também o termo anterior (o de ordem q-1), já que ambos deverão ser impressos.

```
Exemplos de uso:
Informe a, b e q: 1 1 7
Termo q-1: 8 - Termo q: 13
Informe a, b e q: 2 7 6
Termo q-1: 25 - Termo q: 41
Informe a, b e q: 3 7 7
Termo q-1: 44 - Termo q: 71
Informe a, b e q: 123 456 5
Termo q-1: 1035 - Termo q: 1614
Informe a, b e q: 0,0,0
```

```
def fibo(a,b,qual):
    i=3
    while i<=qual:
        res=a+b
        a=b
        b=res
        i=i+1
    return [a,b]
```

```
...
i=somq=somqm=0
while i<200:
    x=f.readline()
    nums=x.split()
    a=int(nums[0])
    b=int(nums[1])
    qual=int(nums[2])
    zz=fibo(a,b,qual)
    somq=somq+zz[1]
    somqm=somqm+zz[0]
    i=i+1
    print(somqm,somq)
```

No arquivo acima, há 200 triplas a,b,q, devendo-se calcular os termos X_{q-1} e X_q e somar os 200 resultados.

$\sum X_{q-1}$	$\sum X_q$

Para você fazer

simétricas	$\sum$ platos	$\sum$ min
X_{q-1}	X_q	////// ////// ////// ////// ////// ////// ////// //////



Truques de programação V - C++

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F187001.myd

publicado no lugar usual.

Operações com conjuntos Imagine uma matriz de 6 linhas por 13 colunas, na qual: cada linha corresponde a um conjunto de até 12 números. O primeiro elemento de cada linha não é um elemento e sim um contador de elementos válidos naquela linha. As 3 primeiras linhas correspondem aos conjuntos A , B e C e já vêm preenchidas. As outras linhas correspondem aos conjuntos D , E e F e devem ser calculados pelo programa, usando as seguintes regras:
 $D = A \cap B$, $E = D \cup C$ e $F = B \cap C$. Escreva um programa C++ que leia uma matriz deste tipo, atualize as linhas 4.6 e imprima a matriz completa.

...

Truques: O primeiro é o conceito de conjunto de elementos nos quais: i. não pode haver repetição de elementos e ii. A ordem dos elementos não interessa. Adicionalmente, a operação de intersecção ($\cap$) devolve elementos que estão no primeiro e no segundo conjuntos. A união ($\cup$) recupera elementos que estão no primeiro ou no segundo conjuntos.

Um truque adicional é contador de elementos válidos. É usado quando se simula um conjunto de tamanho variável (o conjunto) sobre uma estrutura de dados de tamanho fixo (a linha da matriz).

Para você No arquivo acima, há 100 matrizes (6×13 cada) Você deve processá-las e gerar em cada uma as linhas 4.6. Obtenha a soma das cardinalidades (a quantidade de elementos, ou em termos físicos a primeira coluna de cada linha) dos conjuntos D , E e F .

$\sum t(D)$	$\sum t(E)$	$\sum t(F)$

Determinante O determinante de uma matriz quadrada é um número real que informa algumas características da matriz e que pode e é usado em diversos processamentos sobre a matriz. O determinante de uma matriz 1×1 contendo um único valor é o próprio valor. Já uma matriz 2×2 tem como determinante a expressão $a.d - b.c$. Uma matriz 3×3 tem como determinante a expressão (tirada da regra de Sarrus) $aei + dhc + bfg - (gec + ahf + dbi)$. Para as ordens maiores, existe uma regra (recursiva) que vai ser adotada aqui. Para calcular o determinante de uma matriz de ordem 4, escolhe-se uma linha (ou coluna) qualquer e para todo elemento $m[i,j]$ dessa linha, calcula-se a expressão $m[i,j] \times (-1)^{i+j} \times \det(m')$ onde m' é a matriz resultante quando se extrai de m a linha i e a coluna j .

```
#include<iostream>
#include<fstream>
#include<iomanip>
#define N 4
using namespace std;
void tira(float vai[N][N], float volta[N][N],
        int tam, int qual){
    int i,j,k;
    i=0; k=0;
    while (i<tam){
        if (i==qual) {i++;}
        for (j=1;j<tam;j++){
            volta[k][j-1]=vai[i][j];
        }
        i++;
    }
}
```

```
    }
    k++;
    i++;
}
}
float det(float m[N][N],int tam){
    float x,y;
    float mlin[N][N];
    int k1,k2,i,j,k;
    if (tam<4){
        x=m[0][0]*m[1][1]*m[2][2];
        x=x+m[1][0]*m[2][1]*m[0][2];
        x=x+m[0][1]*m[1][2]*m[2][0];
        y=m[2][0]*m[1][1]*m[0][2];
        y=y+m[0][0]*m[2][1]*m[1][2];
        y=y+m[1][0]*m[0][1]*m[2][2];
        return x-y;
    }
    else {
        x=0;
        i=0;
        while (i<tam){
            tira(m,mlin,tam,i);
            y=det(mlin,tam-1);
            if (((1+i)%2)==0){
                x=x+m[i][0]*-1*y;
            }
            else {
                x=x+m[i][0]*y;
            }
            i++;
        }
        return x;
    }
}
int main() {
    float m[N][N];
    int i,j;
    fstream f;
    float x;
    f.open("c:/p/n/...txt");
    for (i=0;i<N;i++){
        for (j=0;j<N;j++){
            f>>m[i][j];
        }
    }
    for (i=0;i<N;i++){
        for (j=0;j<N;j++){
            cout<<setw(5)<<m[i][j];
        }
        cout<<endl;
    }
    x=det(m,N);
    cout<<"Determinante = "<<x;
}
```

Truque: recursividade Como a definição de determinante usa a recursividade (a solução de um determinante 4 é igual a 3 vezes a solução de um determinante 3...), tem-se o ambiente ideal para a recursividade: i. mesma solução, ii. universo menor e iii. caso básico ($\det=3$).
Para você No arquivo acima, há 40 matrizes quadradas de 6×6 . Ache o determinante delas e conte quantas dessas 40 têm determinante $\neq 0$.

qtd det $\neq 0$

Matriz inversa A matriz inversa surge em inúmeros momentos da álgebra: por exemplo na solução de sistemas lineares. Uma matriz A poderá ter uma matriz inversa chamada A^{-1} que é única e que quando multiplicada matricialmente à matriz original A reproduz a matriz identidade I . Quem é inversa de quem é mera convenção, já que uma é a inversa da outra. A matriz A só terá inversa se o seu determinante for diferente de zero.

```
#include<iostream>
#include<iomanip>
#define ta 5
using namespace std;
int cami(float x[ta][ta], float y[ta][ta]){
    // x é a matriz A e y é a matriz A^-1
    int i,j,k;
    float pivot,alvo;
    for (i=0;i<ta;i++){
        for (j=0;j<ta;j++){
            if (i==j) {
                y[i][j]=1;
            }
            else {
                y[i][j]=0;
            }
        }
    }
}
```

```
    }
}
for (i=0;i<ta;i++){
    for (j=0;j<ta;j++){
        if (j!=i){
            pivot=-x[j][i]/x[i][i];
            for (k=0;k<ta;k++){
                x[j][k]=x[j][k]+x[i][k]*pivot;
                y[j][k]=y[j][k]+y[i][k]*pivot;
            }
        }
    }
    alvo=x[i][i];
    for (k=0;k<ta;k++){
        y[i][k]=y[i][k]/alvo;
        x[i][k]=x[i][k]/alvo;
    }
}
return 0;
}
int main() {
    float mat[ta][ta]={ {...}, {...}, {...}, {...};
    float res[ta][ta];
    int i,j;
    cami(mat,res);
    for (i=0;i<ta;i++){
        for (j=0;j<ta;j++){
            cout<<fixed<<setprecision(5)
                <<res[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

Para você No arquivo acima, há 20 matrizes (10 pares) de 10 linhas por 10 colunas. Você deve contar quantos pares realmente são inversos um do outro.

qtd pares de inversos

Para ler o arquivo Para ler os dados de entrada de um arquivo (ao invés do teclado) em C++ deve-se: i.incluir o pacote `<fstream>`. ii.definir uma variável (digamos f) de tipo `ifstream`. iii. Abrir o arquivo passando a localização do mesmo. iv. fazer $f >> \text{variável}$ (ao invés de $\text{cin} >> \text{variável}$). v. Ao final, fechar o arquivo (`f.close()`). Acompanhe

```
#include<fstream>
ifstream f;
f.open("c:/p/apl/...")
...
f>>v[i];
...
f.close()
```

Para você fazer

$\sum t(D)$	$\sum t(E)$	$\sum t(F)$
q det $\neq 0$	q pares inv	//////// //////// //////// //////// ////////



Truques de programação V - Python

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F187001.myd

publicado no lugar usual.

Operações com conjuntos Imagine uma matriz de 6 linhas por 13 colunas, na qual: cada linha corresponde a um conjunto de até 12 números. O primeiro elemento de cada linha não é um elemento e sim um contador de elementos válidos naquela linha. As 3 primeiras linhas correspondem aos conjuntos *A*, *B* e *C* e já vêm preenchidas. As outras linhas correspondem aos conjuntos *D*, *E* e *F* e devem ser calculados pelo programa, usando as seguintes regras:
 $D = A \cap B$, $E = D \cup C$ e $F = B \cap C$. Escreva um programa Python que leia uma matriz deste tipo, atualize as linhas 4..6 e imprima a matriz completa.

...

Truques: O primeiro é o conceito de conjunto de elementos nos quais: i. não pode haver repetição de elementos e ii. A ordem dos elementos não interessa. Adicionalmente, a operação de intersecção ($\cap$) devolve elementos que estão no primeiro e no segundo conjuntos. A união ($\cup$) recupera elementos que estão no primeiro ou no segundo conjuntos.

Um truque adicional é contador de elementos válidos. É usado quando se simula um conjunto de tamanho variável (o conjunto) sobre uma estrutura de dados de tamanho fixo (a linha da matriz).

```
def f187():
    f=open("c:/p/n/187/f187001_exemplo.myd","r")
    i=ctdd=ctde=ctdf=0
    while i<40:
        m=np.zeros((6,13),int)
        for ii in range(6):
            x=f.readline()
            nums=x.split()
            for jj in range(13):
                m[ii,jj]=int(nums[jj])

        idd=1
        iee=1
        for iaa in range(1,m[0,0]+1):
            for ibb in range(1,m[1,0]+1):
                if m[0,iaa]==m[1,ibb]:
                    m[3,idd]=m[0,iaa]
                    m[4,idd]=m[0,iaa]
                    idd=idd+1
                    iee=iee+1
                    m[3,0]=m[3,0]+1
                    m[4,0]=m[4,0]+1
        for icc in range(1,m[2,0]+1):
            tem=0
            for idd in range(1,m[3,0]+1):
                if m[2,icc]==m[3,idd]:
                    tem=1
            if tem==0:
                m[4,iee]=m[2,icc]
                iee=iee+1
                m[4,0]=m[4,0]+1
        iefe=1
        for ibb in range(1,m[1,0]+1):
            for icc in range(1,m[2,0]+1):
                if m[1,ibb]==m[2,icc]:
                    m[5,iefe]=m[1,ibb]
                    iefe=iefe+1
                    m[5,0]=m[5,0]+1
        ctdd=ctdd+m[3,0]
        ctde=ctde+m[4,0]
        ctdf=ctdf+m[5,0]
        i=i+1
    print("D=",ctdd," E=",ctde," F=",ctdf)
```

Para você No arquivo acima, há 100 matrizes (6×13 cada) Você deve processá-las e gerar em cada uma as linhas 4..6. Obtenha a soma das cardinalidades (a quantidade de elementos, ou em termos físicos a primeira coluna de cada linha) dos conjuntos *D*, *E* e *F*.

$\sum t(D)$	$\sum t(E)$	$\sum t(F)$

Determinante O determinante de uma matriz quadrada é um número real que informa algumas características da matriz e que pode e é usado em diversos processamentos sobre a matriz. O determinante de uma matriz 1 × 1 contendo um único valor é o próprio valor. Já uma matriz 2 × 2 tem como determinante a expressão $a.d - b.c$. Uma matriz 3 × 3 tem como determinante a expressão (tirada da regra de Sarrus) $aei + dhc + bfg - (gec + ahf + dbi)$. Para as ordens maiores, existe uma regra (recursiva) que vai ser adotada aqui. Para calcular o determinante de uma matriz de ordem 4, escolha-se uma linha (ou coluna) qualquer e para todo elemento $m[i,j]$ dessa linha, calcula-se a expressão $m[i,j] \times (-1)^{i+j} \times det(m')$ onde m' é a matriz resultante quando se extrai de *m* a linha *i* e a coluna *j*.

```
def tira(x,qual):
    ta=len(x)-1
    z=np.zeros((ta,ta),int)
    i=k=0
    while i<len(x):
        if i==qual:
            i=i+1
            continue
        for j in range(1,len(x)):
            z[k,j-1]=x[i,j]
            k=k+1
            i=i+1
    return z

def det(m):
    if len(m)==3:
        x=(m[0,0]*m[1,1]*m[2,2])+(m[1,0]*m[2,1]*m[0,2])+(m[0,1]*m[1,2]*m[2,0])
        y=(m[2,0]*m[1,1]*m[0,2])+(m[0,0]*m[2,1]*m[1,2])+(m[1,0]*m[0,1]*m[2,2])
        return x-y
    else:
        x=0
        i=0
        while i<len(m):
            z=tira(m,i)
            y=det(z)
            if ((1+i)%2)==0:
                x=x+m[i,0]*-1*y
            else:
                x=x+m[i,0]*y
            i=i+1
        return x

int main() {
    float m[N][N];
    int i,j;
    fstream f;
    float x;
    f.open("c:/p/n/...txt");
    for (i=0;i<N;i++){
        for (j=0;j<N;j++){
            f>>m[i][j];
        }
    }
    for (i=0;i<N;i++){
        for (j=0;j<N;j++){
            cout<<setw(5)<<m[i][j];
        }
        cout<<endl;
    }
    x=det(m,N);
    cout<<"Determinante = "<<x;
}
```

Truque: recursividade Como a definição de determinante usa a recursividade (a solução de um determinante 4 é igual a 3 vezes a solução de um determinante 3...), tem-se o ambiente ideal para a recursividade: i. mesma solução, ii. universo menor e iii. caso básico (det=3).

Para você No arquivo acima, há 40 matrizes quadradas de 6 × 6. Ache o determinante delas e conte quantas dessas 40 têm determinante ≠ 0.

qtd det ≠ 0

Matriz inversa A matriz inversa surge em inúmeros momentos da álgebra: por exemplo na solução de sistemas lineares. Uma matriz *A* poderá ter uma matriz inversa chamada A^{-1} que é única e que quando multiplicada matricialmente à matriz original *A* reproduz a matriz identidade *I*. Quem é inversa de quem é mera convenção, já que uma é a inversa da outra. A matriz *A* só terá inversa se o seu determinante for diferente de zero.

```
def cami(m):
    y=np.zeros((len(m),len(m)),float)
    for i in range(10):
        for j in range(10):
            if i==j:
                y[i,j]=1.0
            else:
                y[i,j]=0.0
    for i in range(10):
        for j in range(10):
            if j!=i:
                pivot=-m[j,i]/m[i,i]
                for k in range(10):
                    m[j,k]=m[j,k]+m[i,k]*pivot
                    y[j,k]=y[j,k]+y[i,k]*pivot
            alvo=m[i][i]
        for k in range(10):
            y[i,k]=y[i,k]/alvo*1.0
            m[i,k]=m[i,k]/alvo*1.0
    return y
i=ctm=0
while i<40:
    m=np.zeros((6,6),int)
    for ii in range(6):
        x=f.readline()
        nums=x.split()
        for jj in range(6):
            m[ii,jj]=int(nums[jj])
    xx=det(m)
    if xx!=0:
        ctm=ctm+1
        i=i+1
    print("Dets diferentes de zero = ",ctm)
    i=0
    ctinv=0
    while i<10:
        ma=np.zeros((10,10),float)
        mb=np.zeros((10,10),float)
        mc=np.zeros((10,10),float)
        for ii in range(10):
            x=f.readline()
            nums=x.split()
            for jj in range(10):
                ma[ii,jj]=float(nums[jj])
        for ii in range(10):
            x=f.readline()
            nums=x.split()
            for jj in range(10):
                mb[ii,jj]=float(nums[jj])
        naoe=0
        mc=cami(ma)
        for ii in range(10):
            for jj in range(10):
                if abs(mb[ii,jj]-mc[ii,jj])>0.2:
                    naoe=1
        if naoe==0:
            ctinv=ctinv+1
            i=i+1
    print("Quantas mat inv diferentes=",ctinv)
```

Para você No arquivo acima, há 20 matrizes (10 pares) de 10 linhas por 10 colunas. Você deve contar quantos pares realmente são inversos um do outro.

qtd pares de inversos

Truques de programação VI - C++

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F188001.myd

publicado no lugar usual.

Ponto central Este problema surge em diversos contextos (ainda mais nos dias de hoje, com a popularização da geolocalização e georeferência via GPS). Trata-se de uma coleção arbitrária de pontos, devendo-se escolher o ponto central do conjunto. A definição formal sugere escolher o ponto que tiver o menor raio de cobertura a todos os demais. Trocando em miúdos, para cada ponto deve-se calcular a distância ao mais distante. O ponto que tiver o menor maior raio é o centro.

Truque: Usa-se aqui o Teorema de Pitágoras para descobrir distâncias, já que estas sempre ou quase sempre virão na forma de coordenadas x, y o que sugere imediatamente um triângulo retângulo.

Para fazer: Há no arquivo acima descrito 50 conjuntos de 10 pares de pontos cada um. Os pontos estão em 2 linhas: na primeira 10 abscissas e na segunda 10 ordenadas, correspondendo o conjunto a 10 pontos. Para cada conjunto, o seu programa deve localizar o ponto central numerando-o de 1 a 10. A resposta à busca é a soma desses 50 números.

$\sum$ nums centro

Histograma Um histograma é uma representação gráfica de algum fenômeno. Feito em colunas, cada uma delas é associada ao valor numérico de alguma medida e a comparação visual imediatamente dá uma boa idéia do comportamento do fenômeno.

Veja-se uma definição: Escreva um programa em C++ para ler um vetor de inteiros de tamanho N (N indicado via `#define`) que representa um histograma normalizado, com valores variando de 1 a N . O histograma é uma representação gráfica em colunas dos valores do vetor, ou seja, uma distribuição de frequências. Seu programa deve imprimir o histograma a partir do vetor de entrada, conforme exemplos abaixo.

Exemplo de execução (com $N=5$):

Digite vetor:

1 2 3 4 5

Histograma:

```

  @
  @ @
 @ @ @
@ @ @ @
@ @ @ @ @
```

Digite vetor:

1 2 5 2 1

Histograma:

```

  @
  @
 @ @ @
@ @ @ @ @
```

```
#include<iostream>
#define N 5
using namespace std;
int main() {
    int v[N];
    int i,j;
    cout<<"Digite o vetor: "<<endl;
    for (i=0;i<N;i++){
        cin>>v[i];
    }
    for (i=0;i<N;i++){
```

```
        for(j=0;j<N;j++){
            if (v[j]>=N-i){
                cout<<"@";
            }
            else{
                cout<<" ";
            }
        }
        cout<<endl;
    }
}
```

O truque: É imprimir um gráfico a partir de informações numéricas. A solução aqui é imaginar o espaço no papel como uma matriz, que pode ser preenchida com espaço ou com algum caracter, a depender de alguma condição.

Para você No arquivo acima, há 50 vetores de diversos tamanhos (o tamanho do vetor é primeiro número e ele deve ser desconsiderado no processamento do vetor) e que devem gerar histogramas. Totalize em todos eles quantas vezes o comando de impressão de um espaço em branco foi acionado.

$\sum$ espaços

Xadrez O xadrez é um universo riquíssimo, inclusive no mundo da programação. Há inúmeras coisas que podem ser programadas nesse universo. Veja-se aqui uma definição: O xadrez é jogado em um tabuleiro quadriculado 8×8 de casas brancas e pretas alternadas, sendo que a casa superior esquerda (linha 0, coluna 0) é branca. No xadrez, um jogador controla as peças brancas e o outro as pretas. Embora o objetivo seja capturar o rei adversário, é comum atribuir pontos às peças que cada jogador tem no tabuleiro a fim de estabelecer uma medida de desempenho do jogador. A seguinte pontuação pode ser utilizada: peão: 1, cavalo: 3, bispo: 3, torre: 5, dama: 9, rei: 0. Considere um método de pontuação que contabilize apenas os pontos das peças que têm A MESMA COR da casa em que está no tabuleiro. Por exemplo, um bispo preto em uma casa branca tem valor 0 ao invés de 3, enquanto uma torre branca em uma casa branca possui a pontuação normal de 5. A figura abaixo mostra uma situação de jogo na qual o jogador de brancas tem 22 pontos e o jogador de pretas tem 23 pontos.



Faça um programa em C++ que contabilize os pontos de cada jogador. Seu programa deve ler uma matriz 8×8 representando uma configuração do tabuleiro de xadrez onde cada elemento é um número inteiro que pode ser 0, indicando que não há peça naquela posição, ou um número de 2 dígitos onde o primeiro dígito indica a cor da peça que lá está (1 se for branca e 2 se for preta) e o segundo dígito indica o valor da peça. Por exemplo, 13 indica que naquela posição há um cavalo/bispo branco, já o número 20 indica que a posição tem o rei preto, e assim por diante.

Seu programa deve calcular a pontuação do jogador de brancas e do jogador de pretas e mostrar essa pontuação. Lembre-se que a casa inferior direita de cada jogador é branca.

```
25 0 0 0 0 25 20 0
21 0 29 23 23 21 21 21
0 21 21 23 21 0 0 0
0 0 0 0 0 23 0 0
0 0 19 11 0 0 0 0
0 0 13 0 11 13 13 0
11 11 0 0 13 11 11 11
```

```
0 0 15 0 10 0 0 15
Pontuacao Brancas: 22
Pontuacao Pretas: 23

#include<iostream>
#include<iomanip>
#include<fstream>
using namespace std;
void le_tab(int t[8][8]){
    int i,j;
    ifstream f;
    f.open("c:/ufpr/ci208/p2019.txt");
    for (i=0;i<8;i++){
        for (j=0;j<8;j++){
            f>>t[i][j];
        }
    }
}
void imp_tab(int t[8][8]){
    int i,j;
    for (i=0;i<8;i++){
        for (j=0;j<8;j++){
            cout<<setw(3)<<t[i][j];
        }
        cout<<endl;
    }
}
int main(){
    int t[8][8];
    int i,j,cor,pec,pb,pn;
    le_tab(t);
    imp_tab(t);
    pb=0;
    pn=0;
    for(i=0;i<8;i++){
        for(j=0;j<8;j++){
            cor=t[i][j]/10;
            pec=t[i][j]%10;
            if (((i+j)%2)==0)&&(cor==1)){pb=pb+pec;}
            if (((i+j)%2)==1)&&(cor==2)){pn=pn+pec;}
        }
    }
    cout<<"Pontuacao brancas: "<<pb<<endl;
    cout<<"Pontuacao negras : "<<pn<<endl;
    imp_tab(t);
}
```

Para você No arquivo acima, há 100 tabuleiros de 8 linhas por 8 colunas. Devem ser totalizados os pontos de brancas e de negras nos 100 casos.

$\sum$ brancas	$\sum$ negras

Para ler o arquivo Para ler os dados de entrada de um arquivo (ao invés do teclado) em C++ deve-se: i.incluir o pacote `<fstream>`. ii.definir uma variável (digamos `f`) de tipo `ifstream`. iii. Abrir o arquivo passando a localização do mesmo. iv. fazer `f>>variável` (ao invés de `cin>>variável`). v. Ao final, fechar o arquivo (`f.close()`). Acompanhe

```
#include<fstream>
ifstream f;
f.open("c:/apl/...")
...
f>>v[i];
...
f.close()
```

Para você fazer

$\sum$ nums centro	$\sum$ espaços
$\sum$ brancas	$\sum$ negras



Truques de programação VI - Python

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

Para seu processamento, você deve ler o arquivo

F188001.myd

publicado no lugar usual.

Ponto central Este problema surge em diversos contextos (ainda mais nos dias de hoje, com a popularização da geolocalização e georeferência via GPS). Trata-se de uma coleção arbitrária de pontos, devendo-se escolher o ponto central do conjunto. A definição formal sugere escolher o ponto que tiver o menor raio de cobertura a todos os demais. Trocando em miúdos, para cada ponto deve-se calcular a distância ao mais distante. O ponto que tiver o menor maior raio é o centro. **Truque:** Usa-se aqui o Teorema de Pitágoras para descobrir distâncias, já que estas sempre ou quase sempre virão na forma de coordenadas *x, y* o que sugere imediatamente um triângulo retângulo.

```
def f188():
    f=open("c:/p/n/188/f188001_ex2.myd","r")
    ctd=0
    i=0
    maximo=np.zeros(10,float)
    raios=np.zeros(10,float)
    while i<50:
        pts=np.zeros((10,2),int)
        alfa=f.readline()
        nums=alfa.split()
        for j in range(10):
            pts[j,0]=float(nums[j])
        alfa=f.readline()
        nums=alfa.split()
        for j in range(10):
            pts[j,1]=float(nums[j])
        for ii in range(10):
            for jj in range(10):
                raios[jj]=pitag(pts[ii,0],
                                pts[ii,1],pts[jj,0],pts[jj,1])
            maximo[ii]=-999999999999.0
        for jj in range(10):
            if raios[jj]>maximo[ii]:
                maximo[ii]=raios[jj]
        menor=999999999999.0
        for ii in range(10):
            if maximo[ii]<menor:
                menor=maximo[ii]
                imenor=ii
        ctd=ctd+imenor+1
        i=i+1
    print("So= ",ctd)
```

Para fazer: Há no arquivo acima descrito 50 conjuntos de 10 pares de pontos cada um. Os pontos estão em 2 linhas: na primeira 10 abscissas e na segunda 10 ordenadas, correspondendo o conjunto a 10 pontos. Para cada conjunto, o seu programa deve localizar o ponto central numerando-o de 1 a 10. A resposta à busca é a soma desses 50 números.

∑ nums centro

Histograma Um histograma é uma representação gráfica de algum fenômeno. Feito em colunas, cada uma delas é associada ao valor numérico de alguma medida e a comparação visual imediatamente dá uma boa idéia do comportamento do fenômeno. Veja-se uma definição: Escreva um programa em Python para ler um vetor de inteiros de tamanho N que representa um histograma normalizado,

com valores variando de 1 a N. O histograma é uma representação gráfica em colunas dos valores do vetor, ou seja, uma distribuição de frequências. Seu programa deve imprimir o histograma a partir do vetor de entrada, conforme exemplos abaixo.

Exemplo de execução (com N=5):

```
Digite vetor:
1 2 3 4 5
Histograma:
@
@ @
@ @ @
@ @ @ @
@ @ @ @ @

Digite vetor:
1 2 5 2 1
Histograma:
@
@
@
@ @ @
@ @ @ @ @

v=[0]*100
i=ctb=0
while i<50:
    x=f.readline()
    nums=x.split()
    v[0]=int(nums[0])
    for z in range(v[0]):
        v[z+1]=int(nums[z+1])
    for ii in range(v[0]):
        for j in range(v[0]):
            if v[j+1]<(v[0]-ii):
                ctb=ctb+1
        i=i+1
    print("qtd brancos=",ctb)
```

O truque: É imprimir um gráfico a partir de informações numéricas. A solução aqui é imaginar o espaço no papel como uma matriz, que pode ser preenchida com espaço ou com algum caracter, a depender de alguma condição. **Para você** No arquivo acima, há 50 vetores de diversos tamanhos (o tamanho do vetor é primeiro número e ele deve ser desconsiderado no processamento do vetor) e que devem gerar histogramas. Totalize em todos eles quantas vezes o comando de impressão de um espaço em branco foi acionado.

∑ espaços

Xadrez O xadrez é um universo riquíssimo, inclusive no mundo da programação. Há inúmeras coisas que podem ser programadas nesse universo. Veja-se aqui uma definição: O xadrez é jogado em um tabuleiro quadriculado 8 x 8 de casas brancas e pretas alternadas, sendo que a casa superior esquerda (linha 0, coluna 0) é branca. No xadrez, um jogador controla as peças brancas e o outro as pretas. Embora o objetivo seja capturar o rei adversário, é comum atribuir pontos às peças que cada jogador tem no tabuleiro a fim de estabelecer uma medida de desempenho do jogador. A seguinte pontuação pode ser utilizada: peão: 1, cavalo: 3, bispo: 3, torre: 5, dama: 9, rei: 0. Considere um método de pontuação que contabilize apenas os pontos das peças que têm A MESMA COR da casa em que está no tabuleiro. Por exemplo, um bispo preto em uma casa branca tem valor 0 ao invés de 3, enquanto uma torre branca em uma casa branca possui a pontuação normal de 5. A figura abaixo mostra uma situação de jogo na qual o jogador de brancas tem 22 pontos e o jogador de pretas tem 23 pontos.



Faça um programa que contabilize os pontos de cada jogador. Seu programa deve ler uma matriz 8x8 representando uma configuração do tabuleiro de xadrez onde cada elemento é um número inteiro que pode ser 0, indicando que não há peça naquela posição, ou um número de 2 dígitos onde o primeiro dígito indica a cor da peça que lá está (1 se for branca e 2 se for preta) e o segundo dígito indica o valor da peça. Por exemplo, 13 indica que naquela posição há um cavalo/bispo branco, já o número 20 indica que a posição tem o rei preto, e assim por diante.

Seu programa deve calcular a pontuação do jogador de brancas e do jogador de pretas e mostrar essa pontuação. Lembre-se que a casa inferior direita de cada jogador é branca.

```
25 0 0 0 0 25 20 0
21 0 29 23 23 21 21 21
0 21 21 23 21 0 0 0
0 0 0 0 23 0 0 0
0 0 19 11 0 0 0 0
0 0 13 0 11 13 13 0
11 11 0 0 13 11 11 11
0 0 15 0 10 0 0 15
Pontuacao Brancas: 22
Pontuacao Pretas: 23
```

```
t=np.zeros((8,8),int)
i=spb=spn=0
while i<100:
    for jj in range(8):
        x=f.readline()
        nums=x.split()
        for kk in range(8):
            t[jj,kk]=int(nums[kk])
    pb=pb+0
    for ii in range(8):
        for jj in range(8):
            cor=t[ii,jj]/10
            pec=t[ii,jj]%10
            if ((ii+jj)%2==0 and cor==1:
                pb=pb+pec
            if ((ii+jj)%2==1 and cor==2:
                pn=pn+pec
            spb=spb+pb
            spn=spn+pn
            i=i+1
    print("SPB=",spb," SPN=",spn)
```

Para você No arquivo acima, há 100 tabuleiros de 8 linhas por 8 colunas. Devem ser totalizados os pontos de brancas e de negras nos 100 casos.

∑ brancas	∑ negras

Para você fazer

∑ nums centro	∑ espaços
∑ brancas	∑ negras



Truques de programação VII - C++

Nesta folha você deve resolver alguns exercícios de programação. Cada um deles sugere um truque que quando apreendido pode ser usado em inúmeros outros problemas parecidos ou não.

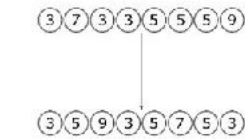
Para seu processamento, você deve ler o arquivo

F189001.myd

publicado no lugar usual.

Bolas trocadas Este exercício trazido da Olimpíada Brasileira de Informática pede que você leia 8 bolas e tente trocá-las de lugar de modo que duas bolas vizinhas não tenham o mesmo número. com a palavra o site da OBI:

Temos oito bolas, colocadas lado a lado em uma sequência. Cada bola tem um número impresso, que pode ter valor de 0 até 9. Queremos trocar algumas bolas de posição na sequência de modo que nenhum par de bolas vizinhas na sequência tenha o mesmo número. Quer dizer, não pode haver duas bolas, uma ao lado da outra, com o mesmo número. A figura ao lado mostra um exemplo para o qual isso foi possível. Mas será que sempre é possível? Seu programa deve decidir se é ou não possível obter uma sequência em que não haja bolas vizinhas com o mesmo número.



Entrada A única linha da entrada contém uma sequência de oito inteiros B_i , para $1 \leq i \leq 8$, representando os números impressos em cada bola da sequência.

Saída Imprima uma linha contendo o caractere "S" se for possível trocar bolas de posição e obter a sequência sem bolas vizinhas com o mesmo número; ou o caractere "N" se não for possível.

Restrições B_i é um inteiro entre 0 e 9, inclusive.

Exemplos

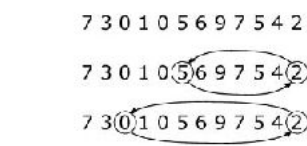
Entrada
3 7 3 3 5 5 5 9
Saída
S

Entrada
8 3 8 8 8 8 8 0
Saída
N

Para Fazer No arquivo acima há 100 conjuntos de bolas numeradas. Você deve processar esse conjunto e contar quantas vezes será possível efetuar as trocas de posição. Em outras palavras, deve contar quantas vezes a função responderá "S".



Cinco Considere um número decimal não divisível por 5. Queremos fazer exatamente uma operação de troca entre os dígitos de duas posições distintas para obter um número que seja divisível por 5. Quer dizer, precisamos escolher duas posições distintas e trocar os dígitos dessas duas posições. Mas queremos que o número resultante após a troca seja o maior número divisível por 5 possível.



Veja o exemplo da figura, 730105697542, que não é divisível por 5. Podemos fazer a primeira troca ilustrada e obter 730105697542, que é divisível por 5. Mas, se fizermos a segunda troca ilustrada na figura, vamos obter um número divisível por 5 ainda maior, 732105697540.

Dados os dígitos decimais de um número na entrada, não divisível por 5, seu programa deve imprimir os dígitos decimais do maior número divisível por 5 que pode ser obtido com exatamente uma troca de dígitos entre duas posições distintas. Caso não seja possível obter um número divisível por 5, imprima apenas -1.

Entrada A primeira linha da entrada contém um inteiro N , indicando quantos dígitos decimais tem o número não divisível por 5. A segunda linha contém N inteiros D_i , $1 \leq i \leq N$, representando os dígitos decimais do número em questão.

Saída Imprima uma linha contendo N inteiros representando os dígitos decimais do maior número divisível por 5 que pode ser obtido com exatamente uma troca de dígitos entre duas posições distintas. Caso não seja possível obter um número divisível por 5, imprima apenas -1.

Restrições $2 \leq N \leq 1000$ e D_i é um inteiro entre 0 e 9, inclusive.

Exemplos
Entrada
12
7 3 0 1 0 5 6 9 7 5 4 2
Saída
7 3 2 1 0 5 6 9 7 5 4 0

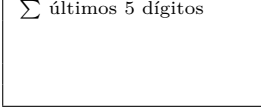
Entrada
5
7 4 1 2 9
Saída
-1

Entrada
8
0 0 7 8 4 5 3 1
Saída
1 0 7 8 4 5 3 0

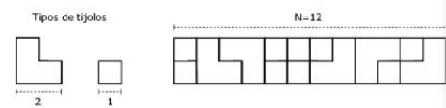
Entrada
10
6 5 0 5 0 4 5 3 4 4
Saída
6 5 4 5 0 4 5 3 4 0

Entrada
7
9 7 4 5 3 5 2
Saída
9 7 4 5 3 2 5

Para fazer No arquivo acima há 100 casos de números para você efetuar a troca de dígitos. O resultado buscado é a soma comum dos números formados pelos últimos 5 dígitos (à direita) dos resultados obtidos.

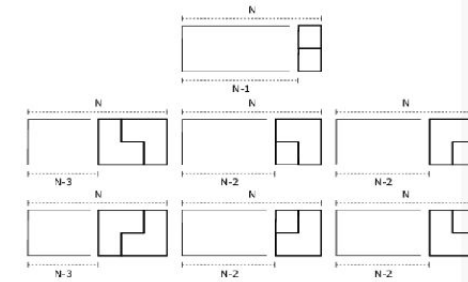


Muro Com a palavra a OBI: Nós temos dois tipos de tijolos, como mostrado na parte esquerda da figura abaixo. A ideia é construir uma mureta de altura 2 e comprimento N . A parte direita da figura ilustra uma forma de usar os dois tipos de tijolos para construir uma mureta de comprimento $N=12$.



Precisamos saber quantas formas distintas existem de construir a mureta com esses dois tipos de tijolos. Para isso, já temos duas observações: qualquer mureta de comprimento N vai terminar de uma das sete maneiras ilustradas abaixo e; o

número de formas distintas de construir uma mureta de comprimento 2, 1 e 0 é, respectivamente, 5, 1 e 1 (sim! Existe uma forma de construir a mureta de comprimento 0: usar nenhum tijolo).



Dado N , seu programa deve computar o número de formas distintas de construir uma mureta de comprimento N . Como esse número pode ser muito grande, seu programa deve imprimir o resto da divisão dele por $10^9 + 7$.

Entrada A única linha da entrada contém um inteiro N , representando o comprimento da mureta.

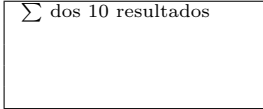
Saída Imprima uma linha contendo um inteiro, o número de formas distintas de construir a mureta com os dois tipos de tijolos. Imprima o resto da divisão desse número por $10^9 + 7$.

Restrições $0 \leq N \leq 10^4$.

Exemplos
Entrada
2
Saída
6
Entrada
5
Saída
241

Entrada
11
Saída
36543
Entrada
8712
Saída
844673301

Para Fazer No arquivo acima há 10 valores de n (um em cada linha). Você deve calcular de quantas maneiras pode ser construído um muro de comprimento igual a n e somar os resultados, sempre lembrando de tomar o módulo $10^9 + 7$ porque esses números crescem assustadoramente.



Dica Este é um problema de combinatória que pode ser resolvido usando-se relações de recorrência. Note que se você tiver um muro de comprimento N , ele poderá ser construído juntando-se um muro de comprimento $N - 1$ a um conjunto de 1 tijolos. Outra possibilidade é juntar uma das 4 possibilidades de 2 tijolos ao lado de um muro de comprimento $N - 2$. E finalmente, há a possibilidade de juntar um muro de $N - 3$ a uma das 2 possibilidades. Daqui:

$$F_N = F_{N-1} + 4 \times F_{N-2} + 2 \times F_{N-3}$$

. Dos dados acima: $F_0 = 1$, $F_1 = 1$, $F_2 = 5$, e agora
 $F_3 = F_2 + 4F_1 + 2F_0 = 5 + 4 + 2 = 11$
 $F_4 = F_3 + 4F_2 + 2F_1 = 11 + 20 + 2 = 33$
 $F_5 = F_4 + 4F_3 + 2F_2 = 33 + 44 + 10 = 87$
 $F_6 = F_5 + 4F_4 + 2F_3 = 87 + 132 + 22 = 241$
Note que se você quiser estudar estes 241 possibilidades o professor tem um mapa com todas elas.

Para você fazer

Σ "S"	Σ ults 5	muro



Palíndromes Uma cadeia de caracteres é chamada de palíndrome pura se a sequência de caracteres da esquerda para a direita é igual à sequência de caracteres da direita para a esquerda (uma outra definição é que o primeiro caractere da cadeia deve ser igual ao último caractere, o segundo caractere seja igual ao penúltimo caractere, o terceiro caractere seja igual ao antepenúltimo caractere, e assim por diante). Por exemplo, as cadeias de caracteres ‘MIM’ , ‘AXXA’ e ‘ANANAGANANA’ são palíndromes puras. Uma outra definição de palíndrome utiliza comparações entre caracteres, considerando que os caracteres são ordenados crescentemente de A até Z, ou seja, A < B < C < ... < Z. Uma cadeia de caracteres é chamada de palíndrome quebrada se a sequência de resultados da comparação entre o primeiro e o segundo caracteres é igual ao resultado da comparação entre o último e o penúltimo caractere, e o resultado da comparação entre o segundo e o terceiro caracteres é igual ao resultado da comparação entre o penúltimo e o antepenúltimo caracteres, e assim por diante. Por exemplo, as cadeias ‘MIN’ e ‘ASSOCFUUS’ são palíndromes quebradas. Obviamente, toda cadeia que é palíndrome pura é também palíndrome quebrada.

Exemplos: A palavra ‘OTO’ é um palíndrome puro (e também quebrado) enquanto a palavra ‘TECLADO’ é um palíndrome quebrado pois: T>E e O>D; E>C e D>A além de C<L e A<L.

Q01: Considere as seguintes cadeias:

(i) BIT

(ii) AAAEAEAEAAA

(iii) SUBINOONIBUS

(iv) TIPOERROTUDO

(v) ERAREHUMANO

Qual das seguintes afirmações é correta?

a) (i) e (ii) são palíndromes puras.

b) (ii) não é palíndrome pura.

c) (iii) é palíndrome pura e (iv) é palíndrome quebrada.

d) (v) é palíndrome quebrada.

e) (iv) e (v) são palíndromes quebradas

Q02: Quantos palíndromes quebrados há nesta lista: TOTAL, ZIQUIZIRA, VIZINHO e IGUAL

a) 2

b) 1

c) 3

d) 4

e) 0

Roland Garros No torneio de Roland Garros, um dos mais tradicionais torneios de tênis do mundo, realizado em Paris, participam n tenistas. Em cada partida, participam dois jogadores, sendo que o vencedor passa para a próxima fase, e o perdedor é eliminado do torneio. A cada rodada, os tenistas que ainda continuam no torneio participam de exatamente uma partida.

Q03: sendo 64 jogadores, quantas rodadas existem durante o torneio?

a) 7

b) 16

c) 8

d) nenhuma das alternativas apresentadas

e) 5

Q04: Sendo 128 jogadores, qual o número total de partidas deste torneio?

a) 128

b) 64

c) nenhuma das alternativas apresentadas

d) 65

e) 127

Cavalo O jogo de xadrez, além de ser um jogo que exige bastante raciocínio, é uma ótima fonte de quebra-cabeças. Esta tarefa trata de um destes quebra-cabeças, envolvendo os movimentos de uma de suas peças, o cavalo. Os movimentos do cavalo são ditos em ‘L’, pois ele sempre deve andar duas casas em uma direção e uma casa na direção perpendicular. A figura acima ilustra os possíveis movimentos do cavalo, onde o caractere representando o cavalo indica a posição inicial e as letras de A a H representam as possíveis posições finais. É importante notar que o cavalo é a única peça que pode saltar sobre outras peças do xadrez. Note ainda que na representação que usamos não distinguimos casas brancas de casas pretas no tabuleiro. Usando os movimentos do cavalo, você deve determinar qual o número mínimo de movimentos do cavalo para ir de uma casa Início até uma casa Final, sendo proibido que o cavalo ‘pare’ sobre algumas casas especificadas durante a sequência de movimentos. São fornecidas várias configurações do tabuleiro (8 por 8 casas), com as casas marcadas com os seguintes valores: - I: início - F: Final - X: proibido. Exemplo:

I			X	2			
		1	X			3	
X	X	X	X	X	X	X	X
	2		X				F
			X				
3		3	X				
			X				
			X				

A maneira mais fácil de resolver este problema é numerar seqüencialmente as posições que podem ser alcançadas pelo cavalo. Inicialmente, colocamos 1 em cada posição que o cavalo pode alcançar a partir da posição inicial. Depois numeramos com 2 todas as posições ainda não marcadas que podem ser alcançadas a partir das posições numeradas com 1, e assim sucessivamente até alcançar a posição final.

Q05:

Q06:

Biblioteca Exatamente nove revistas devem ser expostas em uma das prateleiras da biblioteca da escola. Vamos chamar a posição mais à esquerda

de primeira, e a mais à direita de nona posição. Das nove revistas, quatro são sobre ciência, três são sobre esportes, e duas são sobre culinária. As quatro revistas sobre ciência devem ser colocados uma vizinha à outra, e as duas revistas sobre culinária também devem ser dispostas uma vizinha à outra (ou seja, uma ao lado da outra). As três revistas sobre esportes não precisam estar colocadas uma vizinha à outra.

Q07: Se a sexta posição é ocupada por uma revista de ciência, e a oitava posição é ocupada por uma revista de esportes, qual das seguintes posições deve ser ocupada por uma revista de culinária?

a) A nona

b) A primeira

c) A segunda

d) A quarta

e) A terceira

Q08: Se uma revista de esportes ocupa a quinta posição na prateleira, e uma revista de ciência ocupa a nona posição, qual dos seguintes pares de posições devem ter revistas de tipos diferentes?

a) A primeira e a segunda

b) A terceira e a quinta

c) A terceira e a quarta

d) A segunda e a quarta

e) A segunda e a terceira

Salas de Aulas Uma escola conta com sete salas de aula que devem ser pintadas. As salas estão localizadas em um único lado do corredor, sendo numeradas seqüencialmente, de um a sete, da frente para os fundos do prédio.

sala	sala	sala	sala	sala	sala	sala	Entrada
7	6	5	4	3	2	1	

Cada sala de aula deve ser pintada utilizando uma única cor, de acordo com as seguintes restrições:

- Duas salas devem ser pintadas de branco; duas salas devem ser pintadas de azul; duas salas devem ser pintadas de verde; e uma sala deve ser pintada de amarelo.

- As duas salas pintadas de verde devem ser vizinhas (contíguas).

- As duas salas pintadas de azul não podem ser vizinhas (contíguas).

- A sala pintada de amarelo não pode ser vizinha de uma sala pintada de branco.

- A sala número 3 deve ser pintada de branco.

Q09: Se a sala 5 for pintada de branco, qual das seguintes alternativas pode ser verdadeira?

a) Sala 4 é pintada de amarelo.

b) Sala 1 é pintada de azul.

c) Sala 2 é pintada de amarelo.

d) Sala 6 é pintada de azul.

e) Sala 4 é pintada de verde.

Q10: Se a sala 4 for pintada de branco e uma sala que é pintada de verde for vizinha de uma sala que é pintada de branco, qual das seguintes alternativas precisa ser verdadeira?

a) Sala 5 será pintada de azul.

b) Sala 1 será pintada de amarelo.

c) Sala 6 será pintada de amarelo.

d) Sala 7 será pintada de azul.

e) Sala 1 será pintada de verde.

Responda a opção de cada questão										
1	2	3	4	5	6	7	8	9	10	1

Exemplo de Dados

5
 3 9 14 3
 6 13 17 6
 4 16 12 11
 20 15 25 9
 23 11 29 6

Resposta: 203 unidades de área

Q04: Considere o conjunto de mapas abaixo:

3
 2 6 7 2
 4 15 13 9
 8 11 17 6

A área total coberta pelas fotografias é:

a) 119 b) 101 c) 135 d) 109 e) 123

Q05: Considere o conjunto de mapas abaixo:

7
 2 7 7 2
 2 16 13 8
 5 13 16 5
 9 11 19 3
 9 19 18 14
 17 19 25 13
 19 18 23 14

A área total coberta pelas fotografias é:

a) 374 b) 287 c) 291 d) 303 e) 275



Responda a opção de cada questão

1	2	3	4	5

1

Supermercado Um supermercado iniciou um serviço de entregas das compras de seus clientes. O serviço tem duas categorias, prioritária (quando a compra inclui alimentos congelados) e ordinária (quando a compra não inclui congelados). Seis clientes, identificados por K, L, M, O, P e S, já solicitaram o serviço no primeiro dia de funcionamento. A entrega das compras (dos dois tipos) é sujeita às seguintes condições:

- Independentemente do tipo (ordinária ou prioritária), compras para os clientes P e S devem ser entregues antes das compras serem entregues para o cliente M.
- Independentemente do tipo (ordinária ou prioritária), compras para os clientes L e K devem ser entregues antes das compras serem entregues para o cliente S.
- Desde que as duas restrições acima permitam, compras para clientes que possuam entrega prioritária devem ser entregues antes de compras para clientes que não tenham entrega prioritária.

Q01: Se os clientes L, M e S têm entrega prioritária, qual das seguintes alternativas lista os clientes na ordem em que estes DEVEM receber suas compras?

- a) S, L, M, P, K e O
 b) L, K, O, P, S e M
 c) L, K, S, P, M e O
 d) M, L, S, P, K e O
 e) K, L, P, S, O e M

Q02: Se apenas um dos clientes tem entrega prioritária, e como resultado, O não pode receber suas compras antes que três outros clientes, qual dos seguintes clientes deve estar recebendo entrega prioritária?

- a) K b) M c) S d) P e) L

Maionese Um grande número de convidados ficou intoxicado após a festa de casamento de João e Joana. A suspeita logo recaiu sobre a salada de maionese servida na comemoração. Para tratar com mais eficácia os convidados afetados, amostras de maionese devem ser testadas para determinar a presença das toxinas R, S e T. Serão utilizados dois testes: X e Z.

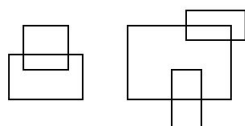
Sabe-se que:

- uma amostra mantém a cor adquirida em um teste a menos que outro teste altere a cor da amostra;
- o teste X faz a amostra adquirir cor verde se esta contém R ou S, ou ambas, e faz a amostra adquirir cor laranja se esta não contém nem R nem S;
- o teste Z faz a amostra adquirir cor roxa se esta contém T; se a amostra não contém T, permanece com a mesma cor que tinha antes do teste Z.

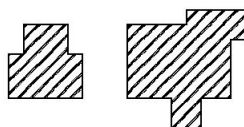
Q03: Uma amostra que contenha R e S mas não contenha T adquirirá que sequência de cores, a primeira após o teste X ser aplicado, e a segunda após o teste Z ser aplicado?

- a) Verde, verde b) Laranja, laranja c) Verde, roxo d) Laranja, amarelo
 e) Laranja, roxo

Mapas da prefeitura A Prefeitura de sua cidade iniciou um mapeamento das áreas urbanas do município. Para isso contratou uma empresa de aerofotogrametria, que tirou fotografias aéreas de uma grande porção da cidade. As fotografias são todas retangulares, mas foram tiradas de alturas distintas, de forma que cobrem áreas de tamanhos variados. Além disso, algumas áreas de algumas fotografias ficaram inutilizadas e foram cortadas, porque a visão do terreno foi prejudicada por nuvens; assim, apesar de todas as fotografias terem sido tiradas com a mesma máquina, nem todas têm as mesmas dimensões (mas todas são retangulares).



Conjunto de cinco fotografias



Área Total

Tarefa Dado um conjunto de fotografias do município, você deve calcular a área total coberta por todas as fotografias.

Formato dos Dados A primeira linha da entrada conterá um número N, que corresponde ao número de fotografias. Nas próximas N linhas serão descritas as fotografias, uma por linha. A descrição de uma fotografia será indicada por quatro números inteiros X1 Y1 X2 Y2, que serão as coordenadas do canto superior esquerdo (X1,Y1) e as do canto inferior direito (X2,Y2) da fotografia. Os lados das fotografias serão sempre paralelos aos eixos das coordenadas.

Atendimento ao Consumidor O Serviço de Atendimento ao Consumidor (SAC) de uma empresa emprega seis pessoas que respondem a mensagens de correio eletrônico: Gustavo, Horácio, Ivan, Joana, Kátia e Luiza. Cada mensagem de cliente recebida pelo SAC é classificada como vermelha ou azul. O seguinte procedimento é utilizado:

- Mensagens vermelhas são encaminhadas para tratamento inicialmente para Gustavo ou Horácio;
- Mensagens azuis são encaminhadas para tratamento inicialmente para qualquer um dos seguintes: Gustavo, Joana ou Ivan. Se uma mensagem menciona um problema que não pode ser resolvido pela pessoa à qual a mensagem foi inicialmente encaminhada para tratamento, ela deve ser redirecionada para outra pessoa, até que chegue a alguém que possa tratar do problema e responder a mensagem. Uma mensagem deve ser redirecionada da seguinte maneira:
 - de Gustavo para Ivan se a mensagem é vermelha, mas para Joana se a mensagem é azul;
 - de Horácio para Gustavo ou para Ivan;
 - de Ivan para Joana se a mensagem é vermelha, mas para Kátia se a mensagem é azul;
 - de Joana para Ivan ou para Kátia, não importando se a mensagem é vermelha ou azul;
 - de Kátia para Luiza, não importando se a mensagem é vermelha ou azul; Além disso, Luiza responde todas as mensagens que cheguem a ela.

Q01: Qualquer mensagem que chegou até Luiza deve ter passado por

- a) Gustavo b) Joana c) Ivan d) Kátia e) Horácio

Q02: Qual das seguintes situações poderia fazer com que a mensagem passasse por todos os membros do SAC?

a) Uma mensagem vermelha que é encaminhada inicialmente para Horácio
 b) Uma mensagem azul que é encaminhada inicialmente para Gustavo
 c) Uma mensagem azul que é encaminhada inicialmente para Ivan
 d) Uma mensagem vermelha que é encaminhada inicialmente para Gustavo
 e) Uma mensagem azul que é encaminhada inicialmente para Joana

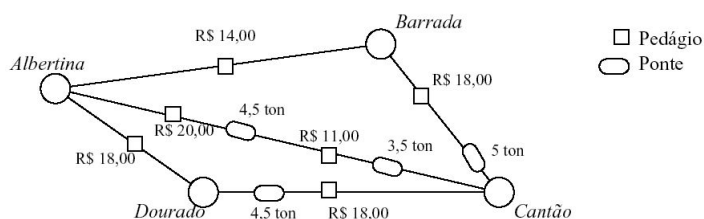
Clara e Luiz

Q03: Clara e Luiz estudam na mesma escola e cursaram as mesmas seis matérias durante o ano, mas apenas em uma das matérias "História do Brasil" tiraram a mesma nota. Em cada uma das matérias as notas variaram de 60 a 100.

Qual das seguintes afirmações permite que se deduza que a média das notas que Clara tirou nas outras cinco matérias foi maior que a média das notas de Luiz nessas mesmas cinco matérias?

- a) Clara teve notas maiores do que Luiz em três matérias.
 b) A menor nota de Luiz e a maior nota de Clara foram na mesma matéria.
 c) A maior nota de Luiz foi maior do que a maior nota de Clara.
 d) A menor nota de Clara foi HBr, mas a menor nota de Luiz foi Matem.
 e) A menor nota de Clara foi igual à maior nota de Luiz.

Economia no Transporte A empresa de transportes Super Barateiro Cargas quer economizar o pagamento de tarifas de pedágio, que representam uma parcela significativa do custo do transporte. No entanto, além do preço dos pedágios, os gerentes da SBC têm que levar em conta uma restrição adicional, que é a capacidade das pontes das estradas em suportar o peso dos caminhões. Por exemplo, para o caso da figura abaixo, para uma viagem entre as cidades de Albertina e Cantão, supondo que o peso total do caminhão com a carga é igual a 4,5 toneladas, o menor custo total de pedágio possível é igual a R\$ 32,00 (soma dos pedágios entre Albertina e Barrada e entre Barrada e Cantão). Note que a estrada que existe entre Albertina e Cantão, apesar de ter uma tarifa de pedágio atrativa, não comporta o peso do caminhão carregado.



Tarefa Dados um conjunto de cidades, os valores dos pedágios entre as cidades, as restrições de peso das pontes em cada estrada, e o peso total do

caminhão carregado, você deve determinar o menor valor de pedágio a ser pago entre duas cidades especificadas.

Formato dos Dados A primeira linha de dados conterá um número N, indicando o número de cidades e, em seguida, os nomes das N cidades (o nome de uma cidade não inclui espaços em branco). A segunda linha conterá um número M, indicando o número de estradas existentes entre as cidades. As próximas M linhas descrevem os valores de pedágios e restrições de peso para cada uma das estradas. Cada linha descreve uma estrada, e começa com o nome das duas cidades que essa estrada liga, seguido pelo número P de pedágios naquela estrada e P valores de pedágios, terminando com um número C de pontes na estrada e os C valores correspondentes aos pesos máximos suportados por cada ponte. A última linha conterá o peso do caminhão e o nome de duas cidades, indicando o peso do caminhão que se deseja transportar da primeira cidade para a segunda.

Exemplo de dados

```
4 Albertina Barrada Dourado Cantão
5
Albertina Cantão 2 20 11 2 4,5 3,5
Albertina Barrada 1 14 0
Albertina Dourado 1 18 0
Dourado Cantão 1 18 1 4,5
Barrada Cantão 1 18 1 5,0
4,5 Albertina Cantão
Resposta: R$ 32,00
```

Q04: Considere o conjunto de dados abaixo.

```
4 Lavras Ribeira Paulista Araras
5
Lavras Ribeira 1 18 1 3,5
Ribeira Araras 1 11 1 4,0
Araras Paulista 1 7 1 3,0
Paulista Lavras 1 10 1 2,5
Ribeira Paulista 1 19 1 3,0
3,0 Lavras Paulista
```

O menor custo total do pedágio será:

- a) R\$ 37,00 b) R\$ 19,00 c) R\$ 36,00 d) R\$ 17,00 e) R\$ 10,00

Q05: Considere o conjunto de dados abaixo:

```
5 Campos Vinhedo Valinhos Parati Palmas
10
Palmas Campos 2 25 30 1 4,0
Palmas Vinhedo 1 100 2 4,8 4,7
Palmas Valinhos 1 100 2 5,0 5,5
Palmas Parati 3 50 53 57 3 4,9 4,8 5,0
Campos Parati 2 15 18 2 5,0 4,7
Campos Valinhos 2 7 2 2 5,0 7,0
Campos Vinhedo 1 1 0
Vinhedo Parati 2 1 1 3 5,0 4,6 5,0
Vinhedo Valinhos 1 5 0
Valinhos Parati 2 33 33 1 4,7
4,7 Palmas Parati
```

O menor custo total do pedágio será:

- a) R\$ 139,00 b) R\$ 88,00 c) R\$ 160,00 d) R\$ 102,00 e) R\$ 134,00



Responda a opção de cada questão

1	2	3	4	5

1

Projeto de Software O engenheiro-chefe de uma empresa de desenvolvimento de software está atribuindo funcionários a cada um dos três novos projetos, identificados como Projeto 1, Projeto 2 e Projeto 3. Exatamente dois dos funcionários seguintes devem ser atribuídos a cada um dos projetos: Alice, Bia, Clara, Denise, Eduardo e Felipe. Cada funcionário deve trabalhar em um único projeto. Alice, Bia e Clara são engenheiras sênior; os demais são engenheiros junior. Alice, Denise e Eduardo falam inglês os demais não falam inglês.

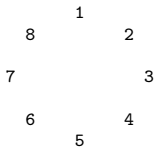
O engenheiro-chefe deve atribuir os funcionários aos projetos de acordo com os seguintes requisitos:
* pelo menos uma engenheira sênior deve trabalhar em cada projeto;
* pelo menos um funcionário que fale inglês deve trabalhar em cada projeto;
* Denise somente pode trabalhar no Projeto 1 ou no Projeto 2.

Q01: Se Eduardo trabalhar no Projeto 2, qual das seguintes afirmações é necessariamente verdadeira?

- a) Felipe trabalhará no projeto 3
- b) Bia trabalhará no Projeto 1
- c) Clara trabalhará no Projeto 2
- d) Felipe trabalhará no Projeto 1
- e) Clara trabalhará no Projeto 3

Q02: O engenheiro-chefe NAO PODE fazer as seguintes atribuições:
a) Felipe trabalhar ni projeto 1 e denise trabalhar no projeto 2
b) Alice trabalhar no Projeto 1 e Bia trabalhar no Projeto 2
c) Denise trabalhar no Projeto 1 e Eduardo trabalhar no Projeto 3
d) Eduardo trabalhar no Projeto 1 e Clara trabalhar no Projeto 3
e) Alice trabalhar no Projeto 2 e Clara trabalhar no Projeto 3

Florista Uma florista está arranjando oito flores (A, B, C, F, G, J, K e L) em oito vasos colocados em formato de círculo, como mostrado abaixo:



Sabe-se o seguinte sobre o arranjo de flores:
* A, B e C são lírios; F e G são margaridas; J, K e L são rosas;
* apenas uma flor deve ser colocada em cada vaso;
* lírios devem colocados em vasos vizinhos;
* rosas devem colocadas em vasos vizinhos;
* nenhum lírio pode ser colocado em vaso vizinho a um vaso de rosa;
* a flor F deve ser colocada no vaso 5;
* a flor A deve ser colocada no vaso 2;
* se F for colocada em vaso vizinho a J, então o outro vaso vizinho de F não pode conter C.

Q03: Qual das seguintes afirmações NÃO é verdadeira?

- a) o vaso 4 contém um lírio
- b) o vaso 1 contém uma rosa
- c) o vaso 3 conté um lírio
- d) o vaso 8 contém uma rosa
- e) o vaso 7 contém uma rosa

Q04: Se K for colocada no vaso 8 e C no vaso 4, qual dos seguintes pares de flores NÃO podem estar em vasos vizinhos?

- a) G e K b) F e L c) J e L d) B e F e) B e C

Pane Seca A estrada entre Abrigo e Feliciano é a única ligação terrestre entre essas duas cidades. Ela atravessa todo o extenso cerrado de Mato Baixo, sem passar por nenhuma outra cidade. Assim, os motoristas têm que se preocupar em escolher bem em quais dos postos de gasolina irão colocar combustível, sob pena de ficar parado no meio do caminho por "pane seca"(falta de combustível).

Há um posto de combustível em Abrigo, um em Feliciano, e outros quatro postos ao longo da estrada entre as duas cidades. O combustível nos postos de Abrigo e Feliciano é mais barato do que nos outros quatro postos ao longo da estrada, de forma que os motoristas preferem encher o tanque no posto da cidade antes de uma viagem entre as duas cidades. O preço do combustível é igual para os quatro postos ao longo da estrada.

Os seis postos são identificados pelas letras A (posto em Abrigo), B, C, D, E e F (posto em Feliciano). A distância entre o posto de Abrigo e o posto de Feliciano é de 430 quilômetros. Considerando como ponto inicial o posto da cidade de Abrigo, os postos estão localizados nos quilômetros 0 (posto de Abrigo), 33, 108, 171, 332 e 430 (posto de Feliciano).

Alan quer viajar entre as duas cidades com sua motocicleta, que tem um tanque com capacidade de 8 litros, e necessita de sua ajuda para planejar a viagem.

Q05: Considere que a motocicleta de Alan percorre 24 quilômetros por litro de combustível. Se Alan vai viajar de Abrigo a Feliciano, e já encheu o tanque de sua motocicleta no posto de Abrigo, qual das seguintes alternativas é verdadeira?
a) Alan certamente vai ter uma pane seca durante a viagem
b) Alan deve abastecer em ao menos três outros postos
c) Alan deve abastecer obrigatoriamente no posto C
d) Alan deve abastecer em todos os outros postos
e) Alan deve abastecer obrigatoriamente no posto D

Linguagem de Programação Atualmente existe um grande número de diferentes linguagens de programação: Java, Pascal, C, e muitas outras. Essas linguagens são utilizadas para programar computadores e têm regras muito estritas. Em uma determinada linguagem de programação, um comando é definido como uma sequência de cinco palavras, de tal forma que

- cada palavra deve conter pelo menos cinco letras, sendo que não mais do que duas podem ser vogais (a, e, i, o, u);
- uma palavra não pode iniciar com as letras c, q ou z;
- considere a sequência de cinco letras formadas pela primeira letra de cada uma das cinco palavras; essas letras devem ser consecutivas na ordem alfabética (abcdefghijklmnopqrstuvwxyz).

Q06: Cada uma das seguintes palavras pode ser a última palavra em um comando EXCETO

- a) norte b) super c) vidro d) ladrar e) morte

Q07: A última palavra de um comando NAO PODE iniciar com a letra

- a) v b) t c) n d) j e) o

Semana de Provas O diretor de uma escola está planejando a utilização do laboratório de computação para a semana de provas práticas. As provas ocorrerão entre os dias 1 e 5 (inclusive) do ultimo mês de aulas, com duas provas agendadas para cada dia. Das dez provas que serão agendadas, quatro são do prof. Rui, duas da profa. Sônia, duas do prof. Téo, um do prof. Udo e uma da profa. Vera. As provas devem ser agendadas obedecendo às seguintes restrições:

* duas provas de um mesmo professor ou professora não podem ser agendadas para o mesmo dia, exceto para o dia 3, quando serão agendadas duas provas do prof. Rui;
* a prova da profa. Vera será agendada para o dia 5;
* nenhuma prova da profa. Sônia pode ser agendada para o mesmo dia de uma prova do prof. Rui.

Q08: Se as provas do prof. Rui forem agendadas para os dias 2, 3 e 4, qual das seguintes afirmações é necessariamente verdadeira?

- a) A prova do prof. Udo será dia 1
- b) Uma prova da profa. Sônia será dia 2
- c) Uma prova do prof. Téo será dia 4
- d) Uma prova da profa. Sônia será dia 5
- e) Uma prova do prof. Téo será dia 2

Jogo de Futebol Um grupo de amigos conseguiu ingressos para assistir ao jogo da seleção brasileira que vai acontecer na cidade onde moram. O grupo é formado por três mulheres (Renata, Sílvia e Tais), dois homens (Ulisses e Vinicius) e quatro crianças (Waldo, Xuxa, Yara e Zinho). O grupo tem um total de nove ingressos, mas os assentos são em três seções diferentes do estáadio; os três assentos em cada uma das seções são contíguos. Os amigos devem ser divididos em sub-grupos de três de acordo com as seguintes restrições:

* não pode haver adultos do mesmo sexo no mesmo sub-grupo;
* Waldo não pode estar no mesmo sub-grupo que Renata;
* Xuxa deve estar no mesmo sub-grupo de Sílvia, ou de Ulisses, ou de ambos.

Q09: Se Renata e Ulisses são dois dos três integrantes de um primeiro sub-grupo, quais amigos podem estar no segundo e terceiro sub-grupo, respectivamente:

- a) 2=Sílvia, Tais Waldo e 3=Vinicius, Yara, Zinho
- b) 2=Tais, Vinicius, Waldo e 3=Sílvia, Yara, Zinho
- c) 2=Sílvia, Xuxa, Yara e 3=Tais, Waldo, Zinho
- d) 2=Waldo, Xuxa, Yara e 3=Sílvia, Vinicius, Zinho
- e) 2=Sílvia, Waldo, Zinho e 3=Tais, Vinicius, Xuxa

Q10: Qualquer dos seguintes pares de pessoas pode estar no sub-grupo de Xuxa, EXCETO:

- a) Sílvia e Ulisses
- b) Sílvia e Waldo
- c) Sílvia e Tais
- d) Tais e Ulisses
- e) Renata e Ulisses

 Responda a opção de cada questão

1	2	3	4	5	6	7	8	9	10

Excursão

A escola está preparando uma excursão, e os pais também estão convidados. Para levar pais e alunos foram contratados sete ônibus, identificados como A, B, C, D, E, F e G. Para permitir uma maior escolha das famílias, os sete ônibus deverão partir cada um em horário diferente, seguindo as seguintes restrições:

- os ônibus devem partir sábado, nos seguintes horários: 6:00, 7:00, 8:00, 9:00, 10:00, 11:00 e 12:00;
- o ônibus A deve partir às 6:00;
- o ônibus E deve partir após o ônibus C e também após o ônibus B;
- os nibus D, F e G devem partir em horários consecutivos, nessa ordem.

- Q01: Se os ônibus C e D forem escalados para partir às 8:00 e 9:00, respectivamente, o ônibus B deve ser escalado para partir às
- a) 7:00 b) 11:00 c) 6:00 d) 10:00 e) 12:00
- Q02: Se o ônibus F for escalado para partir as 11:00, o ônibus E deve ser escalado para partir às
- a) 8:00 b) 7:00 c) 11:00 d) 10:00 e) 9:00

Banda de Rock

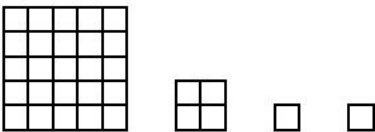
Uma famosa banda de rock esté procurando um novo guitarrista. Hé seis candidatos (R, S, T, U, V e W), e o novo membro da banda será escolhido através de uma série de audições, que acontecerão durante um período de duas semanas. Exatamente um candidato se apresentará a cada dia, de segunda-feira a sábado (não haverá audições aos domingos). Cada candidato fará exatamente uma apresentação na primeira semana e uma apresentação na segunda semana. As seguintes restrições existem para determinar a ordem das apresentações dos candidatos:

- Em ambas as semanas a apresentação de R ocorre antes da apresentação de S.
- A apresentação de S na primeira semana é no mesmo dia da semana que a sua apresentação na segunda semana.
- Em ambas as semanas a apresentação de U não pode ser no dia imediatamente anterior nem no dia imediatamente posterior ao dia da apresentação de W.
- T e V devem apresentar-se em dias imediatamente consecutivos, em qualquer ordem, nas duas semanas.
- As apresentações de W são na quinta-feira da primeira semana e na terça-feira da segunda semana.

- Q03: Se uma das apresentações de T é em uma segunda-feira, qual das seguintes alternativas lista dias em que U poderia apresentar-se, respectivamente, na primeira e na segunda semanas?
- a) Sábado, quinta-feira
b) Segunda-feira, sábado
c) Sábado, sábado
d) Terça-feira, sexta-feira
e) Terça-feira, quinta-feira
- Q04: Se ambas as apresentações de U são tão cedo quanto possível na semana, qual das seguintes é a lista completa dos dias da semana em que T poderia apresentar-se?
- a) Quarta-feira, quinta-feira, sexta-feira, sábado
b) Terça-feira, quarta-feira, sexta-feira, sábado
c) Segunda-feira, terça-feira
d) Sexta-feira, sábado
e) Segunda-feira, terça-feira, quarta-feira

Azulejos

São dados N azulejos de dimensões 10cm × 10cm. Com eles, você deve montar um conjunto de quadrados (com espessura de um azulejo) de modo a utilizar TODOS os azulejos dados. Inicialmente você deve montar o maior quadrado possível com os azulejos dados; então, com os azulejos que sobram, você deve montar o maior quadrado possível, e assim sucessivamente. Por exemplo, se forem dados 31 azulejos, o conjunto montado terá quatro quadrados, conforme ilustra a figura abaixo



Conjunto com quatro quadrados, montado a partir de 31 azulejos

- Q05: Qual o número de quadrados do conjunto montado se forem dados 75 azulejos?
- a) 6 b) 3 c) 4 d) 5 e) 2

Transporte Escolar

Quatro meninos – Aldo, Beto, Carlos, Dida, e Edu – e cinco meninas – Fátima, Guta, Helena, Julia e Kátia j vão para a escola em três vans distintas.

- Aldo e Fátima sempre viajam juntos.
- Guta e Helena sempre viajam juntas.
- Julia e Kátia nunca viajam juntas.
- Dida sempre viaja na van que carrega menos crianças.
- Em qualquer van, o número de meninos não pode ser maior do que o número de meninas.
- O número máximo de crianças em qualquer van é quatro.

- Q06: A van em que Dida viaja pode carregar no máximo quantas crianças?
- a) 1 b) 5 c) 4 d) 3 e) 2
- Q07: Se Beto viaja na mesma van que Fátima, então Guta deve necessariamente viajar com qual das seguintes crianças?
- a) Dida b) Fátima c) Julia d) Kátia e) Carlos

Responda:

1	2	3	4	5	6	7

Pantanal

Uma fazenda no pantanal, de terras totalmente planas, tem quatro armazéns – A, B, C e D – muito distantes entre si, que são ligados por quatro trilhas retas – Q, R, S e T. As trilhas têm todas o mesmo comprimento e conectam os armazéns da seguinte forma:

- * Q conecta apenas A e C.
- * R conecta apenas B e C.
- * S conecta apenas A e B.
- * T conecta apenas B e D.

Q01. Se, tomando atalhos fora das trilhas, o fazendeiro pudesse ir de C a D por um caminho mais curto do que a distância mínima entre C e D utilizando apenas trilhas, 5 afirmações são feitas:

1. A menor distância de A a D utilizando apenas trilhas é menor do que a menor distância de C a D utilizando apenas trilhas.
2. A sequência de trilhas de menor comprimento entre A e D é a menor distância entre A e D.
3. A rota composta por R e T não forma uma linha reta.
4. A rota composta por S e T não forma uma linha reta.
5. As trilhas R e T encontram-se em um ângulo reto (90°).

Assinale a alternativa correta

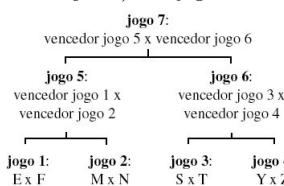
- a) A afirmação 5 é necessariamente verdadeira
- b) A afirmação 1 é necessariamente verdadeira
- c) A afirmação 4 é necessariamente verdadeira
- d) A afirmação 3 é necessariamente verdadeira
- e) A afirmação 2 é necessariamente verdadeira

Q02. Qual das seguintes alternativas é a ordem em que o fazendeiro, partindo de A, usando apenas as trilhas, e nunca usando uma trilha mais de uma vez, visita todos os outros armazéns?

- a) C, B, D b) D, C, B c) B, C, D d) D, B, C e) C, D, B

Torneio de Tênis

Um torneio de tênis está sendo organizado no clube. O torneio terá sete jogos, numerados de 1 a 7. Oito jogadores vão competir: E, F, M, N, S, T, Y e Z. Em cada jogo dois jogadores enfrentam-se, e um deles sai vencedor (não há empates). Os jogos serão da seguinte forma:



O vencedor do jogo 7 é o campeão do torneio. As seguintes restrições devem ser obedecidas:

- * Não pode haver um jogo de E contra N.
- * Se M vencer o jogo 2, então ele deve enfrentar F no jogo 5 e S deve vencer o jogo 3.
- * O campeão não pode ser N, S ou T.
- * Se Z vencer o jogo 6, então M deve vencer o jogo 2.
- * Se T e Y enfrentarem-se, T deve vencer o jogo.

Q03: Se N vence o jogo 5, qual dos seguintes é um jogador que vence pelo menos dois jogos?

- a) F b) Y c) Z d) T e) S

Q04: Qual dos seguintes poderia ser um par de jogadores que enfrentam-se no jogo 7?

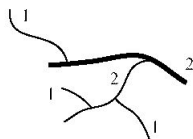
- a) N, S b) N, Z c) M, Y d) M, T e) E, S

Hidrologia

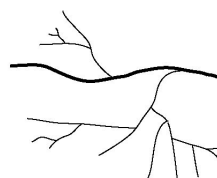
O índice de complexidade de um rio é calculado da seguinte maneira:

- se o rio não tem afluentes, o seu índice de complexidade é 1;
- se o rio tem afluentes de índices diferentes, então o seu índice de complexidade é igual ao do afluente de maior índice;
- se todos os afluentes do rio têm o mesmo índice H de complexidade (incluindo o caso quando há apenas um afluente), o seu índice será H + 1.

Por exemplo, o rio principal da figura abaixo (representado por uma linha mais grossa) tem índice de complexidade 2.



Q05: Qual o índice de complexidade do rio principal da figura abaixo?



- a) 3 b) 7 c) 5 d) 6 e) 4

Tornado

Será que este clima maluco é o resultado da contínua interferência da humanidade no meio ambiente? Ou é simplesmente um ciclo normal, que acontece desde as eras remotas? Ninguém parece saber por certo, mas o fato é que fenômenos naturais como tornados e enchentes têm atingido nosso país com mais força e frequência do que em décadas passadas. Um tornado acabou de atingir a Fazenda Boa Vista, que produz leite e carne, e causou grande destruição. O telhado da sede foi afetado, diversas árvores arrancadas, o trator foi tombado pelo vento... Mas a pior coisa que aconteceu foi que o tornado destruiu diversas seções da cerca que circundava a propriedade. A cerca era muito bem construída, com postes de concreto a cada dois metros, e arame farpado cercando todo o perímetro da fazenda (o perímetro, em metros, é um número par, de forma que a cerca era perfeitamente regular).



- poste de concreto
- poste quebrado ou faltando

Agora diversos postes estão quebrados ou simplesmente sumiram, levados pela força do vento, deixando vãos abertos na cerca. Para evitar que o gado saia da propriedade, a cerca deve ser restaurada o mais breve possível. No entanto, a reconstrução nos mesmos padrões anteriores vai levar muito tempo; como solução provisória, os donos da fazenda decidiram fechar os vãos com uma cerca temporária, feita com postes de madeira. Os postes de madeira serão colocados exatamente nos mesmos lugares onde postes quebrados ou faltantes estavam. Além disso, para permitir uma reconstrução mais rápida e menos custosa, os donos decidiram utilizar menos postes do que a cerca original: um poste de madeira será utilizado para repor um poste de concreto quebrado ou faltante apenas se o comprimento do arame farpado necessário para fechar o vão até o próximo poste (de madeira ou concreto) exceder quatro metros.

Tarefa: Dada a descrição de quais postes originais estão quebrados ou faltando, você deve determinar o menor número de postes de madeira serão necessários para reconstruir a cerca de acordo com a decisão dos donos da fazenda.

Formato dos Testes: Cada caso de teste considera um mapa de danos, que consiste de uma sequência de valores 0 e 1, representando o estado de cada poste de concreto após o tornado. O valor 0 indica que o poste respectivo está quebrado ou faltando, e o valor 1 indica que o poste está em bom estado. Note que o último valor da sequência indica um poste que está contíguo ao poste representado pelo primeiro valor.

Alguns exemplos: o caso 100101 tem como resultado 1. O caso 00000 dá 3 e o caso 10111 dá resultado 0.

Q06: Se o mapa de danos for 100101001000 o menor número de postes de madeira a serem utilizados é

- a) 4 b) 2 c) 0 d) 1 e) 3



Resposta:

1	2	3	4	5	6

Palestras

O professor de Química está organizando um ciclo de palestras no próximo sábado. São oito palestras ao todo, apresentadas por oito empresas diferentes: F, L, M, O, R, T, X e Z. As palestras serão organizadas em dois grupos de quatro palestras. As palestras de cada grupo serão apresentadas em sequência, uma de cada vez; um grupo fará as apresentações no Auditório 1 e o outro grupo no auditório 2. Todas as palestras terão a mesma duração, e enquanto uma palestra estiver sendo apresentado no Auditório 1, uma outra estará sendo apresentada no Auditório 2, com mesmos horários de início e fim. A programação deve ainda obedecer as seguintes condições:

- A palestra F deve ocorrer no mesmo horário que a palestra M.
- A palestra L deve ocorrer no mesmo horário que a palestra O.
- A palestra R deve ocorrer no mesmo auditório que a palestra F.
- A palestra T deve ocorrer no mesmo auditório que a palestra O.
- A palestra X deve ser a segunda a ser apresentada no Auditório 2.

Q01: Se a palestra F for apresentada entre a palestra X e a palestra R no Auditório 2, qual das seguintes é necessariamente a primeira palestra no Auditório 1?

- a) M b) T c) O d) Z e) L

Q02: Se a palestra T for apresentada no Auditório 1, qual das seguintes palestras será necessariamente apresentada no Auditório 2?

- a) M b) L c) F d) R e) Z

Analistas de Sistemas

Uma empresa está instalada no primeiro, segundo, terceiro e quarto andares de um prédio de escritórios. Três analistas de sistemas, Alice, Bia e Clara, com cargos de analista sênior e analista junior, trabalham na empresa, e devem ser alocadas nos andares de acordo com as seguintes restrições:

- nem todos os analistas têm o mesmo cargo;
- cada andar da empresa pode ter apenas uma analista alocada;
- Alice é alocada no quarto andar;
- Bia é analista senior;
- Bia é alocada em algum andar acima do andar de Clara;
- Se há analista alocada no terceiro andar, então tal analista tem o mesmo cargo que a analista do quarto andar.

Q03: Qual das seguintes afirmações é sempre verdadeira?

- a) Alice e Bia têm o mesmo cargo.
b) Alice e Clara têm o mesmo cargo.
c) Bia e Clara têm o mesmo cargo.
d) Ou Alice ou Clara ou ambas são analistas sêniores.
e) Ou Alice ou Clara ou ambas são analistas juniores.

Q04: Qual das seguintes afirmações é necessariamente falsa?

- a) Bia é alocada no segundo andar.
b) Clara é alocada no primeiro andar.
c) Clara é alocada no segundo andar.
d) Bia é alocada no terceiro andar.
e) Clara é alocada no terceiro andar.

Uma Casa Muito Suspeita

A polícia desconfia que uma casa freqüentada por quatro pessoas, identificadas como Juca, Kiko, Lia e Mel, é utilizada para clonagem de telefones celulares e cartões de crédito. Montou-se então uma operação de vigilância, que obteve as seguintes informações:

- a cada dia, os suspeitos entram e posteriormente saem da casa exatamente uma vez;
- um suspeito nunca entra ou sai da casa com outro suspeito;
- nenhum dos suspeitos sai da casa na mesma posição (primeira, segunda, terceira e quarta) em que entrou na casa;
- tanto entrando como saindo do prédio, Kiko sempre precede Lia.

Q05: Qual das seguintes opções abaixo contém uma ordem de saída da casa que poderia ser verdadeira em um dia em que os suspeitos entram na casa na ordem Mel, Juca, Kiko, Lia?

- a) Lia, Mel, Juca, Kiko
b) Juca, Kiko, Mel, Lia
c) Juca, Kiko, Lia, Mel
d) Kiko, Juca, Lia, Mel
e) Mel, Kiko, Lia, Juca

Representação Pós-Fixa

Desde pequenos aprendemos a escrever expressões aritméticas em que as operações com dois argumentos que são escritas na seguinte ordem: o primeiro argumento, o símbolo de operação, o segundo argumento. Esta maneira de escrever expressões é denominada notação infixa. Um dos problemas que ela apresenta é a necessidade de regras de prioridade e de parênteses para indicar exatamente as operações. Por exemplo, as expressões infixas:

$5 + 7 * 3$ e $(5 + 7) * 3$

representam valores distintos (26 e 36). A fim de evitar o uso de parênteses, existe uma outra notação, denominada pós-fixa, na qual a ordem é o primeiro argumento, o segundo argumento, o símbolo de operação. Por exemplo, as expressões acima seriam escritas nesta notação como:

$5\ 7\ 3\ *\ +$ e $5\ 7\ +\ 3\ *$

Note que esta notação, além de não usar parênteses, não precisa também de regras de prioridade.

Q06: Qual a tradução correta para a forma pós-fixa da expressão infixa abaixo?

- (a * b) + (c * d)
a) a b * c d * +
b) a b * c d + *
c) a b c d * * +
d) a b c d + * *
e) a b * + c d *

🗒️ Responda:

1	2	3	4	5	6

Mapas

Um desenhista est  produzindo dois mapas, um para mostrar as linhas do metro e outro para mostrar as rotas de  nibus da cidade. H  tr s linhas de metro e quatro de  nibus, e cada linha ou rota deve ser representada nos mapas por uma cor usada para representar apenas essa linha ou rota. As cores dispon veis para o desenhista s o Azul, Verde, Laranja, Roxo, Preto, Marrom e Cinza. Qualquer atribui  o de cores a linhas e rotas   aceit vel desde que obede a as seguintes condi   es:

- Azul n o pode ser usado no mesmo mapa que Roxo.
- Laranja n o pode ser usado no mesmo mapa que Preto, nem no mesmo mapa que Cinza.

Q01: Se Cinza e Roxo s o usados no mapa do metro, a terceira cor a ser usada nesse mapa   necessariamente

- a) Verde b) Preto c) Laranja d) Azul e) Marrom

Q02: Se Azul   usado no mapa do metro, qual das seguintes alternativas   necessariamente verdadeira?

- a) Laranja   usado no mapa do metro.
b) Cinza   usado no mapa do metro.
c) Preto   usado no mapa de  nibus
d) Verde   usado no mapa de  nibus
e) Roxo   usado no mapa de  nibus

MP3 na Cabe a

Clara est  carregando m sicas em seu prendedor de cabelo, que funciona tamb m como um tocador MP3. O tocador   capaz de armazenar exatamente tr s m sicas longas – entre C, D e F – ou exatamente seis m sicas curtas – entre H, J, K, L, P e Q. Cada m sica curta ocupa exatamente a metade da mem ria de uma m sica longa. Clara usa toda a mem ria dispon vel. As seguintes restri   es devem ser obedecidas:

- Se J   carregada, ent o L   carregada.
- Se C   carregada, ent o K n o   carregada.
- Se L   carregada, ent o C   carregada.
- Se ou P ou D s o carregadas, ent o ambas s o carregadas.
- Se C   carregada, ent o Q   carregada.
- Se F   carregada, ent o D n o   carregada.

Q03: Qual dos seguintes   um par de m sicas que N O pode ser carregado junto?

- a) Q, F b) F, L c) L, P d) H, F e) C, D

Q04: Se H n o   carregada, qual dos seguintes pares de m sicas poderia ser carregado?

- a) F, P b) D, L c) K, P d) J, Q e) C, P

Q05: Se L   carregada, qual das seguintes m sicas n o pode ser carregada?

- a) F b) P c) H d) J e) C

Horta da Maria

Maria tem um grande quintal e resolveu fazer uma horta, que ter  tr s ou quatro canteiros. Em cada canteiro, apenas um tipo de hortali a ser  plantada. Exatamente os mesmos canteiros que forem utilizados no primeiro ano ser o tamb m utilizados no segundo ano, mas em nenhum canteiro a mesma hortali a ser  plantada por dois anos consecutivos. Para cada canteiro, Maria vai escolher entre cinco poss veis hortali as, sendo duas verduras (alface e repolho) e tr s legumes (beterraba, cenoura e nabo). O plantio deve ser feito de acordo com as seguintes condi   es:

- em qualquer ano, pelo menos um dos canteiros deve ter um legume (beterraba, cenoura e nabo);
- no ano seguinte ao que nabo   plantado em um canteiro, uma verdura (alface ou repolho) deve ser plantada nesse canteiro.

Q06. Se Maria plantar quatro canteiros, o numero m ximo de canteiros que podem ser plantados com legumes por dois anos  

- a) 1 b) 0 c) 2 d) 3 e) 4

Sandwiches do Joao

Jo o chamou dois amigos para o lanche e est  fazendo tr s sandu ches A, B e C. Cada sandu che   composto de duas camadas: a de cima e a de baixo, obedecendo as seguintes caracter sticas:

- uma vez feito, o sandu che n o pode ser virado (ou seja, a camada de cima n o se torna a camada de baixo);
- cada camada   feita com exatamente um dos seguintes ingredientes: queijo, presunto, tomate ou ovo;
- nos tr s sandu ches, as camadas de baixo s o diferentes entre si;
- nos tr s sandu ches, as camadas de cima s o diferentes entre si;
- em cada sandu che, a camada de cima   diferente da camada de baixo;
- exatamente uma camada de cima   de tomate;
- nenhuma das camadas de baixo   de presunto;
- no sandu che C, uma das camadas   de queijo;
- a camada de cima do sandu che B   de presunto.

Q07: Dadas 5 afirma  es:

1. Tanto a camada de cima do sandu che A como a camada de baixo do sandu che B s o de queijo.
2. Tanto a camada de cima do sandu che A como a camada de baixo do sandu che B s o de ovo.
3. Tanto a camada de cima do sandu che A como a camada de baixo do sandu che C s o de ovo.
4. A camada de cima do sandu che A   de queijo e a camada de baixo do sandu che C   de tomate.
5. A camada de cima do sandu che C   de ovo e a camada de baixo do sandu che A   de tomate.

Assinale a alternativa correta:

- a) a afirma  o 4 poderia ser verdadeira
b) a afirma  o 2 poderia ser verdadeira
c) a afirma  o 5 poderia ser verdadeira
d) a afirma  o 3 poderia ser verdadeira
e) a afirma  o 1 poderia ser verdadeira



Responda:

1	2	3	4	5	6	7

Logaritmos

Números

Definem-se, e trabalha-se com os seguintes conjuntos de números:

Naturais $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.

Inteiros $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$.

Racionais $\mathbb{Q} = \{\text{Conjunto das frações } a/b, \text{ onde } a \in \mathbb{Z} \text{ e } b \in \mathbb{Z}^*\}$.

Reais $\mathbb{R} = \{\text{Conjunto dos racionais mais as decimais não exatas e não periódicas, chamados números irracionais}\}$.

Complexos $\mathbb{C} = \{\text{Números na forma } a + b.i, \text{ onde } a \text{ e } b \in \mathbb{R} \text{ e } i \text{ é } \sqrt{-1}\}$.

Para todos os conjuntos $\mathbb{X}$ acima, valem os seguintes sub-conjuntos: $\mathbb{X}^*$ é o conjunto $\mathbb{X}$ sem o zero. $\mathbb{X}_+$ é o conjunto dos positivos de $\mathbb{X}$ e $\mathbb{X}_-$ é o conjunto dos negativos de $\mathbb{X}$.
É óbvio que $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$.

Potência

Seja a um real e n um natural. A potência de a ao expoente n , denotada a^n , é o número real:

- $a^0 = 1$
- $a^n = a^{n-1} \times a, \forall n, n \geq 1$

Uma definição que permite estender o conjunto $\mathbb{N}$ dos expoentes em direção ao $\mathbb{Z}$ é:

$$a^{-n} = \frac{1}{a^n}$$

Generalizando, podem-se escrever as seguintes propriedades:

- $a^m \times a^n = a^{m+n}$; 2. $\frac{a^m}{a^n} = a^{m-n}, a^n \neq 0$; 3. $(a.b)^n = a^n.b^n$; 4. $(a/b)^n = a^n/b^n, b^n \neq 0$; 5. $(a^m)^n = a^{m.n}$.

Exemplos

$$\begin{array}{ll} 3^0 = 1 & (-2)^0 = 1 \\ 5^1 = 5 & 3^2 = 3.3 = 9 \\ (1/7)^1 = 1/7 & (-2)^3 = -2. -2. -2 = -8 \\ (-2)^2 = -2. -2 = 4 & 2^2 = 2.2 = 4 \\ (2/3)^4 = 16/81 & 0^3 = 0.0.0 = 0 \\ 0^0 = 1 & 0^1 = 0 \end{array}$$

Para você fazer

Calcule os 3 exercícios abaixo e responda no local correto:

$$\begin{array}{ll} \text{a. } x = (-5)^0 & \text{b. } (1/5)^x = 125 \\ \text{c. } x = \frac{2^{10}-3^6}{2^5+3^3} & \end{array}$$

Raiz

A raiz n -ésima r de a , denotada $r = \sqrt[n]{a}$ é o número r que elevado a n gera a . Em outras palavras, tem-se:

$$\sqrt[n]{a} = r \Leftrightarrow r^n = a$$

Originalmente, a raiz aritmética é definida apenas para $a \in \mathbb{R}$ e $a \geq 0$, além de $n \in \mathbb{N}$, mas com as convenientes generalizações, valem aqui as seguintes propriedades: Se $a \in \mathbb{R}_+, b \in \mathbb{R}_+, m \in \mathbb{Z}, n \in \mathbb{N}^*$ e $p \in \mathbb{N}^*$, temos: 1. $\sqrt[n]{a^m} = \sqrt[n]{a^m} = \sqrt[n]{a^m}$; 2. $\sqrt[n]{a.b} = \sqrt[n]{a}.\sqrt[n]{b}$; 3. $\sqrt[n]{a/b} = \frac{\sqrt[n]{a}}{\sqrt[n]{b}}, (b \neq 0)$; 4. $(\sqrt[n]{a})^m = \sqrt[n]{a^m}$; 5. $\sqrt[p]{\sqrt[n]{a}} = \sqrt[p.n]{a}$;

Exemplos

$$\begin{array}{ll} 2.\sqrt[3]{3} = \sqrt[3]{3.2^3} = \sqrt[3]{24} & -5.\sqrt{2} = -\sqrt{2.5^2} = -\sqrt{50} \\ -2.\sqrt[4]{2} = -\sqrt[4]{2.2^4} = -\sqrt[4]{32} & \sqrt[3]{64} = \sqrt[3]{2^6} = 2^2 = 4 \end{array}$$

Para você fazer

Calcule os 3 exercícios abaixo e responda abaixo d. $x = 256^{\frac{5}{4}}$

$$\begin{array}{l} \text{e. } x = 1024^{\frac{1}{10}} \\ \text{f. } x = (16^{\frac{5}{4}})^{\frac{2}{5}} \end{array}$$

Logaritmos

O logaritmo n de a na base r , denotado por $n = \log_r a$ é o expoente ao qual se deve elevar a base r para obter a . Em outras palavras, tem-se:

$$\log_r a = n \Leftrightarrow r^n = a$$

Note a similaridade da expressão do logaritmo com a da raiz, o que nos permite montar a seguinte equivalência:

$$\sqrt[n]{a} = r \Leftrightarrow r^n = a \Leftrightarrow \log_r a = n$$

O conceito de logaritmo foi muito importante na história da matemática desde a sua proposição original por Napier em 1614, pois ele simplificava muito os cálculos ao permitir converter potências em multiplicações e estas em somas¹. Na informática, (no tópico complexidade computacional) os logaritmos são fundamentais porque descrevem o comportamento de uma classe importante de algoritmos. Em particular, são muito usados os logaritmos de base binária. Relembrando as potências de dois:

n	1	2	4	8	16	32	64	128	256	2^n
$\log_2 n$	0	1	2	3	4	5	6	7	8	n
n	512	1024	2048	4096	8192	16384	32768	65536	131072	2^n
$\log_2 n$	9	10	11	12	13	14	15	16	17	n

Exemplos

- Suponha que eu contrate José para carregar caixas, oferecendo-lhe n reais pelo transporte de n caixas. Também contrato João, oferecendo-lhe $\log_2 m$ dólares pelo transporte de m caixas. Quem será mais bem remunerado? Em que condições?
- Quanto é $\log_2 \frac{1}{8}$? 3. Quanto é $\log_{0,25} 32$?
Mais exemplos
 $\log_2 1024 = 10$ $\log_4 16 = 2$ $\log_2 10 = 3,32$

Para você fazer

Calcule os 3 exercícios abaixo:

$$\begin{array}{l} \text{g. } x = \log_{\frac{1}{2}} 8 \\ \text{h. } x = 3^{\log_3 2} \\ \text{i. } x = 4^{\log_2 3} \end{array}$$

De novo, calcule os 3 exercícios abaixo:

$$\begin{array}{l} \text{j. } x = 2^{1+\log_2 5} \\ \text{k. } \log_4 (3x+2) = \log_4 (2x+5) \\ \text{l. } (\log_{10} (2x-5)) = 0 \end{array}$$

Respostas

a	b	c	d	e	f	g	h	i	j	k	l

2



- 1 - /

¹a regra básica aqui é a seguinte: para obter o produto $m.n$, obtem-se o $\log m$ e o $\log n$, soma-se-os e obtem-se o antilogaritmo da soma.

²Quando ao lado do exercício aparecerem as siglas SRP ou SRI, o significado delas é

SRP = somente o resultado positivo

SRI = somente o resultado inteiro

Se estas letras não aparecerem, desconsidere esta nota

Complexidade, recursão, relação de recorrência

Uma parte pequena de algoritmos não contempla a repetição de instruções. O algoritmo começa no primeiro comando, vai descendo (por ação da gravidade, como se costuma dizer) e termina no último comando. Esta análise é simples: basta olhar o algoritmo e prever o comportamento do programa que o implemente.

Nas outras vezes (e muito mais frequentemente) o algoritmo prevê um ou mais de um ciclos de repetição dentro do código. Nesses momentos pode-se lançar mão de dois mecanismos específicos de repetição: a iteração e a recursão. São visões intercambiáveis (já que quase sempre uma pode ser transformada na outra), ambas devem ser conhecidas e o bom programador deve estar apto a usar uma ou outra, e a converter algoritmos de um tipo para os do outro tipo.

A implementação iterativa usa comandos de repetição (são eles: *enquanto...fimenquanto*, *repita...até e para...fimpara*. Além do desvio condicional (este mais *fora de moda*).

A implementação recursiva usa o truque de uma função, chamar a si própria sucessivas vezes, até que ocorra o caso básico – quando então deixa de haver a recursão.

Antes de estudar as diferenças, com a palavra Leigh Caldwell, do Stack Overflow: *Os loops (iteração) podem melhorar o desempenho de seu programa. A recursão melhora o desempenho do seu programador. Escolha o que for mais importante para a sua situação.*

Para entender bem esta diferenciação, vamos acompanhar o mesmo algoritmo implementado dos dois jeitos

```
1: função FATI(N: inteiro)
2: R, I: inteiro
3: R ← I ← 1
4: enquanto I ≤ N
5:   R ← R × I
6:   I ← I + 1
7: fim{enquanto}
8: retorne R
9: fimfunção
1: função FATR(N: inteiro)
2: se N < 2
3:   retorne 1
4: senão
5:   retorne N × FATR(N-1)
6: fim{se}
7: fimfunção
```

Desempenho Dado um algoritmo, deve-se estudar qual o seu desempenho. Qual a unidade de medida do desempenho ? São duas, em geral intercambiáveis: o tempo e o espaço requeridos pelo algoritmo. Usa-se – em geral – o tempo como medida universal de desempenho. Mas o que se precisa não é um valor exato (pois isto dependeria de implementar o dito algoritmo) e sim uma ordem de grandeza. Esta análise tem que ser feita ANTES de implementar o algoritmo. Se for feita depois, perde o seu sentido. Esta análise é chamada também de estudo da complexidade do algoritmo. A questão é como conhecer o desempenho de um algoritmo sem que ele exista implementado ?

De cara, salta aos olhos que este desempenho depende de pelo menos duas variáveis: o próprio algoritmo e o tamanho da entrada. Então para juntar as duas, vai-se medir quantas operações um dado algoritmo para uma dada entrada.

Novamente, precisa-se separar a análise nos dois casos estudados: algoritmos iterativos e recursivos

Complexidade de algoritmos iterativos

Diante da impossibilidade (e não necessidade) de um número exato, fazem-se as seguintes simplificações:

1. Considere apenas as linhas que realizam operações.
2. Considere apenas as linhas internas a laços de repetição.
3. Considere que cada operação sempre demora UMA unidade de tempo.

4. Um laço encima de outro deve ter tempos somados.
5. Um laço dentro de outro deve ter tempos multiplicados.
6. Desconsidere constantes multiplicativas.

Usando estas regras, admite-se a criação de funções matemáticas que explicam a complexidade: São elas: $O(n)$ - mede o pior caso, $\Omega(n)$ mede o melhor caso e $\Theta(n)$ mede o caso exato. Por razões de ordem prática, a função mais utilizada é a $O(n)$.

As classes mais comuns para a função $O(n)$ são:

classe	função
constante	$O(1)$
logaritmica	$O(\log(n))$
linear	$O(n)$
quadrática	$O(n^2)$
cúbica	$O(n^3)$
exponencial	$O(k^n)$
fatorial	$O(n!)$

A resposta nem sempre é evidente. Em muitos casos é difícil estabelecer a expressão algébrica da complexidade. Veja-se alguns exemplos:

Exemplos

Veja-se, por exemplo, alguns casos, sendo o primeiro o de uma função que recebe um vetor e soma todos os valores desse vetor.

```
1: função SOMA(X[n] : inteiro) : inteiro
2: S, I : inteiro
3: I ← 1
4: S ← 0
5: enquanto I ≤ n
6:   S ← S + X[I]
7: fim{enquanto}
8: retorne S
9: fimfunção
```

Este algoritmo tem $O(n)$ (e também $\Omega(n)$ e $\Theta(n)$).

Seja agora um algoritmo que ordena um vetor, o chamado algoritmo de ordenação por seleção:

```
1: inteiro I,J,COR,INUI,AUX
2: TAM ← Tamanho do vetor a ordenar
3: I ← 1
4: enquanto (I ≤ TAM)
5:   COR ← V[I]
6:   INUI ← I
7:   J ← I + 1
8:   enquanto (J ≤ TAM)
9:     se V[J] < COR
10:      COR ← V[J]
11:      INUI ← J
12:   fim{se}
13:   J ++
14:   fim{enquanto}
15:   AUX ← V[I]
16:   V[I] ← V[INUI]
17:   V[INUI] ← AUX
18:   I ++
19: fim{enquanto}
```

No segundo algoritmo (ordenação por seleção), o menor elemento é procurado (linhas 8 a 14). Encontrado (agora ele é apontando por *INUI*, ele é trocado de lugar com o elemento apontado por *I* (linhas 15 a 17). Note que há uma estrutura de repetição dentro de outra, pelo que a complexidade deste algoritmo é $O(n^2)$.

Um terceiro exemplo é o algoritmo de Floyd-Warshall que calcula todos os caminhos mínimos dentro de um grafo:

```
1: Algoritmo FLOYD-MOD
2: C[n,n] ← matriz de custos.
3: R[n,n] ← 0
4: Gerar K, a partir de C (troque 0 por infinito e ponha 0 na DP)
5: para Q de 1 até N
6:   para S de 1 até N
7:     para T de 1 até N
8:       se K[S][Q] + K[Q][T] < K[S][T]
9:         K[S][T] ← K[S][Q] + K[Q][T]
10:      se R[S][Q] = 0
11:        R[S][T] ← Q
12:      senão
13:        R[S][T] ← R[S][Q]
14:      fim{se}
15:    fim{se}
16:  fim{para}
17: fim{para}
18: fim{para}
19: fim algoritmo
```

Este algoritmo tem complexidade $O(n^3)$.

Complexidade de algoritmos recur-

sivos Agora, temos uma dificuldade adicional: apenas olhando o código recursivo não é possível inferir qual vai ser o comportamento do algoritmo. Agora, se necessita de uma ferramenta matemática chamada *relação de recorrência*. A recorrência é um método para definir sucessões. Estas são conjuntos de equações de 2 tipos: **equação de fronteira** cujo formato é $f(k) = c$ e **geral** de formato $f(n) = \text{expressão envolvendo } f(i) \text{ com } i < n$.

Nesse sentido, resolver uma recorrência é achar uma fórmula fechada para a função pedida, isto é, sem envolver outras chamadas à função. Veja-se um exemplo:

$$\begin{cases} a_0 = 0 \\ a_{n+1} = a_n + 2n \end{cases}$$

Tablando valores possíveis, tem-se

n	a _n
0	0
1	0
2	2
3	6
4	12
5	20

Para ajudar o raciocínio, insere-se uma coluna de n^2 .

n	n ²	a _n
0	0	0
1	1	0
2	4	2
3	9	6
4	16	12
5	25	20

A partir daqui fica facil descobrir que $f(n) = n^2 - n$.

Para encerrar vamos estudar o (talvez principal) algoritmo recursivo: o QUICK-SORT:

```
1: função RAP(inteiro I, F)
2: inteiro ANT,DEP,MEI,AUX
3: ANT ← I
4: DEP ← F
5: MEI ← V[(I + F) ÷ 2]
6: repita
7:   enquanto (V[ANT] < MEI)
8:     ANT ++
9:   fim{enquanto}
10:  enquanto (V[DEP] > MEI)
11:    DEP --
12:  fim{enquanto}
13:  se (ANT ≤ DEP)
14:    AUX ← V[ANT]
15:    V[ANT] ← V[DEP]
16:    V[DEP] ← AUX
17:    ANT ++
18:    DEP --
19:  fim{se}
20: até (ANT > DEP)
21: se (I < DEP)
22:   RAP (I, DEP)
23: fim{se}
24: se (ANT < F)
25:   RAP (ANT, F)
26: fim{se}
27: fim função
28: TAM ← Tamanho do vetor a ordenar
29: RAP (1,TAM)
```

Tente visualizar o desempenho deste algoritmo para uma entrada de tamanho n .

Para você fazer

Esta folha tem como tarefa um projeto de pesquisa. Procure um par de algoritmos equivalentes (mesma entrada $\Rightarrow$ mesma saída) sendo um iterativo e outro recursivo: (busca linear e busca binária; ordenação por seleção e ordenação quick; transformada de Fourier e transformada rápida de Fourier; multiplicação de matrizes e algoritmo de Strassen; busca da mediana e quickselect; busca do determinante para $n > k$ com k grande; etc etc) e implementando ambas as soluções e daí uma análise numérica de ambos os casos reais. O resultado final é uma conclusão sobre a implementação, além de um excelente material para um *paper* científico. Boa sorte !

Alto e Baixo nível na programação I

Na apresentação de linguagens de programação costuma-se falar (escrever) sobre o nível da linguagem: **baixo** quando estamos mais próximos do hardware e suas complexidades e **alto** quando – pelo contrário – se está mais próximo do problema, pensando e atuando de maneira mais abstrata, sem os detalhes finos da implementação e preocupados com detalhes teóricos do que se pretende fazer. Fica óbvio nas frases acima a dificuldade de estabelecer a diferença entre os níveis e portanto esta folha vai procurar apresentar casos práticos destas diferenças. A idéia é que a prática mostre as diferenças. Como sempre (citando Aristóteles), fazer é melhor e mais útil do que apenas falar.

Nossa prática vai começar pela implementação do conceito de **conjunto** noção vital na matemática. É uma idéia fundamental que precisa de uma definição ainda que informal ou não rigorosa. Então vamos lá:

Um conjunto é uma coleção de itens primitivos. O conjunto pode ser nomeado, em geral por uma letra maiúscula e os seus elementos podem ser descritos por letras minúsculas entre chaves. Por exemplo $A = \{a, b, c, d, e\}$. Outra maneira de descrever os elementos é dar a sua lei de formação, por exemplo $S = \{segunda, terca, quarta, quinta, sexta, sábado, domingo\}$ ou $S = \{dias da semana\}$. Ou então $I = \{0, 2, 4, \dots\}$ que é igual a $I = \{natural e par\}$. Note que quando vai-se enumerar um conjunto infinito, pela impossibilidade de citar todos os elementos, usam-se as reticências

No conjunto não pode haver itens repetidos e a ordem deles no conjunto não pode ter importância, pelo que os conjuntos $A = \{1, 2, 3\}$ e $B = \{1, 3, 2\}$ são iguais ou $A = B$.

Uma relação importante em conjunto é a de **pertença** identificada pelo símbolo $\in$ e a sua negação pelo símbolo $\notin$. Então pode-se dizer, por exemplo se $P = \{pares\}$ tem-se $2 \in P$ e também $3 \notin P$.

Para terminar esta parte introdutória, eis alguns conjuntos de números importantes na matemática

Naturais $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.

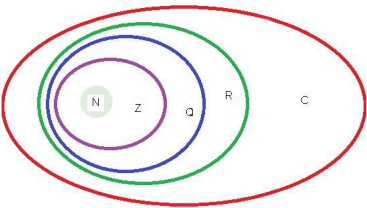
Inteiros $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$.

Racionais $\mathbb{Q} = \{\text{Conjunto das frações } a/b, \text{ onde } a \in \mathbb{Z} \text{ e } b \in \mathbb{Z}^* \text{ ou seja } b \neq 0\}$.

Reais $\mathbb{R} = \{\text{Conjunto dos racionais mais as decimais não exatas e não periódicas, chamados números irracionais}\}$.

Complexos $\mathbb{C} = \{\text{Números na forma } a + b.i, \text{ onde } a \text{ e } b \in \mathbb{R} \text{ e } i = \sqrt{-1}\}$.

Note que para estes conjuntos $\mathcal{N} \subset \mathcal{Z} \subset \mathcal{Q} \subset \mathcal{R} \subset \mathcal{C}$ ou usando diagramas de Venn



Implementação

Vai-se programar, primeiro em baixo nível e depois em alto, na mesma linguagem (Python) as funções necessárias para criar uma coleção de conjuntos, depois as funções de inclusão e exclusão, e depois as operações de conjuntos a saber: união, interseção, diferença. Antes disso, deve-se criar a função de listagem dos elementos de um dado conjunto. Ao final, poder-se-á olhar as diferenças do alto e do baixo nível.

Criação Para este exercício, vai-se definir uma coleção de até 8 conjuntos numéricos inteiros podendo cada um ter o máximo de 10 elementos. Naturalmente estes números são muito pequenos, mas podem ser aumentados à vontade. Usar-se-á um truque bastante comum em conjuntos de número variável de elementos que é agregar um quantificador a cada conjunto. Assim, o elemento [0] de cada conjunto será um quantificador de elementos desse conjunto. Por isso há 11 colunas (e não 10) na tabela. Ou o que em Teoria dos Conjuntos é conhecido como *cardinalidade* do conjunto.

```
def criacon():
    import numpy as np
    global C
    C = np.zeros((8,11),int)
    criacon()

def insere(conj,ele):
    if ele in C[conj,1:]:
        return
    if type(ele)==int and conj<8:
        aumenta(conj,1)
        C[conj,C[conj,0]]=ele

def exclui(conj,ele):
    if C[conj,0]==0:
        return
    i=1
    while C[conj,i]!=ele:
        i=i+1
        if i>C[conj,0]:
            return
    while i<C[conj,0]:
        C[conj,i]=C[conj,i+1]
        i=i+1
    aumenta(conj,-1)

def uniao(cr,ca,cb):
    C[cr]=C[ca]
    i=1
    while i<=C[cb,0]:
        if C[cb,i] not in C[cr,1:C[cr,0]+1]:
            aumenta(cr,1)
            C[cr,C[cr,0]]=C[cb,i]
        i=i+1

def inter(cr,ca,cb):
    i=1
    while i<=C[ca,0]:
        if C[ca,i] in C[cb,1:C[cb,0]+1]:
            aumenta(cr,1)
            C[cr,C[cr,0]]=C[ca,i]
        i=i+1

def dif(cr,ca,cb): #a-b: não é comutativa
    i=1
    while i<=C[ca,0]:
        if C[ca,i] not in C[cb,1:C[cb,0]+1]:
            aumenta(cr,1)
            C[cr,C[cr,0]]=C[ca,i]
        i=i+1

def dif_aumenta(li,qt):
    C[li,0]=C[li,0]+qt
    if C[li,0]>10 or C[li,0]<0:
        print("erro")
```

Alto nível Usando o mesmo Python (versão 3), agora vai-se ver como é muito mais fácil usar funções, métodos e estruturas primitivas da linguagem. De cara, percebe-se que não há necessidade de estabelecer tetos de quantidades ou elementos. Enquanto houver memória, os conjuntos podem crescer. Outro detalhe é que não haverá erros: o Python garante. Toda uma preocupação lá no baixo nível deixa de haver. A criação de um conjunto agora é

```
R = set() # criando vazio OU
R = {1,5,88,90} #criando com elementos
```

Note que não se pode criar um conjunto vazio em Python fazendo $X=\{\}$ pois esta construção comanda a criação de um dicionário vazio. Um dicionário NÃO é a mesma coisa que um conjunto. Aqui precisamos lidar com a falta de símbolos no teclado...

A inclusão/exclusão de elementos em R obedece:

```
R.add(elemento) # inclusão
R.remove(elemento) # exclusão
```

A limpeza de um conjunto é `R.clear`. As operações de conjuntos são `|` união, `&` interseção e `-` diferença. Aparece aqui uma novidade que é a diferença simétrica (`^` que responde com o conjunto dos elementos que não são comuns aos dois conjuntos). Além disso, há vários métodos adicionais para lidar com conjuntos. A cardinalidade de um conjunto pode ser obtida com `print(len(conj))`.

Comparando Suponha criar o conjunto 0 = {5, 12, 13, 6, 7}, o conjunto 1 = {1, 2, 3, 4, 5, 6} e ainda 2 = 0 ∩ 1, 3 = 0 ∪ 1, 4 = 1 - 2 e 5 = 2 - 1. Em baixo nível

```
insere(0,5)
insere(0,12)
insere(0,13)
insere(0,6)
insere(0,7)
insere(1,1)
insere(1,2)
insere(1,3)
insere(1,4)
insere(1,5)
insere(1,6)
inter(2,0,1)
uniao(3,0,1)
dif(4,0,1)
dif(5,1,0)
print(C)
[[ 5 12 13 6 7 0 0 0 0 0 0]
 [ 6 1 2 3 4 5 6 0 0 0 0]
 [ 2 5 6 0 0 0 0 0 0 0 0]
 [ 9 5 12 13 6 7 1 2 3 4 0]
 [ 3 12 13 7 0 0 0 0 0 0 0]
 [ 4 1 2 3 4 0 0 0 0 0 0]
 ...]
```

Em alto nível

```
c0 = {5,12,13,6,7}
c1 = {1,2,3,4,5,6}
c2 = c0 & c1
c3 = c0 | c1
c4 = c0 - c1
c5 = c1 - c0
print('c0 = ',c0)
... impressão de todos eles

c0 = {5, 6, 7, 12, 13}
c1 = {1, 2, 3, 4, 5, 6}
c2 = {5, 6}
c3 = {1, 2, 3, 4, 5, 6, 7, 12, 13}
c4 = {12, 13, 7}
c5 = {1, 2, 3, 4}
```

Para você fazer

Escolha a estratégia preferida (alto ou baixo nível; ou até as duas para conferir e verificar) e responda aos seguintes comandos:

Crie os conjuntos $R1 = \{2, 9, 10, 5, 6, 3\}$ e $R2 = \{6, 7, 3, 5, 4, 2\}$ e depois calcule

- 1. $R3 = R1 \cap R2$
- 2. $R4 = R1 \cup R2$
- 3. $R5 = R4 - R1$
- 4. $R6 = R5 \cup R3$
- 5. $R7 = R1 - R6$

Responda a somatória dos elementos de cada conjunto. Se usar a implementação em baixo nível NÃO some a cardinalidade:

1	2	3	4	5
$\sum R3$	$\sum R4$	$\sum R5$	$\sum R6$	$\sum R7$

Alto/baixo nível programação II

Nossa prática vai pela implementação de estruturas elementares, a saber: pilhas e filas. Para deixar a prática mais realista, vão-se usar duas linguagens distintas para cada abordagem. A de baixo nível usará *C* enquanto a de alto nível será feita em *Python*. Ambas estruturas são conhecidas como listas, e a única distinção entre elas se dá pela maneira como os dados entram e saem de cada uma.

Pilha Uma pilha é uma estrutura linear cuja entrada e saída sempre se dá em uma única extremidade. Ela implementa o protocolo LIFO (*last – in – first – out*) ou seja, o primeiro a sair é o último que entrou. Fica fácil entender se pensarmos em uma PILHA de pratos, de pneus, de rosquinhas etc.

Muito frequentes na programação, eis alguns exemplos de uso de pilhas:

- na notação polonesa reversa (*RPN = Reverse polish notation*) permite escrever expressões numéricas sem parênteses e principalmente sem a necessidade de tabelas de prioridade entre operações. Este algoritmo é usado sempre que um programa precisa calcular uma expressão.
- no caminhamento de um robot (ou softbot) em um labirinto, usando a estratégia de Te-seu com o novelo de Ariadne.
- para determinar a envoltória convexa de um conjunto de pontos, algoritmo importante em geometria computacional
- dentro de um compilador para determinar a ida e a volta de subrotinas
- na visitação de nodos de um grafo, usando a estratégia de busca em profundidade
- no processamento de scripts no pagamento-recepção de bitcoins

Na implementação de baixo nível de uma pilha há duas maneiras distintas: usando alocação sequencial e usando alocação encadeada. Na primeira usa-se um vetor para armazenamento dos dados. Aqui define-se uma estrutura base (o vetor) sobre o qual residirá a pilha. No momento da criação determina-se o tamanho máximo que a pilha terá (o tamanho do vetor). Os elementos têm contiguidade, pelo que a busca do próximo é muito fácil: o vizinho. Além disso, a busca do *k*-ésimo elemento do vetor é instantânea, basta calcular sua posição. Na alocação encadeada não há limites máximos (apenas o limite teórico da memória) mas não há contiguidade, pelo que cada elemento tem que ter o endereço de seu vizinho e chegar o elemento *k* implica em percorrer todos os itens anteriores a *k*. Nesta folha só a alocação sequencial vai ser estudada.

Eis a implementação em baixo nível de uma pilha sequencial usada para guardar inteiros e com tamanho máximo de 100 inteiros.

```
#include<stdio.h>
int topo; // o topo contém o número de itens
int elem[100];
int empil(pilha P, int x){
    if (topo >= 100) {
        return -1;
    }
    elem[topo]=x;
    topo=topo+1;
    return 0;
}
int desempil(pilha P){
    if (topo <=0) {
        return -1;
    }
    topo=topo-1;
    return elem[topo];
}
main(int ac, char *av[]) {
    topo=0;
    empil(...);
    ... }
```

Fila A fila também é uma estrutura linear cujo protocolo agora é o FIFO o que significa *first in first out* ou seja quem sai é o primeiro que entrou.

Basta pensar em uma fila de ônibus: quando o veículo chega o primeiro a embarcar é quem chegou primeiro na fila. Sempre que há mais consumidores do que recursos, forma-se uma fila. Este fato foi estudado e generalizado pelo engenheiro dinamarquês A. Erlang, o que deu origem a um notável ramo da matemática aplicada denominado Teoria das Filas. A unidade em uma fila é *erlang*. Eis alguns exemplos de uso de filas em ambientes computacionais

- em simulações, sempre que há espera no uso de recursos
- em filas de impressão: diversos computadores querendo imprimir em uma única impressora
- na ordenação topológica: listas de atividades com relações de dependência entre elas
- no caminhamento em grafos com a busca em largura.

Eis uma implementação (*naive:ingênua*) de uma fila. Para ultrapassar a ingenuidade, veja as filas circulares, a seguir.

```
#include<stdio.h>
int entrada,saida;
int elem[100];
int enfilei(fila F, int x) {
    int aux,limite;
    if (entrada<100) {
        elem[entrada]=x;
        entrada=entrada+1;
    }
    else {
        if (saida>0) {
            limite=entrada-saida;
            for (aux=0;aux<limite;aux++){
                elem[aux]=elem[saida+aux];
            }
            entrada=entrada-saida;
            saida=0;
            enfileira(x);
        }
        else {
            return -1; // erro de overflow
        }
    }
}
int desenfilei(fila F){
    int aux;
    if (saida>=entrada){
        return -1; } // erro de underflow
    else {
        aux=elem[saida];
        saida=saida+1;
        if (saida == entrada){
            saida=entrada=0;
        }
        return aux;
    }
}
main(int ac, char *av[]) {
    entrada=saida=0;
    enfileira(...);
    ... }
```

Fila circular Visando evitar o *passinho à frente* necessário na implementação anterior quando os elementos da fila precisam ser deslocados em direção à entrada e que é fonte de significativo *overhead*, pode-se melhorar o código através do conceito de uma fila circular

```
#include<stdio.h>
int entrada,saida,qtd;
int elem[100];
int enfileirc(fila F, int x) {
    int aux,limite;
    if (qtd<100) {
        entrada=entrada+1;
        if (entrada>100){
            entrada=1;
        }
        qtd=qtd+1;
        elem[entrada-1]=x;
    }
    else{
        return -1; // overflow
    }
}
int desenfileirc(fila F){
    if (qtd>0){
        saida=saida+1;
        if (saida>100){
            saida=1;
        }
        qtd=qtd-1;
    }
```

```
return elem[saida-1];
}
else{
    return -1; // underflow
}
}
main(int ac, char *av[]) {
    entrada=saida=qtd=0;
    enfileirc(...)
    ... }
```

Alto nível Usando o Python (v.3), agora vai-se ver como é muito mais fácil usar funções, métodos e estruturas primitivas da linguagem.

```
pilha=[]
def empilha(x):
    pilha.append(x)
def desempilha():
    return pilha.pop()
empilha(...)
...
fila=[]
def enfileira(x):
    fila.append(x)
def desenfileira():
    return fila.pop()
enfileira(...)
...
```

Exemplo Usando a mesma definição do "para você fazer", (2 pilhas, 2 filas, todas com capacidade para 5 letras pedindo-se ao final a próxima letra a sair, em cada caso), eis um exemplo feito:

```
1. Empil(P2,J)          11. Empil(P2,C)
2. Empil(P1,C)          12. Empil(P1,K)
3. Enfilei(F2,L)        13. Enfilei(F2,Desempil(P1))
4. Enfilei(F1,F)        14. Enfilei(F2,Desempil(P2))
5. Empil(P1,Y)          15. Enfilei(F2,Desempil(P1))
6. Enfilei(F1,K)        16. Empil(P1,Desenfilei(F1))
7. Enfilei(F1,X)        17. Empil(P2,Desenfilei(F2))
8. Enfilei(F1,Z)        18. Enfilei(F1,Desempil(P2))
9. Enfilei(F1,N)        19. Empil(P2,Desenfilei(F1))
10. Empil(P2,D)         20. Enfilei(F2,Desempil(P1))
A resposta é: C K X K
```

👉 Para você fazer

Usando a estratégia preferida (alto ou baixo nível) e possivelmente usando ambas – para conferir, crie 2 pilhas (P1 e P2) e duas filas (F1 e F2) com espaço limitado para 5 letras em cada. Depois implemente as 20 operações a seguir:

```
Enfilei(F2,A)
Enfilei(F2,J)
Enfilei(F1,G)
Enfilei(F1,J)
Empil(P2,X)
Empil(P1,X)
Enfilei(F2,R)
Empil(P1,K)
Empil(P1,W)
Enfilei(F2,K)
Empil(P1,J)
Empil(P2,J)
Empil(P2,Desenfilei(F1))
Enfilei(F1,Desempil(P2))
Empil(P2,Desenfilei(F1))
Enfilei(F2,Desempil(P2))
Empil(P2,Desenfilei(F2))
Empil(P2,Desenfilei(F2))
Enfilei(F1,Desempil(P2))
Enfilei(F2,Desempil(P2))
```

E, a seguir, responda quais seriam as próximas saídas em P1, P2, F1 e F2

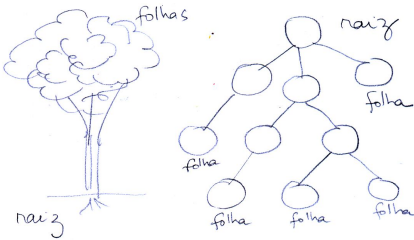
P1	P2	F1	F2



- 1 - /

Alto/baixo nível programação III

A estrutura de dados conhecida como árvore é muito frequente e importante na ciência da computação. Ela permite criar relações hierárquicas entre informações, acesso muito rápido a dados desordenados, além de estar por trás de muitos dos algoritmos fundamentais da ciência. O nome árvore deriva de uma possível similaridade com uma árvore vegetal. Trata-se da reunião de agrupamentos de dados – que aqui recebem o nome de nodo. Na árvore todos os nodos são estruturalmente iguais. Todos os nodos têm um pai (nodo do qual derivam), exceto o nodo inicial da árvore, chamado raiz. Na extremidade oposta, nodos que não têm filhos são chamados nodos folha. Diferentemente da botânica, na computação as árvores são desenhadas de cabeça para baixo, a raiz fica encima e as folhas embaixo.



Nossa prática vai pela implementação da estrutura conhecida como Árvore de Pesquisa Binária ou para reduzir: ABP. Para deixar a prática mais realista, vão-se usar duas linguagens distintas para cada abordagem. A de baixo nível usará C enquanto a de alto nível será feita em Python. Uma ABP é usada para acessar dados de maneira veloz. Cada dado terá um atributo único pelo qual será pesquisado, é a chamada *chave*. Por ser binária, nesta árvore cada nodo pode ter até 2 filhos. A regra de sua construção é: dada a chave do pai, se o filho tiver chave menor, ele será o filho *esquerdo* e se a chave do filho for maior, ele será o filho *direito*.

Baixo Nível

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
typedef struct no {
    struct no *fe;
    struct no *fd;
    int valor;
} nodo;

nodo* criano(int valor){
    nodo *no = (nodo*) malloc (sizeof(nodo));
    no->fe = NULL;
    no->fd = NULL;
    no->valor = valor;
    return no;
}

void preo(nodo *raiz){
    if (raiz == NULL) {return;}
    printf("%i ",raiz->valor);
    preo(raiz->fe);
    preo(raiz->fd);
}

void emo(nodo *raiz){
    if (raiz == NULL) {return;}
    emo(raiz->fe);
    printf("%i ",raiz->valor);
    emo(raiz->fd);
}

void poso(nodo *raiz){
    if (raiz == NULL) {return;}
    poso(raiz->fe);
    poso(raiz->fd);
    printf("%i ",raiz->valor);
}

int insarv(nodo *raiz, int valor){
    nodo *onde,*pai;
    int esqdir = 0;
    nodo *aux = criano(valor);
    onde = raiz;
    while (onde != NULL) {
        if (onde->valor == valor) {
            return -1; // ja existe: erro
        }
        pai = onde;
```

```
if (valor > onde->valor){
    esqdir = 1;
    onde = onde->fd;
}
else {
    esqdir = 0;
    onde = onde->fe;
}
}
if (esqdir==0) {pai->fe = aux;}
else {pai->fd = aux;}
return 0;
}

nodo* excarv(nodo *raiz, int valor){
    nodo *pai,*aaux,*elemx,*save,*tempo;
    int esqdir=0;
    aaux=pai=raiz;
    while(valor != aaux->valor){
        pai=aaux;
        if (valor>aaux->valor){
            esqdir=1;
            aaux=aaux->fd;
        }
        else {
            esqdir=0;
            aaux=aaux->fe;
        }
    }
    elemx=aaux;
    if ((elemx->fe == NULL)
        && (elemx->fd == NULL)) {
        if (esqdir == 0){
            pai->fe = NULL;}
        else {
            pai->fd = NULL;}
        return elemx;
    }
    else{
        if (elemx->fe == NULL){
            if (esqdir==0){
                pai->fe = elemx->fd; }
            else{
                pai->fd = elemx->fd; }
            return elemx;
        }
        else {
            elemx = elemx->fd;
            while (elemx!=NULL){
                save=elemx;
                elemx=elemx->fe;
            }
            tempo=excarv(raiz,save->valor);
            aaux->valor = tempo->valor;
            return aaux;
        }
    }
}

void main(int ac, char av[]){
    nodo* aa;
    int lixo;
    aa=criano(100);
    lixo=insarv(aa,1);
    ...
}
```

Alto nivel

```
# ABP em alto nivel
import numpy as np
global abp,ultimo
abp=np.zeros((100,3),int)
ultimo=-1
def emorder(raiz):
    if raiz==-1:
        return
    emorder(abp[raiz,0])
    print(abp[raiz,2])
    emorder(abp[raiz,1])

def inclabp(raiz,valor):
    global abp,ultimo
    ultimo=ultimo+1
    abp[ultimo]=[-1,-1,valor]
    if ultimo==0:
        return 0
    onde=raiz
    while onde!= -1:
```

```
pai=onde
if valor>abp[onde,2]:
    esqdir=1
    onde=abp[onde,1]
else:
    esqdir=0
    onde=abp[onde,0]
abp[pai,esqdir]=ultimo
return ultimo

def excluiabp(raiz,valor):
    esqdir=0
    aaux=pai=raiz
    while valor!=abp[aaux,2]:
        pai=aaux
        if valor>abp[aaux,2]:
            esqdir=1
            aaux=abp[aaux,1]
        else:
            esqdir=0
            aaux=abp[aaux,0]
    elemx=aaux
    if abp[elemx,0]==-1 and abp[elemx,1]==-1:
        abp[pai,esqdir]=-1
        return elemx
    else:
        if abp[elemx,0]==-1:
            abp[pai,esqdir]=abp[elemx,1]
            return elemx
        else:
            if abp[elemx,1]==-1:
                abp[pai,esqdir]=abp[elemx,0]
                return elemx
            else:
                elemx=abp[elemx,1]
                while elemx!=-1:
                    save=elemx
                    elemx=abp[elemx,0]
                tempo=excluiabp(raiz,abp[save,2])
                abp[aaux,2]=abp[tempo,2]
                return aaux
    inclabp(0,100)
```

Comparando À parte as diferenças entre C e Python, salta aos olhos na implementação em alto nível o uso de cursores em oposição aos apontadores do baixo nível. A grande vantagem desta segunda abordagem (cursores) é que havendo alguma dúvida a qualquer momento a tabela de cursores pode ser impressa (em poucas páginas) e os algoritmos podem ser depurados e certificados. Isso é quase impossível ao se usar apontadores. Havendo a necessidade de eficiência, o que indica para o uso de apontadores, uma estratégia interessante é programar com cursores, depurar, depurar, depurar, certificar e depois traduzir (com muito cuidado) para apontadores.

Para você fazer

Usando qualquer uma das abordagens, crie uma abp vazia e nela faça as operações

```
incluiabp(29)
incluiabp( 5)
incluiabp(24)
incluiabp(16)
incluiabp( 3)
incluiabp( 9)
incluiabp(21)
incluiabp(15)
incluiabp(12)
incluiabp(27)
incluiabp(30)
excluiabp(9)
incluiabp(4)
excluiabp(21)
incluiabp(8)
excluiabp(15)
incluiabp(18)
excluiabp(12)
incluiabp(23)
excluiabp(27)
```

e depois faça um caminhamento em emordem e transcreva o resultado dos nodos visitados em

1	2	3	4	5	6	7	8	9	10



Alto/baixo nível programação IV

Esta folha é radical ao analisar as diferenças entre baixo e alto níveis onde eles são mais profundos: na linguagem de programação. Aqui, o baixo (baixíssimo) nível é representado pela linguagem Assembly, que é o mais próximo que humanos normais chegam da linguagem de máquina. Tem as seguintes características

- tradução à base de 1:1
- cada microprocessador e/ou microcontrolador tem o seu Assembly
- programas são extensos
- a depuração é complexa
- processos indexados demandam múltiplas operações, por exemplo

Veja-se um possível programa Alo Mundo feito para a arquitetura x8086:

```
lea si, string ; Atribui SI ao endereço string.
call printf ; empilha end. atual/chama printf
hlt ; Encerra o computador.
string db "Ola mundo!", 0
printf PROC
    mov AL, [SI] ;AL=endereço SI.
    cmp AL, 0 ;Compara AL com nulo.
    je pfend ;Pula se comparação der igual.
    mov AH, 0Eh
    int 10h ;chama BIOS p/impr.1 car. em AL.
    inc SI ;Incrementa em um o valor de SI.
    jmp printf ;Pula para o início do processo.
pfend:
    ret ;Retorna p/ endereço da pilha.
printf ENDP
```

Compare com o equivalente em Python

```
def alo():
    printf("Alo mundo")
alo()
```

Aqui uma explicação: a linguagem é Assembly e o programa montador (que traduz de Assembly para linguagem de máquina) é o Assembler. Se você estudar algum Assembly (existem dezenas, e este já é um grande problema), verá que eles seguem a arquitetura da máquina, basicamente como sugerida por Turing e principalmente Von Neumann, há mais de 70 anos, logo são linguagens de uma pobreza franciscana. É claro que com a complexidade crescente dos processadores as linguagens têm evoluído também, mas pode-se dizer que é um crescimento em largura não em profundidade: o repertório de ações ainda é bem restrito. Começa-se pela movimentação (na realidade: cópia) de dados na memória. Há diversos tipos de cópia: da memória para os registradores e vice-versa. Há muitos registradores: de segmento, de apontadores e de dados. Muitos deles ainda estão divididos em alta e baixa ordem. Há instruções de adição, subtração, multiplicação e divisão, sendo que ficam por conta do programador as condições de overflow e underflow, entre outras. Há macro instruções de entrada e saída, já que o contacto do programa é com o controlador da unidade acessada. Há também inúmeras operações de comparação, lembrando que há operações binárias, decimais e de ponto flutuante: o programador escolhe (e programa). Finalmente, há inúmeras operações de desvio do código, em geral dependendo de uma comparação recém efetuada. Resta a pergunta: porque alguém se jogaria nesta empreitada? duas possíveis respostas: para obter um desempenho excepcional ou então para ter acesso a algum detalhe absolutamente interno do ambiente. Uma terceira hipótese tem a ver com a possível saúde mental do programador, mas aí já estamos no campo da especulação.

Uma proposta de baixo nível Para adiantar esta folha, precisa-se de uma linguagem de baixo nível. É impossível usar algum assembly real. Só os detalhes da implementação demandariam páginas e páginas de detalhes. Então, vamos projetar e usar um substituto suficientemente compacto para tanto. Seguem-se as operações permitidas:

Movimentação Na verdade a movimentação é uma cópia: $A \leftarrow \text{expressão}$: a expressão é calculada e o seu resultado vai para a variável citada.

Aritmética Valem as regras da matemática, usando-se os caracteres + adição; - subtração; * multiplicação; / divisão; () hierarquia de operações, além das demais funções padrão (sen, cos, x^2 , $\sqrt{x}$, etc etc)

macros de E/S Usa-se-ão apenas duas instruções: **lea(A)** e **escreva(A)**, sem maiores detalhes de implementação.

Teste de condição O comando aqui é $\rightarrow(\text{condição})/\text{rótulo}$: a condição é avaliada e sendo verdadeira ocorre um desvio para o ponto do programa identificado com o rótulo

desvios No formato $\rightarrow \text{rótulo}$: força um desvio incondicional ao rótulo. Finalmente, em pontos determinados do código devem ser incluídos os rótulos, para estabelecer os pontos de destino dos desvios, no formato **rótulo:**.

Note-se a pobreza franciscana da estratégia de teste de condições: o resultado único possível é um desvio ou a sua ausência. Note-se também que não há nada parecido com qualquer tipo de repetição.

Proposta de alto nível

cabeçalho Bem simples, formado apenas pela palavra “algoritmo” seguida do nome do mesmo

Variáveis As variáveis do algoritmo devem ser especificadas logo após o cabeçalho do algoritmo. Haverá apenas o tipo inteiro, que por ser único não precisa ser especificado.

LEIA O comando **lea** ao ser compilado interromperá o fluxo do programa, gerará uma mensagem padrão “LEITURA DA VARIÁVEL nnn”, onde nnn é o nome da variável da lista, aceitará o valor a ela fornecido via teclado, assinalará este valor a ela e continuará o processamento.

IMPRIMA Aqui, ao ser traduzido este comando gerará uma impressão do programa, colocando antes dos valores a mensagem opcional e listando as variáveis cujos nomes fazem parte da lista. Não há interrupção no fluxo.

ASSINALAMENTO O comando de assinalamento tem o formato **receptor←expressão compatível**. Expressão pode ser também uma variável ou um valor.

SE Comando condicional. Tem o formato **se condição ações [senão ações2] fim{se}**. A condição é avaliada. Se ela for verdadeira, a lista de ações é executada. Se ela for falsa e houver a lista do senão (ações2) esta é executada. O comando sempre se encerra por um **fim{se}**.

ENQUANTO É um comando condicional usado para repetir blocos de código. Tem o formato **enquanto cond. lista-ações fim{enquanto}**. A condição é avaliada. Se ela for VERDADEIRA, a lista-ações é executada. Ao chegar ao final da lista, há um desvio ao começo do laço. Se ela for FALSA há um desvio para depois do **fim{enquanto}**.

REPITA Comando condicional. Tem o formato **repita lista-ações ate {condição}**. A lista é executada. Ao se localizar a cláusula **ate**, a condição é avaliada. Se ela for verdadeira, o fluxo segue normalmente. Se ela for falsa, há um desvio ao início do laço.

PARA Comando de repetição. Tem o formato **para var de valor1 ate valor2 [passo valor3] lista-ações fim{para}**. Controla laços finitos e previamente conhecidos, já que o próprio comando determina o valor inicial, o incremento e o teste de fim.

Compilação A seguir o ponto principal desta folha: a tradução da linguagem fonte para a linguagem objeto: Haverá uma pilha cujo elemento tem 22 caracteres (6 de identificação, 8 de variável do para e 8 de incremento do para). Estes dois últimos campos só são usados no para) Haverá uma coleção de variáveis locais, com nomes V00, V01, V02, etc. Elas são numéricas inteiras e servem tanto para efetuar cálculos, como para receber condições (0 é falso, 1 é verdadeiro), como para estabelecer pontos de chegada de desvios, o que em linguagem de

programação se conhece como “label”, desde que apareçam sozinhas no código seguido por “:”.

Na descrição a seguir, os números das variáveis são quaisquer. Usaram-se números diferentes apenas para facilitar o entendimento pelo leitor.

SE simples a condição é V01. Se V01 for falsa, desvie para V02. Empilhe V02. Traduza normalmente. Ao achar **fim{se}**, desempilhe V02 e aplique o rótulo no código.

SE composto a condição é V01. Se V01 for falsa, desvie para V02. Empilhe V02. Traduza normalmente. Ao achar **senão**, desempilhe V02 guardando-o em SALVA. Empilhe V03 e desvie para ela. Aplique SALVA no código. Traduza normalmente. Ao achar **fim{se}**, desempilhe V03 e aplique-o no código.

ENQUANTO Empilhar V04, Aplicar V04 no código. Empilhar V05. Calcular V06 com a condição. Se V06 é falso, desviar para V05. Traduzir normalmente. Ao achar **fim{enquanto}**, desempilhar o V05, guardando-o em SALVA, desempilhar o V04, desviar para V04, e aplicar SALVA no código.

REPITA aplicar V07 no código e empilhar V07. Traduzir. Ao achar “ate”, empilhar V08. Colocar a condição em V09. Se V09 for verdadeira, desempilhe V08 e desvie para V08. Desempilhe o V07 e desvie para V07. Aplique V08 no código.

PARA Fazer variável ← valor1. Empilhar V10 e aplicar V10 no código. Empilhar V11, variável, incremento. Colocar em V12 a condição variável ≤ valor2. (Se o passo é negativo, a condição é ≥). Se V12 for falso, desviar para V11. Traduzir normalmente. Ao achar o **fim{para}** consultar o topo, incrementar a variável, desempilhar V11 guardar em SALVA, desempilhar V10, desviar para lá. Aplicar SALVA no código.

Um exemplo completo A seguir, um programa devidamente “compilado”:

```
1: algoritmo TESTE
2: inteiro X,Y,Z,T
3: enquanto Y≠19
4:   se Y>8
5:     X← 9
6:     Z←T*14
7:     Y←15
8:   fim{se}
9: fim{enquanto}
10: finalgoritmo
cuja "compilação" deu como resultado o seguinte:
```

```
1: TESTE X Z Y V01 V02 V03 V04 V05 V06
2: inteiro X,Y,Z,T
3: V01:
4: V03 ← Y ≠ 19
5: → ( V03)/V02
6: V04 ← Y>8
7: → ( V04)/V05
8: X ← 9
9: Z ← T*14
10: Y ← 15
11: V05:
12: → V01
13: V02:
```

Para você fazer

Você deve “compilar” o programa a seguir e escrever o programa objeto:

```
algoritmo TESTE-13424
inteiro X,Y,Z,T
repita
    se T>= 8
        X<-11
        Z<-18
        T<-Y+Y
    fim{se}
ate Y= 19
fim{algoritmo}
```

Escreva o resultado no verso.



Alto/baixo nível programação V

Nesta folha vai-se estudar o conceito de matriz inversa e as diferenças imensas entre as implementações em baixo nível e em alto nível.

Matriz Inversa

A definição matemática da matriz inversa é $A \cdot A^{-1} = I$. Ou seja, ao multiplicar uma matriz pela sua inversa, obtem-se a matriz identidade I de mesma ordem. Nesse sentido a matriz inversa joga o mesmo papel do inverso no produto comum: Dado um número, digamos 7, seu inverso (1/7) é aquele que que quando multiplicado pelo número original reproduz a unidade: $7 \times \frac{1}{7} = 1$. Uma matriz quadrada pode não ser inversível, isto é não ter inversa. (Como o zero não tem inverso). Ela será não inversível quando seu determinante for igual a zero.

Propriedades

- 1. só existe A^{-1} se $\det(A) \neq 0$
- 2. A inversa da inversa é a própria matriz $(A^{-1})^{-1} = A$
- 3. $(n \cdot A)^{-1} = n^{-1} \cdot A^{-1}$
- 4. $(A_1 A_2 A_3 \dots A_n)^{-1} = A_n^{-1} \dots A_3^{-1} A_2^{-1} A_1^{-1}$
- 5. $(A^t)^{-1} = (A^{-1})^t$
- 6. $A \cdot A^{-1} = A^{-1} \cdot A = I$ (em geral $A \cdot B \neq B \cdot A$)
- 7. $I^{-1} = I$
- 8. $(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$

Usos 1. Ajuda a resolver sistemas lineares de maneira bastante elegante. De fato, a maioria dos sistemas lineares reais têm uma matriz A mais ou menos fixa (já que ela reflete as interdependências das variáveis na obtenção do resultado numérico final) e uma matriz B que varia de caso a caso. Nos métodos usuais, (Gauss, jacobi, LU...) este fato era desconsiderado e a cada nova instância do sistema tudo era recalculado, incluindo-se aí a matriz A .

Agora, A gera a sua inversa A^{-1} uma única vez. E daí, a cada novo B apenas este é recalculado, usando-se o seguinte desenvolvimento: $A \cdot x = B$, multiplicando-se ambos os lados por A^{-1} fica $A^{-1} \cdot A \cdot x = A^{-1} \cdot B$ e lembrando que $A \cdot A^{-1} = I$, fica $x = A^{-1} \cdot B$

2. Em um sistema linear, informa as respostas a determinados estímulos, ou seja funciona para entender as inter-relações de partes componentes. Sempre que um sistema linear for derivado de uma lei de conservação (massa, força, calor, momento, carga, quantidade de bits, ...). Neste caso, uma única equação pode ser escrita em cada ponto do sistema, resultando em um conjunto de equações que que define as propriedades do sistema. Em $A \cdot x = B$, x são os níveis da propriedade sendo balanceada em cada parte do sistema. B contém os elementos do balanço que são independentes do comportamento do sistema - são constantes. Em muitos casos, são os estímulos externos.

Formas de Calcular

- Existem inúmeras maneiras de calcular a matriz inversa como
- Usando a teoria, escrevendo um sistema linear multiplicando A por A^{-1} e igualando isto à matriz I . Depois, basta resolver o sistema linear.
 - Usando determinantes, não muito eficientes para sistemas grandes. Maiores detalhes olhar no excelente site <http://mathworld.wolfram.com>
 - Usando cofatores - Veja em p/n/159/matinv.pdf
 - Usando LU - a partir da decomposição $A = LU$
 - Usando Gauss-Jordan - como descrito a seguir

Implementação em baixo nível Este exercício pede que você implemente o algoritmo de Gauss-Jordan para cálculo da matriz inversa e depois o aplique a uma matriz de ordem suficientemente alta para desencorajar o cálculo manual da mesma. Relembrando as etapas, você deve:

1. Escrever a matriz A e ao seu lado a matriz I .
2. Aplicar operações elementares com linhas a A visando transformá-la em I .
3. Aplicar rigorosamente as mesmas operações sobre a matriz ao lado.
4. Quando A tiver se transformado em I , a matriz que era originalmente I se transformou em A^{-1} .
5. Para conferir, faça a multiplicação matricial entre A e A^{-1} . Tem que dar I .

Vamos aplicar este processo ao sistema:
 $[AI] = \begin{bmatrix} a_{0,0} & \dots & a_{0,n-1} & 1 & 0 & \dots & 0 \\ a_{1,0} & \dots & a_{1,n-1} & 0 & 1 & \dots & 0 \\ \dots & & \dots & \dots & \dots & \dots & \dots \\ a_{n-1,0} & \dots & a_{n-1,n-1} & 0 & 0 & \dots & 1 \end{bmatrix}$

O objetivo é produzir um zero no coeficiente do x na segunda equação: Para isso: $a[1][0] = a[1][0] + a[0][0] \times (-a[1][0]/a[0][0])$ onde a notação $a[1][0]$ deve ser entendida como a aplicação da operação à toda a linha [1] da matriz a . Mas, atenção: isto não vale no compilador C++. Lá, tem que ser por extensão.

Note-se também que o fator $(-a[1][0]/a[0][0])$ é o mesmo para toda a linha e portanto deve ser calculado antes e preservado durante todo o ciclo. O algoritmo vai chamar este fator de *pivot* e ele vai ser previamente calculado e guardado.

para calcular o zero no coeficiente de x na terceira linha, o comando $a[2][0] = a[2][0] + a[0][0] \times (-a[2][0]/a[0][0])$.

Para a segunda coluna, começa-se achando o zero na primeira linha. O comando agora é $a[0][1] = a[0][1] + a[1][1] \times (-a[0][1]/a[1][1])$ e depois $a[2][1] = a[2][1] + a[1][1] \times (-a[2][1]/a[1][1])$. A terceira coluna é

$a[0][2] = a[0][2] + a[2][2] \times (-a[0][2]/a[2][2])$ e $a[1][2] = a[1][2] + a[2][2] \times (-a[1][2]/a[2][2])$. Com estas operações, os zeros foram alcançados. Agora basta transformar os elementos da diagonal principal na unidade. As operações são: $a[0][0] = a[0][0]/a[0][0]$, $a[1][1] = a[1][1]/a[1][1]$ e $a[2][2] = a[2][2]/a[2][2]$.

Note que não houve preocupação em determinar se a matriz A é inversível. Isto foi deixado para não complicar demais o algoritmo. Fica como desafio para os mais corajosos.

Eis o algoritmo em C++:

```
#include<iostream>
#include<iomanip>
#define ta 5
using namespace std;
int cami(float x[ta][ta], float y[ta][ta]){
// x é a matriz A e y é a matriz A^-1
int i,j,k;
float pivot,alvo;
for (i=0;i<ta;i++){
for (j=0;j<ta;j++){
if (i==j) {
y[i][j]=1;
}
else {
y[i][j]=0;
}
}
}
}
for (i=0;i<ta;i++){
for (j=0;j<ta;j++){
if (j!=i){
pivot=-x[j][i]/x[i][i];
for (k=0;k<ta;k++){
x[j][k]=x[j][k]+x[i][k]*pivot;
y[j][k]=y[j][k]+y[i][k]*pivot;
}
}
}
}
alvo=x[i][i];
for (k=0;k<ta;k++){
y[i][k]=y[i][k]/alvo;
x[i][k]=x[i][k]/alvo;
}
}
return 0;
}
int main() {
float mat[ta][ta]={{{...},{...},{...}};
float res[ta][ta];
int i,j;
cami(mat,res);
for (i=0;i<ta;i++){
for (j=0;j<ta;j++){
cout<<fixed<<setprecision(5)
<<res[i][j]<<" ";
}
}
cout<<endl;
```

}
}
Observação importante: Este algoritmo é conhecido como algoritmo de Gauss Jordan *naïve* (ou ingênuo) pela simples razão que ele não prevê o que fazer se algum denominador for zero. Um algoritmo mais robusto, teria previsão para isto.

Em alto nível

APL Dada a matriz A, sua inversa é facilmente calculável. Acompanhe
 $A = \begin{bmatrix} 3 & 3p3 & 4 & ^{-}9 & 2 & 1 & 2 & 7 & ^{-}1 & 1 \end{bmatrix}$
A
 $\begin{bmatrix} 0.02173913043 & 0.03623188406 & 0.1231884058 & 0.08695652174 & 0.4782608696 & ^{-}0.1739130435 & ^{-}0.0652173913 & 0.2246376812 & ^{-}0.03623188406 \end{bmatrix}$

MATLAB

A função é $Y=inv(X)$. Acompanhe
 $X = \begin{bmatrix} 1 & 0 & 2; & -1 & 5 & 0; & 0 & 3 & -9 \end{bmatrix}$
 $Y = inv(X)$
 $Y = \begin{bmatrix} 0.8824 & -0.1176 & 0.1961 \\ 0.1765 & 0.1765 & 0.0392 \\ 0.0588 & 0.0588 & -0.0980 \end{bmatrix}$

Python-Numpy

O pacote Numpy é o de computação numérica no Python.

```
>>> import numpy as np
>>> A=np.array([[1,0,2],[-1,5,0],[0,3,-9]])
>>> A
array([[ 1,  0,  2],
       [-1,  5,  0],
       [ 0,  3, -9]])
>>> B=np.linalg.inv(A)
>>> B
array([[ 0.88235294, -0.11764706,  0.19607843],
       [ 0.17647059,  0.17647059,  0.03921569],
       [ 0.05882353,  0.05882353, -0.09803922]])
```

Python-Sympy

O pacote Sympy é o de computação simbólica.

```
>>> import sympy as sp
>>> a=sp.Matrix([[1,0,2],[-1,5,0],[0,3,-9]])
>>> a
Matrix([
[ 1, 0,  2],
[-1, 5,  0],
[ 0, 3, -9]])
>>> a.inv()
Matrix([
[15/17, -2/17, 10/51],
[ 3/17,  3/17,  2/51],
[ 1/17,  1/17, -5/51]])

>>> a.inv().n(6)
Matrix([
[ 0.882352941176471, -0.117647058823529,  0.196078431372549],
[ 0.176470588235294,  0.176470588235294,  0.0392156862745098],
[ 0.0588235294117647,  0.0588235294117647, -0.0980392156862745]])
```

Para você fazer

Usando o método preferido (ou ambos) ache a matriz inversa de

690	932	482	787	304	328	630	835	812
405	634	576	959	189	796	986	813	845
825	883	703	290	806	315	375	135	774
551	383	749	232	553	178	520	391	823
575	248	954	562	752	569	146	904	529
868	479	207	826	486	750	495	577	896
801	686	867	887	556	219	635	905	762
403	892	584	427	592	710	156	112	421
280	783	606	669	395	263	162	976	352

e responda o valor encontrado no centro (quinta linha e quinta coluna) da matriz inversa. Responda aqui



- 1 motor silencioso
- 2 acendedor de cigarros e de charutos
- 3 dispositivo para detectar radares de
velocidade
- 4 dispositivo para mudar a cor externa
do carro
- 5 capacidade de atravessar lagos e rios
- 6 tela de cinema a bordo com videocas-
sete e DVD
- 7 rádio de alcance mundial
- 8 veículo movido a água
- 9 vidros blindados à prova de balas
- 10 carro que nunca apresenta defeitos
mecânicos
- 11 dispositivo de auto orientação nas es-
tradas. Piloto automático
- 12 geração automática de mapas e loca-
lização por GPS
- 13 bancos de água
- 14 luz interna estroboscópica
- 15 pneus que nunca furam
- 16 chafariz interno
- 17 aceita comandos de voz
- 18 faz mais de 25 km com 1 litro de ga-
solina
- 19 não aceita barbeiragens do motorista
- 20 nunca colide de frente em ultrapassa-
gens mal feitas
- 21 ilumina a estrada com abundância
sem cegar o veículo que vem
- 22 nunca derrapa em curvas, mesmo com
pista molhada
- 23 pode ser bloqueado por comando via
satélite
- 24 pode ser localizado em qualquer parte
da terra
- 25 toalete a bordo com água quente
- 26 equipado com mesa de truco e gela-
deira de cerveja
- 27 cálculo e roteamento para o menor ca-
minho entre 2 pontos
- 28 não deixa o motorista dormir por
maior que seja o sono
- 29 nunca atropela gatos e outros felinos
- 30 nunca deixa o dono chegar atrasado a
qualquer compromisso
- 31 carro que nunca fura o sinal vermelho
- 32 carro que nunca ultrapassa a veloci-
dade permitida
- 33 carro que buzina automaticamente
quando ve uma moça bonitona
- 34 carro no qual cabe uma família de 6
filhos
- 35 carro que atende assobio do dono
- 36 carro que dá um jeito de nunca entrar
em filas

ALGORITMO VAMOS COMPRAR UNS
CARROS (Algoritmo guloso)

2. Seja o conjunto R de requisitos necessários
3. Seja V o menor conjunto de veículos M que cobre R
4. Retire de M o veículo que tenha a maior quantidade de requisitos esperados que pertençam a este veículo e a R simultaneamente
5. Coloque este veículo em V
6. Retire de R os requisitos deste veículo
7. Se R está vazio, o problema está encerrado com resposta em V , senão
8. Retorne a 4.

M = 1.	1	2	7	8	12	17	19	21	28	34
2.	1	15	21	25	28					
3.	8	9	11	21	27	30				
4.	7	14	25	28	30	31	32	34		
5.	7	8	10	15	26	29				
6.	3	4	10	21	25	34	36			
7.	1	10	13	16	17	19	20	36		
8.	5	13	19	33	36					
9.	10	13	15	16	19	33	35			
10.	3	9	13	16	17	25	27	29	35	36
11.	4	5	6	15	18	24	30			
12.	6	10	11	23	25	27	28	31		
13.	1	6	7	9	17	20	23	31	33	
14.	5	16	19	23	28	32	33	36		
15.	5	8	10	11	15	20	21	23	24	36
16.	2	13	15	16	21					
17.	5	15	21	24	27	28	29	31	33	35
18.	3	4	12	13	20	22	23	28		
19.	3	6	8	10	15	16	17	19	26	33
20.	7	12	17	20	26	29	32	33		

Logo, a resposta é o conjunto
 $V = \{4, 15, 1, 6, 17, 11\}$

👉 Para você fazer

```

R =
{ 1 3 4 5 7 8 12 13 15 18 19 20 21
 22 23 27 28 29 30 36 }
M =
1- 3 4 14 18 25 27 29 35 0 0
2- 11 13 15 26 29 36 0 0 0 0
3- 2 9 15 17 18 19 20 36 0 0
4- 4 5 9 14 21 23 24 25 0 0
5- 2 5 11 15 16 29 32 0 0 0
6- 8 11 12 16 20 23 29 30 31 35
7- 2 11 20 21 25 0 0 0 0 0
8- 12 13 23 26 32 34 0 0 0 0
9- 1 5 13 20 21 33 34 35 36 0
10- 5 8 9 10 15 16 17 22 24 0
11- 3 8 14 16 17 23 28 29 0 0
12- 3 6 21 25 27 28 0 0 0 0
13- 10 14 21 22 24 26 32 33 0 0
14- 4 20 21 28 30 0 0 0 0 0
15- 4 5 6 13 21 23 0 0 0 0
16- 11 13 29 30 31 35 0 0 0 0
17- 9 10 11 13 19 21 27 34 0 0
18- 2 8 10 13 32 34 0 0 0 0
19- 2 7 8 13 15 23 30 0 0 0
20- 4 12 15 28 32 0 0 0 0 0

```

1	2	3	4	5	6
7	8	9	10	11	12

Em C Construa aqui o algoritmo em C:

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

1

Exercício : 1

_____ / _____ / _____

Algoritmo guloso: habilidades de um jogador de futebol

Seja o seguinte problema real: Um time de futebol aqui de Curitiba, notou algumas deficiências em sua equipe. Tais deficiências foram obtidas a partir da tabela

hab.	descrição	peso
a	cabecear dentro da pequena área	25
b	fazer gols olímpicos em 20% das cobranças de escanteio	32
c	embaixadas com no mínimo 50 toques sem deixar a bola cair	46
d	bater penalty dentro do gol em 100% das cobranças	20
e	enervar os jogadores adversários sem baixaria	34
f	inflamar a torcida do próprio time	29
g	inflamar a torcida adversária	17
h	comandar o time dentro do campo	35
i	aplicar no mínimo 1 chapéu por jogo	28
j	bater faltas a distância com aproveitamento mínimo de 10%	27
k	fazer folha seca em 20% no mínimo de cobranças	48
l	jogar no mínimo 2 meses sem se contundir	33
m	errar menos de 40% dos passes efetuados	37
n	conseguir sucesso no mínimo em 35% das triangulações	13
o	converter no mínimo 80% dos penalties cobrados	44
p	conseguir correr 4,5 km no teste de Cooper	18
q	conseguir fazer 100m em 13seg ou menos	47
r	acertar uma chapa de 1m x 1m sobre o gol cabeceando em 50%	31
s	acertar o gol em no mínimo 60% das cobranças com barreira	43
t	errar menos de 20% dos passes de 50m efetuados	36
u	jogar 10 partidas sem tomar cartão amarelo	40
v	jogar 40 partidas sem tomar cartão vermelho	42
w	driblar sem ser desarmado em 60% das tentativas	15
x	conseguir a expulsão de um adversário	22
y	arte dramática aplicada ao jogo de futebol	50
z	ser campeão em 30% dos campeonatos disputados	38

Note que cada habilidade tem um valor específico no mercado futebolístico. Suponhamos então, que o time em questão notou precisar de a, f, t, u... Para atender a esta deficiência, o clube conta com um portfólio de jogadores, por exemplo

João tem as habilidades: a, s, t, w, j, c, y e z

José tem as habilidades: z, t, d, c, v, b, n, m, r, a, e, i e o

Paulo tem as habilidades: y, u, i, o, p, w, e e x

...

A pergunta é: qual o menor conjunto de jogadores que satisfazem a TODAS as habilidades que o clube necessita ?

Este problema pode ser resolvido por busca exaustiva, mas convém mencionar que a complexidade associada é 2^n , onde n é o número de jogadores (já que o número de subconjuntos de conjunto de cardinalidade n é 2^n). Para evitar esta complexidade inaceitável em casos de n grande, pode-se usar um ALGORITMO GULOSO. Este nem sempre trará a melhor resposta, mas sempre trará uma resposta, em geral aceitável.

Algoritmo guloso para habilidades de jogadores

1. Seja o conjunto P de jogadores, cada um com suas habilidades
2. Seja o conjunto S de habilidades necessárias
3. Seja R o menor conjunto de Ps que cobre S
4. Retire de P o jogador que tenha a maior soma nos custos das habilidades que pertencem a este jogador e a S simultaneamente
5. Coloque este jogador em R
6. Retire de S as habilidades deste jogador
7. Se S está vazio, o problema está encerrado com resposta em R, senão
8. Zere os custos das habilidades retiradas
9. Retorne a 4.

Um exemplo

Acompanhe o exemplo: (use a tabela de custos acima)

Seja S = {d e f n o q u v w x y z}
Seja P = {(f u), (q f z n), (z e), (x n), (w y z q), (d v q u), (x o w), (y d), (x z q e n), (e v), (z e), (e d)}
O primeiro vetor custos: 69 127 72 35 150 149 81 70 154 76 72 54
O melhor jogador é o 9 que vai para a saída. Todas suas habilidades são zeradas na tabela e retiradas de S e o processo recomeça.
O segundo vetor: 69 29 0 0 65 102 59 70 0 42 0 20
O melhor agora é o 6, zerando suas habilidades e retirando de S,
Terceiro vetor: 29 29 0 0 65 0 59 50 0 0 0 0, melhor = 5

Quarto vetor: 29 29 0 0 0 0 44 0 0 0 0 0, melhor = 7
Quinto vetor: 29 29 0 0 0 0 0 0 0 0 0 0, melhor = 1
e ao retirar suas habilidades de S, S fica vazio e o algoritmo se encerra com a resposta R = {9, 6, 5, 7, 1}

Observação: em caso de empate pegue o jogador de menor número.

Para você fazer

Seja a lista de habilidades:

hab.	descrição	peso
a	cabecear dentro da pequena area	24
b	fazer gols olímpicos em 20% das cobranças de escanteio	12
c	embaixadas com no mínimo 50 toques sem deixar a bola cair	32
d	bater penalty dentro do gol em 100% das cobranças	35
e	enervar os jogadores adversarios sem baixaria	36
f	inflamar a torcida do proprio time	25
g	inflamar a torcida adversaria	27
h	comandar o time dentro do campo	49
i	aplicar no minimo 1 chapéu por jogo	31
j	bater faltas a distancia com aproveitamento minimo de 10%	41
k	fazer folha seca em 20% no minimo de cobranças	29
l	jogar no minimo 2 meses sem se contundir	18
m	errar menos de 40% dos passes efetuados	23
n	conseguir sucesso no minimo em 35% das triangulacoes	40
o	converter no minimo 80% dos penalties cobrados	21
p	conseguir correr 4,5km no teste de Cooper	19
q	conseguir fazer 100m em 13seg ou menos	50
r	acertar chapa de 1m x 1m sobre o gol cabeceando em 50%	34
s	acertar o gol em no minimo 60% das cobranças com barreira	45
t	errar menos de 20% dos passes de 50m efetuados	11
u	jogar 10 partidas sem tomar cartao amarelo	13
v	jogar 40 partidas sem tomar cartao vermelho	47
w	driblar sem ser desarmado em 60% das tentativas	20
x	conseguir a expulsao de um adversario	42
y	arte dramatica aplicada ao jogo de futebol	16
z	ser campeao em 30% dos campeonatos disputados	28

Seja o conjunto S =

{a g h j k m p t u v x z }

Seja o conjunto P =

1-axh
2-tka
3-tjam
4-pzkv
5-vj
6-mvjt
7-guz
8-hj
9-up
10-tz
11-gp
12-xjzk

Execute o algoritmo acima e descubra qual o menor conjunto de jogadores que satisfaz às habilidades necessárias

1	2	3	4	5	6	7	8	9	10	11	12

Obs: nem todos os espaços serão necessários



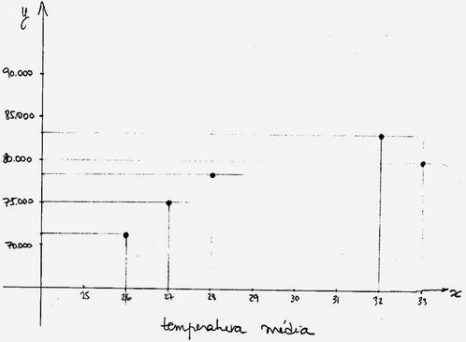
- 1 - /

Seja um problema comum

A Companhia Sonhos gelados produz e comercializa sorvetes. Ela resolveu analisar a temperatura média do verão nos últimos anos e o volume de vendas nessa mesma estação. Uma amostra é apresentada:

temperatura(°C)	32	28	33	27	26
Vendas (mil)	83	78	80	75	71

1. Plote num sistema de eixos os pontos da tabela



2. Determine a equação da reta que melhor se ajusta

As equações normais do problema cujas incógnitas são a e b na equação $y = a + bx$, são

$$na + \left(\sum_{i=1}^n x_i\right)b = \sum_{i=1}^n y_i$$

e

$$\left(\sum_{i=1}^n x_i\right)a + \left(\sum_{i=1}^n x_i^2\right)b = \sum_{i=1}^n x_i y_i$$

Neste caso, tem-se:

n	$=$	5
$\sum_{i=1}^n x_i$	$=$	146
$\sum_{i=1}^n y_i$	$=$	387
$\sum_{i=1}^n x_i^2$	$=$	4302
$\sum_{i=1}^n x_i y_i$	$=$	11351

E fica o sistema

$$\begin{aligned} 5a + 146b &= 387 \\ 146a + 4302b &= 11351 \end{aligned}$$

Da primeira equação fica $a = 77,4 - 29.2b$.
Levando este valor na segunda equação, fica

$$38,8b = 50,6$$

ou

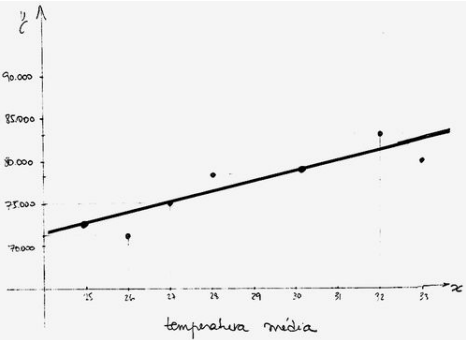
$$b = 1,30$$

. Retornando à primeira equação para obter o valor de a , tem-se $a = 39,44$.
Retornando às equações normais, fica:

$$y = 39,44 + 1,30x$$

3. Esboce a reta obtida

Para tanto, escolheram-se 2 pontos de abcissas $x = 25$ e $x = 30$. As ordenadas correspondentes são $y = 71,94$ e $y = 78,44$ respectivamente. Colocando estes dois novos pontos no gráfico, ficou:



Solução A teoria que se propõe a resolver este problema é o chamado Método dos mínimos quadrados. No caso em particular, busca-se uma reta, de fórmula $y = a + bx$, onde a e b são os parâmetros a determinar. Busca-se minimizar a distância de cada ponto (x_i, y_i) da tabela dada a cada ponto $(X_i, a + bx_i)$ da reta.

Construindo uma tabela com os dados do problema e nomeando-os como segue, fica

$\sum_{i=1}^n n$	$=$	N
$\sum_{i=1}^n x_i$	$=$	J
$\sum_{i=1}^n y_i$	$=$	K
$\sum_{i=1}^n x_i^2$	$=$	L
$\sum_{i=1}^n x_i y_i$	$=$	M

Reescrevendo as equações normais da reta $y = a + bx$, usando os símbolos acima, fica:

$$Na + Jb = K \text{ (eq. 1) e}$$

$$Ja + Lb = M \text{ (eq. 2)}$$

Multiplicando a eq.1 por J e a eq.2 por N, fica:

$$JNa + J^2b = KJ \text{ (eq. 3) e}$$

$$JNa + LNb = NM \text{ (eq. 4)}$$

multiplicando a eq.4 por -1 e somando as duas, fica:

$$(J^2 - LN)b = KJ - NM \text{ (eq.5) ou}$$

$$b = \frac{KJ - NM}{J^2 - LN} \text{ (eq.6) que permite encontrar o valor de } b.$$

Levando este valor à eq.3 e simplificando uma ocorrência de J, fica:

$a = \frac{KJ - J^2b}{JN} = \frac{K - Jb}{N}$ (eq.7) obtém-se o valor de a , o que permite reconstruir a equação da reta procurada. Agora pode-se extrapolar qualquer ponto desejado, bastando colocar o valor de x desejado na reta, e a seguir calcular o valor de y esperado.

Para testar o formulário acima experimente calcular a e b do exercício inicial usando as fórmulas 6 e 7. Veja como funciona.

O algoritmo Dada uma série de valores de x e de y , calcula-se as somas $J = (\sum x)$, $K = (\sum y)$, $L = (\sum x^2)$, $M = (\sum xy)$ e $N = n$ e daí:

```
1: função MINQUA
2: IND,J,K,L,M,N,X,Y,CT:inteiro
3: VX,VY: vetor [1..1000] de inteiro
4: A,B:real
5: leia (X)
6: CT ← 0
7: enquanto X ≠ -1
8:   CT ← CT + 1
9:   VX[CT] ← X
10:  escreva ("digite o par de ",X)
11:  leia (Y)
12:  VY[CT] ← Y
13:  escreva ("digite mais um X ou -1 para encerrar")
14:  leia (X)
15: fim{enquanto}
16: J ← K ← L ← M ← N ← 0
17: para IND=1; IND < CT; IND++
18:   J ← J + VX[IND]
19:   K ← K + VY[IND]
20:   L ← L + VX[IND]^2
21:   M ← M + VX[IND] × VY[IND]
22:   N ← N + 1
23: fim{para}
24: B ← (KJ - NM) / (J^2 - LN)
25: A ← (K - JB) / N
26: leia (X)
27: enquanto X ≠ -1
28:   Y ← A + B × X
29:   escreva (Y)
30:   leia (X)
31: fim{enquanto}
```

Para testar o algoritmo implementado, faça:

x = 26 31 33 36 38 41 42 44
y = 40 41 45 49 51 55 56 58
a = 9.249642005 e b = 1.103102625
para x = 45, y = 58.88 e para x = 50, y = 64.40

Para você fazer

1. No exercício a seguir, considere valores de x=

33 34 36 38 40 44 46 50

associados aos seguintes valores de y=

73 73 78 83 85 100 102 105

Calcule os parâmetros a e b , e depois infira os valores de y para dois valores de x dados (x_1 e x_2)

Para calcular a resposta, arredonde para 2 casas decimais

y_1	y_2

2. Em um país distante chamado LISARB, há muitos problemas econômicos. Em particular, há uma inflação persistente que APARENTEMENTE depende entre outras coisas, da emissão de moeda pelo Banco Central daquele país. Economistas que apoiam esta visão levantaram a inflação e a emissão de moeda nos últimos 10 anos, obtendo as duas séries históricas, a seguir:

EMISSION ANUAL DE MOEDA NOS ULTIMOS 10 ANOS (em bilhoes de dólares)									
206	222	253	203	331	268	683	581	587	213

INFLAÇÃO ANUAL NOS ULTIMOS 10 ANOS									
6.0	6.0	5.5	6.7	7.2	5.9	8.8	6.7	5.1	6.7

Para o ano próximo, o governo pretende efetuar uma emissão de

704

bilhões de dólares. Segundo este modelo econométrico, baseado no método dos mínimos quadrados, qual é a inflação esperada ?

inflação



Regressão múltipla

Quando o passado permite prever o futuro, há um método estatístico denominado Regressão Linear que busca o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados observados.

Em estatística, regressão é uma técnica que permite explorar e inferir a relação de uma variável dependente (variável de resposta) com variáveis independentes específicas (variáveis explicatórias). A análise da regressão pode ser usada como um método descritivo da análise de dados (como, por exemplo, o ajustamento de curvas) sem serem necessárias quaisquer suposições acerca dos processos que permitiram gerar os dados. Regressão designa também uma equação matemática que descreva a relação entre duas ou mais variáveis

Quando há apenas 2 variáveis em análise diz-se "regressão simples"e quando há mais do que 2 variáveis envolvidas, diz-se "regressão múltipla". Na fórmula original tem-se

y = Xb + ε

, que deve ser lido como o parametro matricial b é o que ajusta os y e os x observados.

A fórmula de cálculo de b é b = (X'X)⁻¹X'y, onde X' é a matriz transposta de X é X⁻¹ é a matriz inversa de X.

Acompanhe o exemplo

Suponhamos querer estudar o comportamento do consumo de bens em uma determinada família, sujeita a dois parâmetros:

- renda e
- taxa de juros

Dos estudos de econometria, sabe-se que o consumo aumenta em função da renda e diminui em função da taxa de juros.

O trabalho começa pelo levantamento de dados prévios e obteve-se a seguinte tabela:

medida	consumo y	renda x ₁	tx juros x ₂
1	122	139	11,5%
2	114	126	12,0%
3	86	90	10,5%
4	134	144	9,0%
5	146	163	10,0%
6	107	136	12,0%
7	68	61	10,5%
8	117	62	8,0%
9	71	41	10,0%
10	98	120	11,5%

A matriz X do nosso problema é:

1	139	0.115
1	126	0.12
1	90	0.105
1	144	0.09
1	163	0.1
1	136	0.12
1	61	0.105
1	62	0.08
1	41	0.1
1	120	0.115

Note que a primeira coluna sempre é formada de unidades e as demais colunas são as variáveis em estudo (no caso: renda e taxa de juros).

Tomando a matriz transposta fica-se com

1	1	1	1	1	1	1	1	1	1
139	126	90	144	163	136	61	62	41	120
0.11	0.12	0.10	0.09	0.1	0.12	0.10	0.08	0.1	0.11

Multiplicando uma pela outra, tem-se

10	1082	1.05
1082	132744	115.4
1.05	115.4	0.1118

E a matriz inversa desta é

7.225436859 0.00095983853 -68.85039418
0.00095983853 0.00007350542 -0.084886905
-68.85039418 -0.08488690598 743.1919753

Agora, multiplicando a inversa por X' fica:

-5	-.91	.08	1.16	.49	-.90	.05	1.7	.3	-.5
.0	.00	-.00	.00	.00	.0	-.00	-.0	-.0	.0
4.8	9.63	1.54	-14.1	-8.36	8.7	4.0	-14.6	1.9	6.4

Agora multiplicando esta matriz pela matriz coluna de y (consumo) que é (devidamente horizontalizada para economizar papel)

122 114 86 134 146 107 68 117 71 98

Fica

148.5220194
0.6135933813
-1034.40784

Interpretação

Neste problema, o consumo é

Consumo =

148,52 + 0.6135 × Renda - 1.034.41 × TxJuros

Ou seja, tirando a parte do Consumo que não é influenciada pela Taxa de Juros, o incremento de \$ 1 na Renda causa um incremento esperado de \$ 0,6136 no Consumo; além disso, o incremento de 1 ponto percentual (0,01) na Taxa de Juros causa um decréscimo esperado de \$ 10,3441 no Consumo.

Para você fazer

1. Considere o mesmo problema com os seguintes valores obtidos

medida	consumo y	renda x ₁	tx juros x ₂
1	94	128	.1130
2	86	109	.1130
3	97	124	.1130
4	109	122	.0930
5	110	122	.0930
6	93	123	.0970
7	101	118	.1150
8	100	122	.1000
9	100	125	.1120
10	102	129	.0960

Ache os parâmetros de renda e taxa de juros para o problema proposto.

t. fixo	renda	tx de juros

2. Mais um problema, agora envolvendo um outro modelo econométrico (obviamente FICTICIO) no qual estão relacionadas 3 variáveis independentes e uma dependente. Não se esqueça que a matemática lava as mãos sobre essas definições. É coisa do planejador. São elas:

t	dep.	consumo de whiskie importado (em caixas)
x	indep.	venda de propriedades em Caiobá (em um)
y	indep.	número de falências (sentido oposto, em um)
z	indep.	número de BMW licenciadas (em um)

O objetivo então é achar

t = a + b.x + c.y + d.z

No último ano foram levantados os valores de x, y e z e obteve-se

medida	t	x	y	z
1	-1112078	815	656	-37
2	-288140	86	623	449
3	-725589	19	717	462
4	-2416208	-35	992	342
5	2939912	471	238	677
6	-456408	701	912	393
7	-2425613	827	989	-48
8	-2187865	176	664	8
9	714752	661	448	267
10	3355388	746	43	476

Usando a teoria acima, você deve achar os valores de a, b, c e d.

a	b	c	d

3. O problema agora é que você é o dono do maior restaurante de Santa Felicidade. Precisa prever a quantidade de refeições a serem servidas no próximo Dia das Mães. Depois de muito estudar, você contratou um engenheiro que levantou dados dos últimos 6 anos e elaborou o seguinte modelo econométrico:

a	indep	número de refeições a serem servidas no próximo dia das mães
b	dep	número de casamentos em Curitiba no ano anterior
c	dep	renda média das famílias no Paraná no ano anterior, em reais
d	dep	frangos abatidos na última semana do mês de abril no Paraná
e	dep	veículos emplacadas na ciretran de S. Felicidade no ano anterior

A fórmula do modelo é:

a = α + βb + γc + δd + ε

Os dados:

	a	b	c	d	e
1	3898	15318	13953	169331	7581
2	2366	15382	6852	90308	6770
3	2074	15989	11493	80222	4791
4	3925	17355	6300	165122	9526
5	2520	18003	8129	108352	4126
6	2655	19134	13612	100243	7307

Achar os valores de α, β, γ, δ e ε.

α	β	γ	δ	ε

Observação importante: No Python, ao gerar as matrizes, não esquecer de pedir campos float, porque senão ele pode resolver usando inteiros e vai achar respostas erradas...



Anagramas

Anagramas são construções obtidas combinando as letras de uma dada palavra.
Por exemplo, os anagramas da palavra GATA são: AAGT, GAAT, TAGA, e assim por diante.
Os anagramas são obtidos permutando as letras que compõe a palavra original. O número de anagramas cresce muito rápido, já que o seu número é igual à permutação de n , onde n é a quantidade de letras que existem na palavra.

Acompanhe na tabela:

n	quantidade de anagramas	exemplo
1	1	
2	2	
3	6	***
4	24	
5	120	
6	720	
...	...	
n	$n!$	

*** = por exemplo os anagramas da palavra **ATO** são: AOT, ATO, TAO, TOA, OTA, OAT.
Um possível algoritmo recursivo para obter todos os anagramas de uma dada palavra é a seguir mostrado

```
1: função ANA (inteiro ONDE)
2: inteiro J
3: se ONDE ≠ 1
4:   J ← 1
5:   enquanto J ≤ ONDE
6:     ANA (ONDE - 1)
7:     se ONDE = 2
8:       imprima CONT, PALAVRA
9:       CONT++
10:    fim{se}
11:    RODA ONDE
12:    J++
13:  fim{enquanto}
14: fim{se}
15: fim{função}
```

A função acima, usa esta auxiliar:

```
1: função RODA (inteiro ON)
2: inteiro P, K
3: caracter AUX
4: P ← 1 + tamanho(PALAVRA) - ON
5: K ← P + 1
6: AUX ← PALAVRA[P]
7: enquanto K ≤ tamanho(PALAVRA)
8:   PALAVRA [K-1] ← AUX
9: fim{enquanto}
10: fim{função}
```

 Para você fazer

1. Considere:
PALAVRA ← 'MAIO'
CONT ← 1
2. Execute a função ANA 4
3. Descubra qual palavra será impressa quando CONT for igual a:

8 = _____.

9 = _____.

11 = _____.

12 = _____.

Inversa

Este é o problema em que uma função direta é conhecida (pode ser calculada), enquanto sua inversa não o é. A recursividade pode ser usada para calcular esta inversa por meio se aproximação sucessiva de maneira simples. O algoritmo começa com o valor exato que se quer encontrar e um chute que quando sofresse a ação da função direta deveria gerar o valor exato. Através de modificações desse chute, pode-se chegar rapidamente ao valor esperado. O algoritmo genérico é

```
1: função ACHAINVERSA (valorreal, valorchutado)
2: AUX ← funçãodireta(valorchutado)
3: se 0.0001 > abs(AUX-valorreal)
4:   devolva valorchutado // é a resposta correta
5: senão
6:   devolva ACHAINVERSA (valorreal, NOVOCHUTE(valorchutado))
7: fim{se}
8: fimfuncao
```

O segredo deste procedimento está em achar um NOVOCHUTE que converja para a solução. (Aqui, cada caso é um caso e merece apreciação especial). Para simplificar o raciocínio, pode-se estipular que a função sempre é chamada com valorreal = valorchutado.

Seja um exemplo: Uma empresa quer dar um presente de R\$ 1.000 a um determinado empregado. Se ela simplesmente adicionar este valor ao salário bruto, o empregado acabará recebendo muito menos, já que os descontos são crescentemente proporcionais aos ganhos. A função direta (dado um bruto achar o líquido) é trivial, mas a inversa não.

Em uma situação hipotética, para um bruto de R\$2000, o líquido é de R\$ 1258. Se o empregador aumentar o bruto para R\$ 3000, o líquido passa a ser de 1723, dando ao empregado apenas (1723-1258=465) 465 reais de presente. A pergunta então é: Quanto deve crescer o bruto para que o líquido seja de 2258 ? (1258+1000).

```
Definindo:
1: funcao ACHABRU (vreal, vchut)
2: Z ← CALLIQ (vchut)
3: se 0.01 > abs(Z-vreal)
4:   devolva vchut
5: senão
6:   devolva ACHABRU (vreal,vchut+(vreal-Z))
7: fim{se}
8: fimfunção
```

Chamando ACHABRU (2258,2258) a função acha o líquido para um bruto de 2258 que é 1368. A diferença (1368-2258) é de 890. A nova chamada é ACHABRU (2258,3148). O líquido de 3148 é 1789. A diferença agora é (1789-2258) é 469. A nova chamada é (2258,3617). O líquido de 3617 é 1998. A diferença diminuiu, agora é de 2258-1998=260. A nova chamada é ACHABRU (2258,3877). O líquido de 3877 é 2113... E assim por diante. O valor bruto buscado é de 4202.25, cujo líquido, agora sabemos, é 2258.

O passeio do cavalo

Uma tarefa comum em programação é a busca da solução geral de um problema. Uma estratégia possível (apenas usando computadores) é testar todas as possibilidades mediante o método da tentativa e erro. O caminho mais comum é se decompor a tarefa em sub-tarefas e em geral estas tarefas são muito bem descritas em termos recursivos. Vejamos um exemplo da técnica. Seja um cavalo disposto em um tabuleiro de xadrez de $n \times n = n^2$ casas. (O tabuleiro convencional 8×8 é um caso particular deste mais geral). O cavalo é inicialmente disposto na posição x_0, y_0 , e usando os movimentos usuais do xadrez, deve-se verificar se há algum trajeto que leve o cavalo a passar em todas as casas do tabuleiro, entrando uma única vez em cada uma delas. Uma maneira que "quebrar"este problema é buscar solucionar a maneira de localizar qual o próximo movimento possível ou retornar pela impossibilidade de qualquer movimento.

Estruturas de Dados Usar-se-á uma matriz h de n linhas por n colunas, e nela:

- $h[x, y] = 0$ Indicar que a célula x, y ainda não foi ocupada.
- $h[x, y] = i$ Indicar que a célula x, y foi ocupada no último movimento, onde $1 \leq i \leq n^2$.

A condição tabuleiro não totalmente preenchido pode ser traduzida por $i < n^2$. Se forem criadas duas variáveis u e v a partir das quais se estabelece as novas coordenadas do cavalo, então o predicado movimento aceitável pode ser expresso como $1 \leq u \leq n \wedge 1 \leq v \leq n$ e também $h_{uv} = 0$ ou seja o local de destino do movimento ainda não tenha sido usado. O registro de um movimento válido é então feito usando a atribuição $h_{uv} \leftarrow i$ e o cancelamento deste registro (em caso de chegar-se a um beco sem saída) é pela atribuição $h_{uv} \leftarrow 0$.

Movimentos

Como se sabe, um cavalo localizado na casa central de um tabuleiro 5×5 pode ocupar 8 casas distintas (isto é, pode fazer 8 movimentos distintos. Veja na figura 1.



Algoritmo

e com isso, o algoritmo fica:

```
1: Algoritmo CAVALO
2: inteiro MATRIZ [n] [n]
3: inteiro DLIN [8]
4: inteiro DCOL [8]
5: DLIN ← -2, -1, 1, 2, 2, 1, -1, -2,
6: DCOL ← 1, 2, 2, 1, -1, -2, -2, -1,
7: lógico função ANDACAVALO (int NUMMOV,
   LCAV, CCAV)
8: inteiro LP, CP
9: inteiro K
10: lógico RESP
11: K ← 1
12: repita
13:   RESP ← FALSO
14:   LP ← LCAV + DLIN [K]
15:   CP ← CCAV + DCOL [K]
16:   se (LP ≥ 1) ∧ (LP ≤ n) ∧ (CP ≥ 1) ∧ (CP ≤ n)
17:     se MATRIZ [LP] [CP] = 0
18:       MATRIZ [LP] [CP] ← NUMMOV
19:       se NUMMOV < (n^2)
20:         RESP ← ANDACAVALO (NUMMOV+1, LP, CP)
21:         se (RESP = FALSO)
22:           MATRIZ [LP] [CP] ← 0
23:         fim{se}
24:       senão
25:         RESP ← VERDADEIRO
26:       fim{se}
27:     fim{se}
28:   fim{se}
29:   K++
30: até (K > 8) ∨ (RESP = VERDADEIRO))
31: devolva (RESP)
32: fim função
33: MATRIZ ← 0
34: MATRIZ [linha inicial] [coluna inicial] ← 1
35: se (ANDACAVALO (2, linha inicial, coluna inicial))
36:   imprima MATRIZ
37: senão
38:   imprima "Não existe caminho possível"
39: fim{se}
40: fim algoritmo
```

Alguns resultados Seja um tabuleiro 5×5 inicializado na posição 2,4. Eis uma possível resposta:

25	16	11	6	19
10	5	18	1	12
17	24	15	20	7
4	9	22	13	2
23	14	3	8	21

Este processamento resultou em 86208 movimentos do cavalo.

Para você fazer

Primeiro considere uma modificação no algoritmo acima. Na linha 19, onde se lê $NUMMOV < n^2$ o que equivaleria a testar o valor de i para 64, utilize o valor 56. Ou seja, o algoritmo acaba quando o número 56 for colocado no tabuleiro (e portanto restarem ainda 8 zeros no mesmo). Esta modificação é feita para que o problema possa ser resolvido "à mão".

Para esta instância que você deve resolver, considere a linha inicial do cavalo como sendo a linha 6 e a coluna inicial como sendo a 7. Para um tabuleiro de 8×8 .

Você deve resolver o problema e depois transcrever aqui, o conteúdo da diagonal principal do tabuleiro. Se algum campo não estiver preenchido, responda zero.

 - 1 - /

¹A diagonal que vai de 1,1 até n,n

3 funções recursivas

Máximo Divisor Comum

Este algoritmo recursivo é devido a Euclides

```
1: inteiro MDC(inteiro a, b) {obrigatoriamente:  $a \geq b$ }
2: se  $b = 0$  então
3:   devolva  $a$ 
4: senão
5:   devolva MDC( $b, (a \bmod b)$ )
6: fim se
7: fim {função}
```

Suponha que far-se-á a chamada ao algoritmo acima, como segue:

MDC(1200, 1119)

0 que aconteceria ?
 Chamando com 1200 1119
 Chamando com 1119 81
 Chamando com 81 66
 Chamando com 66 15
 Chamando com 15 6
 Chamando com 6 3
 Chamando com 3 0
 Resposta é 3

Potência recursiva

Seja o seguinte algoritmo que calcula x^y

```
1: inteiro função POTE (inteiro x, y)
2: inteiro t
3: se  $y = 0$  então
4:   devolva 1
5: senão
6:    $t \leftarrow \text{POTE}(x, \lfloor (y \div 2) \rfloor)$ 
7:   se  $0 = (y \bmod 2)$  então
8:     devolva  $t \times t$ 
9:   senão
10:    devolva  $x \times t \times t$ 
11:   fim se
12: fim se
13: end {função}
```

Suponha que far-se-á a chamada ao algoritmo acima, como segue:

POTE(2, 10)

0 que aconteceria ?
 Entrando com 2 10
 Entrando com 2 5
 Entrando com 2 2
 Entrando com 2 1
 Entrando com 2 0
 A resposta é 1024

A pergunta que cabe aqui, é qual a vantagem do algoritmo recursivo acima sobre o algoritmo convencional de potência:

```
1: inteiro função POTECONV (inteiro x, y)
2: inteiro RESP, I
3: RESP  $\leftarrow 1$ 
4: para I de 1 até Y faça
5:   RESP  $\leftarrow \text{RESP} \times x$ 
6: fim para
7: devolva RESP
8: fim {função}
```

Perceba que a função POT NCIA RECURSIVA vai dividindo sucessivamente o expoente por 2. Esse comportamento caracteriza a curva da função logarítmica, pelo qual pode-se dizer que a complexidade dessa função é $O(\log y)$.

Já a função POT NCIA CONVENCIONAL executa um laço de programação que tem exatamente o tamanho do expoente. Por isso, pode-se afirmar que a complexidade da função é $O(y)$.

Um pouco de LISP

Não se pode deixar de lado o LISP quando se fala em recursividade. LISP respira recursividade. Para interpretar a função abaixo, são necessárias algumas noções da linguagem.

defun define uma função do usuário. Contém os seguintes parâmetros: defun, o nome da função, a lista de parâmetros e várias listas de comandos.

cond contém pares de (condição)(ação). Se a condição é *t* executa-se a ação e desvia-se para após o cond. Se ela é *nil* (falsa), desvia-se para a próxima condição.

null pergunta-se se a lista é nula (vazia)

eql retorna *t* se há igualdade e *nil* senão

car(lista) devolve o primeiro elemento da lista

cdr(lista) devolve a lista que sobra quando é excluído o *car* da lista

+ como todas as funções lisp, aplica o *car* da lista sobre o *cdr* da lista. Então (+ 1 2 3) é 6.

format comando de impressão. Exige um *t*, seguido de um string de controle e depois os parâmetros a listar. No string, tem-se *S*: um objeto LISP e *%*: CR+LF

A função que deve ser estudada é

```
(defun quizumba (elemento lista)
  (format t "~S ~%" lista)
  (cond
    ((null lista) 9999)
    ((eql elemento (car lista)) 1)
    (t (+ 1 (quizumba elemento (cdr lista))))))
)
```

Note que cada vez que a função é chamada recursivamente, ela imprime a lista. O que este exercício pede, é que você informe o que é impresso na TERCEIRA chamada da função quizumba.

A pergunta é: “o que faz a função quizumba”? A resposta é uma função que recebe um elemento e uma lista e devolve a posição do elemento na lista. Note que se o elemento citado não pertence à lista, a resposta é ≥ 9999 .

Se quiser mais detalhes de LISP, consulte o livro LIVROIA, de Pedro Kantek, disponível no lugar de sempre. Se quiser baixar o interpretador XLISP para brincar (ou resolver este exercício), vá em <http://almy.us/xlisp.html>. Trata-se de um programa simples, que não exige instalação, nem altera o Registro, basta copiá-lo.

Para interpretar funções LISP, abra um editor ASCII qualquer, de preferência que faça “pareamento” de parênteses – uma sugestão é o scintilla (www.scintilla.org) e digite a função acima. Salve-a com um nome qualquer, com a terminação LSP, no mesmo diretório onde guardou o programa XLWIN32.EXE.

Feito isto, já com XLISP carregado, execute (load 'nome), onde nome é o nome do arquivo LSP que você deu quando o salvou acima.

Finalmente, execute a função definida ou outra qualquer escrevendo (quizumba 'a '(1 2 3 a b c d)). Note o correto pareamento de parênteses. Não esqueça que LISP pode ser entendido como *Lot of Idiot and Stupid Parenthesis*.



Para você fazer

Algoritmo máximo divisor comum Chame o algoritmo mdc com os valores

1494 1235

Responda no quadro próprio, qual o valor da dupla a,b na terceira chamada à função MDC Obs: No exemplo dado acima, seriam os valores 81,66

Algoritmo potência recursiva Chame o algoritmo com a dupla de valores

8 12

Responda no quadro próprio, qual o valor da dupla x,y na terceira chamada à função POTE Obs: No exemplo dado acima, seriam os valores 2,2

Função quizumba

(quizumba 'd '(a b c d e f))

Responda no quadro próprio, qual o valor impresso na TERCEIRA chamada à função.



Árvore QUAD (QUADrante)

O surgimento das árvores QUAD no início dos anos 70, trouxe uma estratégia alternativa na tarefa de armazenar, manipular e analisar imagens computacionais. O nome QUAD vem de QUADrante e a idéia básica é dividir uma imagem em quadrantes de maneira recursiva.

Uma árvore pode ser representada através de uma matriz de cursores. Ganha-se a vantagem do acesso encadeado, sem perder a garantia de que todos os índices (cursores) estarão juntos em uma estrutura sequencial. O preço a pagar por esta facilidade é uma pequena queda na performance.

Combinaremos que a raiz da árvore sempre estará na posição 1 da tabela de cursores, e o endereço de cada filho estará dado pela linha da tabela onde este filho está. Para separar conteúdos de endereços dentro da árvore, estabelecer-se-á: conteúdos=valores negativos e endereços (cursores)=valores positivos.

Dada uma imagem quadrada, a mesma é dividida em quatro quadrantes. Ao mesmo tempo, é criado um nodo contendo 4 filhos na árvore QUAD. Se um determinado quadrante (por exemplo o Q1) é todo ele formado por pixels de mesma cor, esta cor é armazenada no primeiro filho do nodo.

Se outro quadrante (por exemplo o Q3) está formado por pixels de duas cores, o terceiro filho recebe o endereço de outro nodo no qual este quadrante vai ser sub-dividido em quatro novamente.

Acompanhe no exemplo: Seja a imagem
16 x 16 pixels (.=branco; x=preto)

```

      1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
- - - - -
1- . . . . .
2- . . . . .
3- . . . . .
4- . . . . .
5- . . . . .
6- . . . . .
7- . . . . .
8- . . . . . x x x x .
9- . . . . . x x x x .
10- . . . . . x x x x .
11- x x x x . . . . .
12- x x x x . . . . .
13- x x x x . . . . .
14- x x x x . . . . .
15- . . . . .
16- . . . . .

```

Esta imagem geraria a seguinte
árvore QUAD:

```

1- . . 2 7 16
2- . . 3 5
3- . . . 4
4- . . x x
5- . . 6 .
6- . . x x
7- 8 10 12 14
8- . . 9 x
9- . x . x
10- . . 11 .
11- x . x .
12- 13 x . .
13- . x . x
14- 15 . .
15- x . x .
16- 17 18 20 .
17- . x x x
18- x . 19 .
19- x x . .
20- x x . .

```

Seja agora a imagem de um gato

```

      1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
- - - - -
1- . . . . .
2- . . . . .
3- . . . x . . . . .
4- . x x x . . . . .
5- . x x x x . . . . .
6- . . . x x x . . . . .
7- . . . x x x x . . . . .
8- . . . . x x x x x . . . . .
9- . . . . x x x x x . . . . .
10- . . . . x x x x x . . . . .
11- . . . . x x x x x . . . . .
12- x x . . x x . x x x x x x x .
13- . . . . .
14- . . . . .
15- . . . . .
16- . . . . .

```

```

1-2 12 17 25
2-3 5 7 9
3- . . 4
4- . x x
5- . 6 .
6-x . x .
7- 8 .
8-x x .
9-x 10 11 x
10- . x x
11- . x .
12- . 13 15
13- . x 14
14- . x .
15- . . 16
16- . x x
17-18 20 .
18- . 19 .
19- . x x
20-21 22 23 24
21- . x . x
22- . x .
23- . x x x
24- . . x
25-x 26 .
26- . 27 28 29
27- . x . x
28- . x x
29-x x x .

```

A função que cria a árvore QUAD pode ser assim escrita

```

1: função CRIAQUAD (inteiro NUMERO, array IMA)
2: se linhas(IMA) ≤ 2 então
3:   AQUAD[NUMERO][1] ← IMA[1][1]
4:   AQUAD[NUMERO][2] ← IMA[1][2]
5:   AQUAD[NUMERO][3] ← IMA[2][1]
6:   AQUAD[NUMERO][4] ← IMA[2][2]
7: senão
8:   Q1 ← quadrante(1,IMA)
9:   Q2 ← quadrante(2,IMA)
10:  Q3 ← quadrante(3,IMA)
11:  Q4 ← quadrante(4,IMA)
12:  se tudoigual(Q1) então
13:    AQUAD[NUMERO;1] ← Q1[1][1]
14:  senão
15:    PROX++
16:    AQUAD[NUMERO][1] ← PROX
17:    CRIAQUAD (PROX, Q1)
18:  fim se
19:  se tudoigual(Q2) então

```

```

20:   AQUAD[NUMERO][2] ← Q2[1][1]
21: senão
22:   PROX++
23:   AQUAD[NUMERO][2] ← PROX
24:   CRIAQUAD (PROX, Q2)
25: fim se
26: ... {repete para Q3 e Q4}
27: fim se
28: fim função

```

Para chamar, deve-se criar AQUAD como matriz de 64 por 4 contendo zeros, colocar 1 em PROX, e chamar CRIAQUAD(1, IMAGEM-A-COMPACTAR).

Para reconstituir o desenho, deve-se criar a matriz VOLTA contendo 16x16 brancos, a matriz global AQUAD (que contém a árvore quad) e a variável ONDE, que contém originalmente

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

A função retiraquadrante recebe a matriz acima e dela retira as linhas e as colunas do quadrante pedido. Por exemplo, para retirar o quadrante 1, ela faz linhas=1º metade de cima; colunas=1º metade de baixo. Para quadrante2, linhas=1º metade de cima; colunas=2º metade de baixo. Para quadrante3, linhas=2º metade de cima, colunas=1º metade de baixo, e assim por diante.

A função que lê a árvore e cria o desenho é a seguinte:

```

1: função DESENHA (inteiro LINHA, matriz ONDE)
2: inteiro xx ← AQUAD[LINHA]
3: se xx[1] = -1 OU xx[1] = -2 então
4:   VOLTA[retiraquadrante (ONDE, 1)] ← (branco,preto)[abs (xx[1])]
5: senão
6:   DESENHA (xx[1], retiraquadrante (ONDE,1))
7: fim se
8: se xx[2] = -1 OU xx[2] = -2 então
9:   VOLTA[retiraquadrante (ONDE, 2)] ← (branco,preto)[abs (xx[2])]
10: senão
11:   DESENHA (xx[2], retiraquadrante (ONDE,2))
12: fim se
13: ... {repete para xx[3] e xx[4] }
14: fim função

```

Para você fazer agora

1. Suponha a imagem

```

      1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
- - - - -
1- . . . . .
2- . . . . .
3- . . . . .
4- . . . . .
5- . . . . .
6- . . . . .
7- . . . x x x x . . . x x x .
8- . . . x x x x . . . x x x .
9- . . . x x x x . . . x x x .
10- . . . x x x x . . . x x x .
11- . . . . .
12- . . . . . x x x x .
13- . . . . . x x x x .
14- . . . . . x x x x .
15- . . . . . x x x x .
16- . . . . .

```

Escreva o conteúdo da linha 14 da árvore quad criada pelo algoritmo acima (obs: represente o • por -1 e x por -2).

======

2. Suponha a árvore quad

```

1- 2 4 9 11 12- . 13 . 14
2- . . . 3 13- . x . x
3- . . x x 14- . . x x
4- . . 5 7 15- x 16 17 .
5- . . . 6 16- x . x .
6- . x . x 17- . . x x
7- . . x 8 18- . x . 19
8- x . x . 19- x x . .
9- . 10 . . 20- x . 21 .
10- x x . . 21- x x . .
11- 12 15 18 20

```

Responda qual o conteúdo dos pixels selecionados. O • é -1 e x é -2.

IMAGEM[13][14] = =====

IMAGEM[8][1] = =====



Casamento estável

Seja agora o problema de relacionar 2 conjuntos de elementos segundo um determinado critério. Este problema é conhecido na literatura como o casamento estável, uma vez que se usa da analogia com o casamento humano para apresentá-lo.

Como o algoritmo é complexo, antes de apresentá-lo, vamos ver um exemplo. Seja um grupo de 5 moças (Luiza, Joana, Maria, Josefa e Walquíria) e outro conjunto de rapazes (Simão, João, Alfredo, Ernesto e Sebastião). Na nossa analogia, os homens do grupo querem se casar com uma (e só uma) das mulheres e as mulheres com um (e um só) dos homens.

homem	Lista de mulheres com que ele gostaria de casar
Simão	Joana, Josefa, Walquíria, Luiza e Maria
João	Maria, Joana, Luiza, Josefa e Walquíria
Alfredo	Walquíria, Joana, Luiza, Maria e Josefa
Ernesto	Maria, Joana, Josefa, Walquíria e Luiza
Sebastião	Luiza, Joana, Maria, Walquíria e Josefa

Identicamente, foi pedido às mulheres que preparem lista similar. O resultado foi a lista de preferência das mulheres:

mulher	Lista de homens com que ela gostaria de casar
Luiza	João, Simão, Sebastião, Alfredo e Ernesto
Joana	Sebastião, João, Ernesto, Alfredo e Simão
Maria	Simão, Ernesto, João, Alfredo e Sebastião
Josefa	Simão, Sebastião, Alfredo, Ernesto e João
Walquíria	João, Alfredo, Ernesto, Sebastião e Simão

Note-se que, por exemplo João e Ernesto querem se casar com a Maria, enquanto Maria e Josefa querem se casar com Simão. É impossível contentar às 10 pessoas. Alguém terá que ceder. O algoritmo que veremos a seguir, verifica todos os casamentos possíveis apresentando os casamentos estáveis, bem como seu índice de perda para cada um dos casos. O índice de perda é calculado somando-se uma unidade a cada preferência descartada, incluindo a preferência aceita. Ou seja, se num dado casamento, 3 homens conseguiram sua primeira escolha, um deles só conseguiu a segunda e o último só a terceira, a perda dos homens será 8 (os três primeiros perdem 1, o quarto perde 2 e o quinto perde 3, logo $1+1+1+2+3=8$). Nesse mesmo casamento, calcula-se também a perda das mulheres que geralmente é diferente da dos homens. Por hipótese, se aqui, apenas duas tivessem conseguido sua primeira opção, duas a segunda e a última apenas a terceira opção, a perda das mulheres seria de $1+1+2+2+3=9$.

O casamento, neste contexto é a formulação de k pares, com um elemento tirado de cada conjunto, onde k é o tamanho de cada um deles. Ele é dito instável quando para pelo menos um dos casais formados vale a regra: H_1 está casado com M_1 , e M_2 está casada com H_2 . Entretanto, na lista de preferências de H_1 , M_2 aparece antes do que M_1 . Identicamente, na lista de M_2 , H_1 aparece antes do que H_2 . Se não houver nenhum casal formado para o qual a regra acima seja válida, o casamento é dito estável.

O algoritmo será chamado para o primeiro homem da lista. Este será casado com todas as mulheres (na ordem de sua preferência), preservando-se o primeiro resultado que for estável. Feito isso, passa-se ao segundo homem (eis aqui a recursividade), e assim sucessivamente. Quando todos os homens estiverem casados, o resultado será impresso (com as correspondentes perdas), e se retorna na busca da próxima mulher que continuar tornando estável a união. Quando todas as possibilidades forem exauridas, o algoritmo acaba.

O resultado obtido para este conjunto de dados é

sol.	x ₁	x ₂	x ₃	x ₄	x ₅	p. masc	p. fem
1	4	2	5	3	1	7	10
2	4	1	5	3	2	9	7

A melhor solução para os homens (perda mas=7) é formada por Simão-Josefa (p=2); João-Joana (P=2); Alfredo-Walquíria (P=1), Ernesto-Maria (P=1), Sebastião-Luiza (P=1). Esta mesma solução apresenta as seguintes perdas para as mulheres: Josefa-Simão (P=1); Joana-João (P=2); Walquíria-Alfredo (P=2); Maria-Ernesto (P=2), Sebastião-Luiza (P=3).

A melhor solução para as mulheres (perda fem=7) é formada por Josefa-Simão (P=1); Luiza-João (P=1); Walquíria-Alfredo (P=2); Maria-Ernesto (P=2) e Joana-Sebastião (P=1). Essa mesma solução apresenta as seguintes perdas para os homens: Simão-Josefa (P=2); João-Luiza (P=3); Alfredo-Walquíria (P=1); Ernesto-Maria (P=1) e Sebastião-Joana (P=2).

Dito isso, segue o algoritmo

```

1: função IMPRIME ()
2: inteiro I, PERDAHOM, PERDAMUL
3: PERDAHOM ← PERDAMUL ← 0
4: para (I = 0; I ≤ n - 1; I++) faça
5:   imprima MULHER[I]+1
6:   PERDAHOM ← PERDAHOM + CLASHOM [I] [MULHER[I]] + 1
7:   PERDAMUL ← PERDAMUL + CLASMUL [MULHER[I]] [I] + 1
8: fim para
9: imprima PERDAHOM, PERDAMUL
10: fim função
1: função TENTATIVA (HOM)
2: inteiro HOM
3: inteiro K, MUL
4: K ← 0
5: enquanto K < n faça
6:   MUL ← PREFHOM [HOM] [K]
7:   se SOLTEIRO [MUL] ∧ ESTAVEL (HOM, MUL, K) então

```

```

8:   MULHER [HOM] ← MUL
9:   HOMEM [MUL] ← HOM
10:  SOLTEIRO [MUL] ← .FALSO.
11:  se HOM < n-1 então
12:    TENTATIVA (HOM+1)
13:  senão
14:    IMPRIME()
15:  fim se
16:  SOLTEIRO[MUL] ← .V.
17: fim se
18: K++
19: fim enquanto
1: lógico função ESTAVEL (HOM, MUL, LIMITE)
2: inteiro HOM, MUL, LIMITE
3: inteiro CANDHOM, CANDMUL, K, ATEONDE
4: lógico RESP
5: RESP ← .V.
6: enquanto ((K < LIMITE) ∧ RESP) faça
7:   CANDMUL ← PREFHOM [HOM] [K]
8:   K++
9:   se ( SOLTEIRO[CANDMUL]) então
10:    RESP ← CLASMUL[CANDMUL][HOM] > CLAS-
        MUL[CANDMUL][HOMEM[CANDMUL]]
11:   fim se
12: fim enquanto
13: K ← 0
14: ATEONDE ← CLASMUL [MUL] [HOM]
15: enquanto ((K < ATEONDE) ∧ RESP) faça
16:   CANDHOM ← PREFMUL [MUL] [K]
17:   K++
18:   se (CANDHOM < HOM) então
19:    RESP ← CLASHOM[CANDHOM][MUL] >
        CLASHOM[CANDHOM][MULHER[CANDHOM]]
20:   fim se
21: fim enquanto
22: devolva RESP
1: Algoritmo CASAMENTO
2: inteiro PREFHOM [n] [n]
3: inteiro PREFMUL [n] [n]
4: inteiro CLASHOM [n] [n]
5: inteiro CLASMUL [n] [n]
6: inteiro HOMEM [n]
7: inteiro MULHER [n]
8: lógico SOLTEIRO [n]
9: inteiro I,J
10: SOLTEIRO ← .VERDADEIRO.
11: leia PREFHOM, PREFMUL
12: PREFHOM ← PREFHOM - 1
13: PREFMUL ← PREFMUL - 1
14: para i de 0 até n-1 faça
15:   para j de 0 até n-1 faça
16:     CLASHOM[I] [PREFHOM[I] [J]] ← J
17:     CLASMUL[I] [PREFMUL[I] [J]] ← J
18:   fim para
19: fim para
20: TENTATIVA (0)

```

Para você fazer

Seja a preferência dos homens

2	1	7	4	6	3	8	9	10	5
5	10	3	4	8	6	7	2	9	1
7	6	8	10	5	1	9	4	2	3
7	9	10	8	2	5	6	4	1	3
4	1	3	6	7	8	9	5	2	10
7	3	10	4	9	6	2	1	8	5
6	3	1	8	5	4	10	9	7	2
8	10	1	5	3	2	7	6	4	9
8	1	7	2	5	3	10	9	6	4
4	7	2	8	5	9	6	1	3	10

Seja a preferência das mulheres

8	3	4	1	7	6	9	5	10	2
4	5	2	7	10	3	1	8	9	6
4	8	1	5	2	7	9	6	3	10
8	4	7	3	9	5	1	2	6	10
1	5	4	9	2	6	3	10	8	7
5	2	4	8	10	6	7	1	3	9
6	7	10	2	5	3	8	9	4	1
5	4	1	6	7	10	9	8	3	2
7	5	9	6	8	2	10	3	1	4
4	3	1	5	7	8	6	2	10	9

Implemente os algoritmos acima (não esqueça de subtrair 1 de cada elemento das matrizes acima, por causa da origem dos índices = 0) e ache o menor custo para os homens e o menor custo para as mulheres.

menor custo ho- mens	menor custo mulheres

SUDOKU para computadores O Sudoku é um quebra-cabeça moderno (parece ter sido inventado por um arquiteto aposentado de 74 anos, Howard Garns em 1979) e começou a se tornar popular no Japão, com o nome de *suuji wa dokushin ni kagiru*, que significa “os dígitos devem permanecer únicos”. Sua popularidade naquele país se explica pela dificuldade do idioma japonês para fazer palavras cruzadas. No Ocidente, o jogo só começou a se popularizar em 2004, quando Wayne Gould - um juiz aposentado de Hong Kong, que também era fã de puzzles e programador de computador - viajou a Londres para convencer os editores do The Times a publicar o sudoku. Gould havia criado um programa de computador que gerava jogos de sudoku com vários níveis de dificuldade e não estava cobrando nada por ele. O Times decidiu arriscar e no dia 12 de novembro de 2004 publicou seu primeiro sudoku. No Brasil o sudoku é publicado pela Coquetel (Ediouro) desde o início de 2005 e a Folha de São Paulo e a Gazeta do Povo começaram a publicar um sudoku diariamente. O sudoku é muito popular hoje e parece ter as características de um bom e duradouro quebra cabeça:

- é disponível em diversos graus de dificuldade
- sua regra de solução é muito simples
- a habilidade de resolvê-lo cresce com a prática
- ao resolvê-lo a pessoa vê o progresso em direção à solução

O jogo é apresentado em uma grade 9 × 9, que está dividida em 9 linhas, 9 colunas e 9 blocos (de 3 × 3 cada). Neste texto, estes três elementos (linhas, colunas e blocos) serão chamados genericamente de “grupos”.

A regra fundamental (e única) do SUDOKU é que em cada grupo deve haver um conjunto completo de números 1..9. Note-se que os números são usados apenas por comodidade, nenhuma propriedade aritmética deles é importante. Poder-se-iam usar 9 símbolos de times de futebol, cores, signos do zodíaco etc. Antes de continuar a leitura, tente resolver este sudoku

		4	8					
	9		4	6			7	
	5					6	1	4
2	1		6			5		
5	8		7		9		4	1
		7			8		6	9
3	4	5					9	
	6			3	7		2	
				4	1			

Existe muito material sobre como um ser humano pode resolver um sudoku. Há até um exercício VIVX (o vivxz20) que pede a solução de um sudoku proposto. Entretanto, nesta folha a perspectiva é a da solução de um sudoku através de um programa de computador. Este pode se dar ao luxo de usar a técnica mais simples (embora um bocado trabalhosa) de “tentativa e erro”. Para tanto, tem-se o seguinte algoritmo:

```

1: lógico função SUDOKU (inteiro L, C) {L,C é o endereço do primeiro zero}
2: inteiro K ← 1
3: lógico RESP
4: repita
5:   RESP ← FALSO
6:   se L > 3 então
7:     devolva VERDADEIRO // era RESP...
8:   fim se
9:   se EPOSSIVEL K,L,C então
10:    M[L][C] ← K
11:    se (L ≠ 9) ∨ (C ≠ 9) então
12:      L1, C1 ← OPROXIMO L, C
13:      RESP ← SUDOKU L1, C1
14:      se RESP = FALSO então
15:        M[L1][C1] ← 0
16:      fim se
17:    senão
18:      RESP ← VERDADEIRO
19:    fim se
20:  senão
21:    K++
22:  fim se
23: até (K > 9) ∨ (RESP=VERDADEIRO)
24: devolva RESP
25: fimfunção
    
```

Obs: Para usar este algoritmo na solução de um SUDOKU verdadeiro, somente as linhas 6, 7 e 8.

Usa-se o algoritmo para determinar se um determinado valor K pode ou não ser colocado na posição L,C

```

1: lógico função EPOSSIVEL (inteiro K, L, C)
2: lógico R ← VERDADEIRO
3: inteiro Z
4: para Z de 1 até 9 faça
5:   se M[L][Z] = K então
6:     R ← FALSO
7:   fim se
8: fim para
9: para Z de 1 até 9 faça
10:  se M[Z][C] = K então
11:    R ← FALSO
12:  fim se
13: fim para
14: Q1 ← 1 1; 1 2; 1 3; 2 1; 2 2; 2 3; 3 1; 3 2; 3 3
15: Q2 ← 1 4; 1 5; 1 6; 2 4; 2 5; 2 6; 3 4; 3 5; 3 6
16: ...
17: Q9 ← 7 7; 7 8; 7 9; 8 7; 8 8; 8 9; 9 7; 9 8; 9 9
18: se ((L=1)∨(L=2)∨(L=3))∧((C=1)∨(C=2)∨(C=3)) então
19:   para Z de 1 até 9 faça
20:     LX ← Q1[Z][1]
21:     CX ← Q1[Z][2]
22:     se M[LX][CX] = K então
23:       R ← FALSO
24:     fim se
25:   fim para
26: fim se
27: ... repete o bloco para Q2, Q3, ..., Q9
28: devolva R
29: fimfunção
    
```

Usa-se também a função OPROXIMO

```

1: (inteiro LP, LC) função OPROXIMO (inteiro L, C)
2: C++
3: enquanto L ≤ 9 faça
4:   enquanto C ≤ 9 faça
5:     se M[L][C] = 0 então
6:       devolva L,C
7:     saia
8:   senão
9:     C++
10:   fim se
11: fim enquanto
12: C ← 1
13: L++
14: fim enquanto
15: fimfunção
    
```

O número de soluções de Sudoku para uma grade padrão de 9x9 foi calculado em 2005 por Bertram Felgenhauer como sendo 6.670.903.752.021.072.936.960.

 **Para você fazer**

		1			9			
				6			4	3
	3		5			9		
			4					
		8				5	7	
4					3			9
		2						
9		3		2				5

que será colocado na matriz M, colocando-se 0 (zero) nas posições em branco, aplique a função SUDOKU 1,1 e chame-a sucessiva (e recursivamente) até que as primeiras 3 linhas estejam preenchidas.

Observação importante: O sudoku acima pode possuir muitas respostas formalmente corretas (ou mesmo nenhuma). Por essa razão está-se parando o algoritmo após preencher a terceira linha. Veja as linhas 6, 7 e 8 do algoritmo principal. Logo, não adianta resolver na mão... **Na resposta PODEM aparecer zeros.** Transcreva a primeira linha obtida:

1	2	3	4	5	6	7	8	9



*Para entender a recursividade, você precisa antes,
 ... entender a recursividade (anônimo).*

Programação Literária (ou literata) Programação literária é uma filosofia de programação de computadores baseada na premissa que um programa de computador deve ser escrito tendo a legibilidade humana como principal objetivo, similar a um trabalho de literatura. De acordo com essa filosofia, programadores devem almejar um estilo "literário" em sua programação tanto quanto escritores buscam um estilo inteligível e articulado em sua escrita. Essa filosofia contrasta com a visão mais comum que diz que o objetivo principal ou único do programador é criar código fonte e que a documentação deve ser apenas um objetivo secundário.

Na prática, programação literária é atingida pela combinação da documentação (legível por humanos) e código fonte (legível pela máquina) em um único arquivo fonte, de modo a manter uma correspondência próxima entre a documentação e o código fonte. A ordem e a estrutura desse arquivo são especificamente projetadas para auxiliar a compreensão humana: código e documentação juntos são organizados em ordem lógica e/ou hierárquica (tipicamente de acordo com um esquema que acomode explicações detalhadas e comentários como necessárias). Ao mesmo tempo, a estrutura e o formato dos arquivos fontes acomodam utilitários externos que geram documentação do programa e/ou extraem o código legível por máquina dos mesmos arquivos fontes (para processamento subsequente por compiladores ou interpretadores).

O primeiro ambiente de programação literária publicado foi WEB, (nada a ver com a Wide Worl Web, que só surgiria 10 anos depois) introduzido por Donald Knuth em 1981 para seu sistema de tipografia TeX; ele usava Pascal como sua linguagem de programação subjacente e TeX para tipografia da documentação.

A seguir 3 implementações para você fazer, sugerindo-se que se use a programação literária.

99 garrafas de cerveja Este é um programa bem simples, usado para descrever os princípios da programação recursiva, que são

Autochamada Um programa recursivo sempre chama a si próprio ainda que de maneira indireta.

Caso básico a situação limite da recursão, onde esta ACABA, ou seja aqui não há mais chamadas ao programa.

Universo menor A cada chamada, o universo de trabalho deve ser menor, sempre em direção ao caso básico.

O programa das 99 garrafas de cerveja é chamado com um parâmetro (o número das garrafas) e ele deve imprimir os versos a seguir, começando no número dado e diminuindo 1 garrafa a cada chamada até chegar a zero:

```
99 garrafas de cerveja na mesa
99 garrafas de cerveja
  abra uma
    passe-a adiante
98 garrafa de cerveja na mesa
```

Os casos particulares, são: 2 garrafas (a última frase deve ser 1 GARRAFA... (no singular) e 1 garrafa, quando as 2 primeiras frases também devem ser no singular. Finalmente, quando não houver garrafas, acaba a recursão.

Números de Ackerman Esta é uma função recursiva terrivelmente custosa para ser calculada. Apenas as instâncias absolutamente triviais podem ser objeto de manuseio por parte de um computador.

	vale	se
	$n + 1$	$m = 0$
$A(m, n) =$	$A(m - 1, 1)$	$m > 0 \text{ e } n = 0$
	$A(m - 1, A(m, n - 1))$	$m > 0 \text{ e } n > 0$

O objetivo desta função, é obter o número de Ackerman de um par de operandos. Por exemplo, quanto é o Ackerman de 12 e 15 (i.é: $A(12,15)=?$). Curioso é que nenhum computador consegue calcular $A(12,15)$. É demais...

Números por extenso Uma aplicação sempre presente em qualquer ambiente onde computadores manuseiem dinheiro (todos os ambientes) é a escrita de valores por extenso. Seja no preenchimento de cheques, na elaboração de contratos e em muitos documentos de ordem legal, é sempre necessário converter (por exemplo) 1467 em "mil quatrocentos e sessenta e sete".

A maioria dos ambientes de programação têm este módulo pronto, mas nem sempre em português. As empresas maiores já o tem programado, mas a questão é: "e se tivéssemos que programar esta função ?".

Especificação formal: Seja uma função que receba um número inteiro entre 1 e 999.999.999.999 e devolva uma cadeia de caracteres com o número por extenso.

Estratégia: Vai se tentar achar uma regra básica. Deixam-se de lados as exceções, não porque inexistentes, mas sim para não atrapalhar o processo de construção da solução. Examinemos alguns números ao acaso:

123 cento e vinte e três

123456 cento e vinte e três mil quatrocentos e cinquenta e seis

123456789 cento e vinte e três milhões quatrocentos e cinquenta e seis mil setecentos e oitenta e nove

Salta aos olhos, que existe um procedimento básico que transforma um número de 3 dígitos em um texto. Para tanto, este número deve ser subdividido em unidades, dezenas e centenas.

Se se quiser trabalhar com 2 funções, uma delas (a A) será chamada para gerar o texto a partir da centena+dezena+unidade e outra função (a B) irá "descascando" o número original com chamadas sucessivas a A.

Se quiser dar uma de (mais) bacana, faça uma única função, possivelmente recursiva que faça tudo isso.

Para a função A, que recebe um número menor que 1000 e converte-o em texto, são necessárias 3 tabelas: **T1** com 20 linhas,

```
1=(brancos)
2=um
3=dois
...
20=dezenove
```

T2 com 10 linhas:

```
1=2=(brancos)
3=vinte
4=trinta
...
10=noventa
```

Finalmente **T3** com 10 linhas também:

```
1=(brancos)
2=cento
3=duzentos
...
10=novecentos
```

Alguns exemplos:

```
extenso 123456789
cento e vinte e tres milhoes quatrocentos e cinquenta e seis mil setecentos e
oitenta e nove
extenso 1001898
um milhao um mil oitocentos e noventa e oito
extenso 1111111
um milhao cento e onze mil cento e onze
extenso 3892723897489
DOMAIN ERROR
extenso 3723897489
tres bilhoes setecentos e vinte e tres milhoes oitocentos e noventa e sete mil
quatrocentos e oitenta e nove
```

🔗 Para você fazer

Escreva um programa em Java, Pascal ou VisualG que implemente as 3 funções vistas acima (99 garrafas, Ackerman e extenso de números) e apresente o código fonte e o resultado da execução das 3 funções para o professor.

Prepare-se para defender sua implementação e lembre-se de que haverá uma análise para prevenir a contrafação.

1

Problema do caixeiro viajante - TSP

O problema do caixeiro-viajante (PCV), ou como é conhecido em inglês: travelling salesman problem (TSP), é um problema de otimização que, apesar de parecer modesto é, na realidade, um dos mais estudados, por cientistas, matemáticos e investigadores de logística, genética e produção, entre outros (Applegate et al., cop. 2006, p. 1). O problema pertence à categoria NP-hard (em inglês), que o remete para um campo de complexidade exponencial, isto é, o esforço computacional necessário para a sua resolução cresce exponencialmente com o tamanho do problema. Assim, dado que é difícil, se não impossível, determinar a solução ótima desta classe de problemas, os métodos de resolução passam pelas heurísticas e afins que, do ponto de vista matemático, não asseguram a obtenção de uma solução ótima (Cunha, 2002).

Definição e formulação do problema Em termos leigos, imagine um território de vendas (por exemplo, o Estado do Paraná), que tem diversas cidades polo que devem ser visitadas pelo vendedor. Ele começa e termina na sua base e deve visitar uma e só uma vez cada cidade polo. O critério para a escolha da ordem das cidades é minimizar a distância total percorrida.

Usando uma definição acadêmica:

O problema do caixeiro-viajante consiste na procura de um circuito que possua a menor distância, começando numa qualquer cidade, entre várias, visitando cada cidade precisamente uma vez e regressando à cidade inicial (Nilsson, 1982).

O tamanho do espaço de procura aumenta exponencialmente (fatorialmente) dependendo da quantidade de cidades *n*, o número de cidades, uma vez que existem $(n - 1)!$ circuitos possíveis (a posição inicial é arbitrária, e a ordem do circuito pode ser invertida).

A solução do PCV pode ser de dois tipos: a exata (impraticável em instância médias ou grandes, devido ao comportamento exponencial) e as aproximadas, todas dependentes de alguma heurística.

Começando pelo segundo grupo, há diversos métodos. Estes são procedimentos particulares, o que os torna inflexíveis para a determinação de boas soluções para um outro problema ligeiramente diferente.

Quadro esquemático de soluções aproximadas

- construção de circuitos (alg. guloso)
- árvore de cobertura mínima
- *simulated annealing*
- algoritmo de busca tabu
- redes neurais
- algoritmos genéticos
- *aint colony optimization algorithm*

Solução Exata Embora demorada esta é a solução que vai ser trabalhada neste exercício. É um bom exemplo de algoritmo recursivo (neste caso, claramente a recursão vem em nosso auxílio: este mesmo algoritmo não recursivo teria enorme complicação e dificuldade) e suficiente pequeno para poder rodar em minutos usando um computador pessoal simples.

Um caso real Em 1998 uma equipe de matemáticos encontrou o caminho mais curto para visitar as 13.509 cidades americanas que tinham, naquele ano, mais de 500 habitantes. Foram necessários 3,5 meses de processamento de três multiprocessadores (32 pentium cada) ligados em rede. Aqui, eliminaram-se as rotas obviamente ineficientes logo de cara. O problema: a estratégia só vale para este problema e para estas cidades.

Outro caso real Os algoritmos mostrados nesta folha e rodados em um micro bem lerdinho (um pentium 4 com 2.8GHz, com 632 MB de RAM, mas rodando Linux e neste sob Wmware um Windows XP, e neste rodando APL2 da IBM) e em um micro rápido (2 CPUS de 2.80GHz, 3.3GB de memória e com APL2 sob Windows XP nativo)

qtd de cidades	CPU lerda	CPU rápida
6 cidades	31 milisseg	desprezível
7 cidades	156 milisseg	78 milisseg
8 cidades	1.1 seg	500 milisseg
9 cidades	9 seg	4 seg
10 cidades	81 seg	37 seg
11 cidades	836 seg	367 seg
12 cidades	153 min	67 min
13 cidades	30.6 horas	13.4 horas
14 cidades	16.5 dias	7.2 dias

O problema que você vai resolver

Há uma lista de 6 cidades paranaenses, cada uma com suas coordenadas obtidas a partir de um eixo cartesiano imaginário montado sobre o paralelo 26º SUL (x) e sobre o meridiano 54ºO (y). No cálculo, considerou-se uma projeção plana do território, o que certamente introduzirá pequenas distorções. Não importa, avance ! Seja um exemplo de 6 cidades:

1-CANDIDO ABREU	269 157
2-GEN CARNEIRO	268 -49
3-GUAIRA	-25 213
4-JACAREZINHO	399 315
5-PALMAS	200 -54
6-PARANAGUA	543 51

Supondo que a cidade inicial seja 3 (Guaira), a sequência de cidades a visitar é 3 1 4 6 2 5 3 e a distância é de 1514.57 km

Algoritmos Para construir a matriz de distâncias entre pares de cidades, o algoritmo é

```
1: funcao ACHADIST (COORD: nome,xx,yy[nn]) : matd [nn;nn]
2: i,j : inteiro
3: para i = 1 to nn
4:   para j = 1 to nn
5:     matd[i][j] ← COORD[i;2 3] distancia COORD[j;2 3]
6:   fim{para}
7: fim{para}
```

Note-se que a função **distancia** acima usada é a simples aplicação do Teorema de Pitágoras sobre o mapa. Note-se também que se o TSP deve ser feito sobre outro conceito que não a distância linear, esta função deve ser substituída pela que calcula o método desejado. Este pode ser: distância via estradas, tempo de voo, custo do pedágio, ...

Eis a matriz de distâncias para o exemplo

	Cand Abr	Gen Car	Guaira	Jacarez	Palmas	Pgua
Cand Ab	.000	206.002	299.286	204.607	221.995	293.789
Gen Car	206.002	.000	393.056	386.855	68.184	292.617
Guaira	299.286	393.056	.000	436.096	349.162	590.650
Jacarez	204.607	386.855	436.096	.000	419.240	300.719
Palmas	221.995	68.184	349.162	419.240	.000	358.712
Pgua	293.789	292.617	590.650	300.719	358.712	.000

TSP e o algoritmo guloso Se uma resposta degradada for aceitável, pode-se aplicar o algoritmo guloso neste problema. A estratégia aqui é *das cidades não visitadas, devo escolher a mais próxima*. Em um caso real, obtiveram-se os seguintes valores:

Cidades: 1-Cascavel, 2-Fco Beltrao, 3-Guaira, 4-Guarapuava, 5-Irati, 6-Ortigueira, 7-Paranavai, 8-Ponta Grossa, 9-Telêmaco Borba e 10-Umuarama. Inicial: Ponta Grossa: Solução exata: 1111.97 Km. Sequência: 8 5 4 2 1 3 10 7 6 9 8. Algoritmo Guloso: 1329.56 Km. Sequência: 8 5 4 6 9 7 10 3 1 2 8

🔧 Para você fazer

Suponha a seguinte lista de cidades

1-FOZ IGUACU	-59 55
2-GUARAPUAVA	254 67
3-LONDRINA	291 297
4-MARINGA	211 287
5-PARANAVAI	158 323
6-TELEM BORBA	341 185

Sendo que o início e o fim do ciclo de viagens deve ocorrer na cidade de MARINGA . Após calcular o menor caminho entre todas as cidades, responda:

1. Qual o comprimento em quilômetros do menor caminho ?
2. Qual o comprimento em quilômetros do caminho guloso ?

m.c.	guloso

Note que como a instância deste problema é pequena é perfeitamente possível que os valores (caminho mínimo e caminho guloso) sejam iguais.



Problema do caixeiro viajante - TSP em Python

O problema do caixeiro-viajante (PCV), ou como é conhecido em inglês: travelling salesman problem (TSP), é um problema de otimização que, apesar de parecer modesto é, na realidade, um dos mais estudados, por cientistas, matemáticos e investigadores de logística, genética e produção, entre outros (Applegate et al., cop. 2006, p. 1). O problema pertence à categoria NP-hard (em inglês), que o remete para um campo de complexidade exponencial, isto é, o esforço computacional necessário para a sua resolução cresce exponencialmente com o tamanho do problema. Assim, dado que é difícil, se não impossível, determinar a solução ótima desta classe de problemas, os métodos de resolução passam pelas heurísticas e afins que, do ponto de vista matemático, não asseguram a obtenção de uma solução ótima (Cunha, 2002).

Definição e formulação do problema Em termos leigos, imaginem um território de vendas (por exemplo, o Estado do Paraná), que tem diversas cidades polo que devem ser visitadas pelo vendedor. Ele começa e termina na sua base e deve visitar uma e só uma vez cada cidade polo. O critério para a escolha da ordem das cidades é minimizar a distância percorrida.

Usando uma definição acadêmica:

O problema do caixeiro-viajante consiste na procura de um circuito que possua a menor distância, começando numa qualquer cidade, entre várias, visitando cada cidade precisamente uma vez e regressando à cidade inicial (Nilsson, 1982).

O tamanho do espaço de procura aumenta exponencialmente dependendo da quantidade de cidades *n*, o número de cidades, uma vez que existem $(n-1)!/2$ circuitos possíveis (a posição inicial é arbitrária, e a ordem do circuito pode ser invertida).

A solução do PCV pode ser de dois tipos: a exata (impraticável em instância médias ou grandes, devido ao comportamento exponencial) e as aproximadas, todas dependentes de alguma heurística.

Começando pelo segundo grupo, há diversos métodos. Estes são procedimentos particulares, o que os torna inflexíveis para a determinação de boas soluções para um outro problema ligeiramente diferente.

Quadro esquemático de soluções aproximadas

- construção de circuitos (alg. guloso)
- árvore de cobertura mínima
- *simulated annealing*
- algoritmo de busca tabu
- redes neurais
- algoritmos genéticos
- *aint colony optimization algorithm*

Solução Exata Embora demorada esta é a solução que vai ser trabalhada neste exercício. É um bom exemplo de algoritmo recursivo (neste caso, claramente a recursão vem em nosso auxílio: este mesmo algoritmo não recursivo teria enorme complicação e dificuldade) e suficiente pequeno para poder rodar em minutos usando um computador pessoal simples.

Um problema grande Só para ter uma idéia eis o tamanho do universo de rotas em um problema do caixeiro viajante com 200 cidades: (já dividido por 2)
3943289336823952517761816069660925311475679
8884358663164737126662217972498170167146015
2142005992311952088606069459819415128821395
1213185525309633124764149655567314286353816
5861869849447196122281072583212012701664593
2065613714147426638762121203786951620160628
7027897843301130159520851620311758504293980
8946111139481185194868736000000000000000000
0000000000000000000000000000000000000000

Um caso real Em 1998 uma equipe de matemáticos encontrou o caminho mais curto para visitar as 13.509 cidades americanas que tinham, naquele ano, mais de 500 habitantes. Foram necessários 3,5 meses de processamento de três multiprocessadores (32 pentium cada) ligados em rede. Aqui, eliminaram-se as rotas obviamente ineficientes logo de cara. O problema: a estratégia só vale para este problema e para estas cidades.

O problema que você vai resolver

Há uma lista de 10 cidades paranaenses, cada uma com suas coordenadas obtidas a partir de um eixo cartesiano imaginário montado sobre o paralelo 26° SUL (x) e sobre o meridiano 54°O (y). No cálculo, considerou-se uma projeção plana do território, o que certamente introduzirá pequenas distorções. Não importa, avante ! Seja um exemplo de 10 cidades:

1-CAMPO MOURAO	164 216
2-CURITIBA	467 63
3-FCO BELTRAO	91 -8
4-GUAIRA	-25 213
5-IRATI	338 62
6-LONDRINA	291 297
7-ORTIGUEIRA	310 202
8-PALMAS	200 -54
9-PARANAGUA	543 51
10-PARANAVAI	158 323

Supondo que a cidade inicial seja 2 (Curitiba), a sequência de cidades a visitar é 2 5 8 3 4 1 10 6 7 9 2 e a distância é de 1560.3 km. Como nesta folha está-se trabalhando com Python (cuja origem de numeração é zero) a resposta é 1560.36722088 [1, 4, 7, 2, 3, 0, 9, 5, 6, 8, 1]. Mas o resultado é o mesmo.

Algoritmos Para construir a matriz de distâncias entre pares de cidades, o algoritmo é

```
def criamatcv():
    nomes=["Campo Mourao","Curitiba",
    "Fco Beltrao", "Guaira","Irati",
    "Londrina","Ortigueira","Palmas",
    "Paranagua","Paranavai"]
    locax=[164,467,91,-25,338,291,
    310,200,543,158]
    locay=[216,63,-8,213,62,297,
    202,-54,51,323]
    mat=numpy.zeros((len(nomes),len(nomes)))
    for j in range(len(nomes)):
        for i in range(len(nomes)):
            mat[i][j]=((locax[i]-locax[j])**2)+
            ((locay[i]-locay[j])**2)**0.5
    return mat
```

Note-se que a função distancia acima usada é a simples aplicação do Teorema de Pitágoras sobre o mapa. Note-se também que se o TSP deve ser feito sobre outro conceito que não a distância linear, esta função deve ser substituída pela que calcula o método desejado. Este pode ser: distância via estradas, tempo de voo, custo do pedágio, ...

Montada a matriz de distâncias, é hora de chamar a função TSP, mas antes, são necessárias algumas variáveis globais: MENOR: a menor distância até agora encontrada e inicializada com +∞ e CMENOR: o caminho que dá origem à menor distância até agora encontrada.

```
def cv(lja,lfa,km,mat):
    # print(lja,lfa,str(km))
    | global MENOR
    | global CMENOR
    | if len(lfa)!=0:
    | | ori=lja[len(lja)-1]
    | | for i in range(len(lfa)):
    | | | des=lfa[i]
```

```
| | | alfa=lfa[:]
| | | alfa.remove(des)
| | | alja=lja[:]
| | | alja.append(des)
| | | akm=km+mat[ori][des]
| | | cv(alja,alfa,akm,mat)
| else:
| | km=km+mat[lja[len(lja)-1]][lja[0]]
| | lja.append(lja[0])
| | if km<MENOR:
| | | MENOR=km
| | | CMENOR=lja                      Obs: | para clareza!
```

Para chamar faça (depois de definir as 2 funções acima). Os dados são os do exemplo da folha

```
aa=criamatcv()
MENOR+=9999999
CMENOR=[]
#print(aa)
cv([1],[0,2,3,4,5,6,7,8,9],0,aa)
print(MENOR,CMENOR)
```

TSP e o algoritmo guloso Se uma resposta degradada for aceitável, pode-se aplicar o algoritmo guloso neste problema. A estratégia aqui é das cidades não visitadas, devo escolher a mais próxima. Em um caso real, obtiveram-se os seguintes valores:

Cidades: 1-Cascavel, 2-Fco Beltrao, 3-Guaira, 4-Guarapuava, 5-Irati, 6-Ortigueira, 7-Paranavai, 8-Ponta Grossa, 9-Telêmaco Borba e 10-Umuarama. Inicial: Ponta Grossa: Solução exata: 1111.97 Km. Sequência: 8 5 4 2 1 3 10 7 6 9 8. Algoritmo Guloso: 1329.56 Km. Sequência: 8 5 4 6 9 7 10 3 1 2 8. No exemplo antes resolvido, o menor caminho coincidiu com a resposta do algoritmo guloso (1560.37 Km). Isso pode acontecer em instâncias pequenas do problema.

Para você fazer

Suponha a seguinte lista de cidades

1-ADRIANOPOLIS	495 146
2-CAMPO MOURAO	164 216
3-CURITIBA	467 63
4-FOZ IGUAÇU	-59 55
5-GEN CARNEIRO	268 -49
6-GUAIRA	-25 213
7-GUARAPUAVA	254 67
8-LONDRINA	291 297
9-PONTA GROSSA	377 102

Sendo que o início e o fim do ciclo de viagens deve ocorrer na cidade de LONDRINA . Após calcular o menor caminho entre todas as cidades, responda:

1. Qual o comprimento em quilômetros do menor caminho ?
2. Qual o NOME da primeira/última cidade a ser visitada ?
3. Qual o NOME da última/primeira cidade a ser visitada ?

distância	prim/ult	ult/prim



Problema do caixeiro viajante - TSP

O problema do caixeiro-viajante (PCV), ou como é conhecido em inglês: travelling salesman problem (TSP), é um problema de otimização que, apesar de parecer modesto é, na realidade, um dos mais estudados, por cientistas, matemáticos e investigadores de logística, genética e produção, entre outros (Applegate et al., cop. 2006, p. 1). O problema pertence à categoria NP-hard (em inglês), que o remete para um campo de complexidade exponencial, isto é, o esforço computacional necessário para a sua resolução cresce exponencialmente com o tamanho do problema. Assim, dado que é difícil, se não impossível, determinar a solução ótima desta classe de problemas, os métodos de resolução passam pelas heurísticas e afins que, do ponto de vista matemático, não asseguram a obtenção de uma solução ótima (Cunha, 2002).

Definição e formulação do problema Em termos leigos, imaginem um território de vendas (por exemplo, o Estado do Paraná), que tem diversas cidades polo que devem ser visitadas pelo vendedor. Ele começa e termina na sua base e deve visitar uma e só uma vez cada cidade polo. O critério para a escolha da ordem das cidades é minimizar a distância percorrida.

Usando uma definição acadêmica:

O problema do caixeiro-viajante consiste na procura de um circuito que possua a menor distância, começando numa qualquer cidade, entre várias, visitando cada cidade precisamente uma vez e regressando à cidade inicial (Nilsson, 1982).

O tamanho do espaço de procura aumenta exponencialmente dependendo da quantidade de cidades n , o número de cidades, uma vez que existem $(n-1)!/2$ circuitos possíveis (a posição inicial é arbitrária, e a ordem do circuito pode ser invertida). A solução do PCV pode ser de dois tipos: a exata (impraticável em instância médias ou grandes, devido ao comportamento exponencial) e as aproximadas, todas dependentes de alguma heurística. Começando pelo segundo grupo, há diversos métodos. Estes são procedimentos particulares, o que os torna inflexíveis para a determinação de boas soluções para um outro problema ligeiramente diferente.

Quadro esquemático de soluções aproximadas

- construção de circuitos (alg. guloso)
- árvore de cobertura mínima
- *simulated annealing*
- algoritmo de busca tabu
- redes neurais
- algoritmos genéticos
- *aint colony optimization algorithm*

Solução Exata Embora demorada esta é a solução que vai ser trabalhada neste exercício. É um bom exemplo de algoritmo recursivo (neste caso, claramente a recursão vem em nosso auxílio: este mesmo algoritmo não recursivo teria enorme complicação e dificuldade) e suficiente pequeno para poder rodar em minutos usando um computador pessoal simples.

Um caso real Em 1998 uma equipe de matemáticos encontrou o caminho mais curto para visitar as 13.509 cidades americanas que tinham, naquele ano, mais de 500 habitantes. Foram necessários 3,5 meses de processamento de três multiprocessadores (32 pentium cada) ligados em rede. Aqui, eliminaram-se as rotas obviamente ineficientes logo de cara. O problema: a estratégia só vale para este problema e para estas cidades.

Outro caso real Os algoritmos mostrados nesta folha e rodados em um micro bem lerdinho (um pentium 4 com 2.8GHz, com 632 MB de RAM, mas rodando Linux e neste sob Wmware um Windows XP, e neste rodando APL2 da IBM) e em um micro rápido (2 CPUS de 2.80GHz, 3.3GB de memória e com APL2 sob Windows XP nativo)

qtd de cidades	CPU lerda	CPU rápida
6 cidades	31 milisseg	desprezível
7 cidades	156 milisseg	78 milisseg
8 cidades	1.1 seg	500 milisseg
9 cidades	9 seg	4 seg
10 cidades	81 seg	37 seg
11 cidades	836 seg	367 seg
12 cidades	153 min	67 min
13 cidades	30.6 horas	13.4 horas
14 cidades	16.5 dias	7.2 dias

O problema que você vai resolver

Há uma lista de 10 cidades paranaenses, cada uma com suas coordenadas obtidas a partir de um eixo cartesiano imaginário montado sobre o paralelo 26° SUL (x) e sobre o meridiano 54°O (y). No cálculo, considerou-se uma projeção plana do território, o que certamente introduzirá pequenas distorções. Não importa, avante ! Seja um exemplo de 10 cidades:

1-CAMPO MOURAO	164 216
2-CURITIBA	467 63
3-FCO BELTRAO	91 -8
4-GUAIRA	-25 213
5-IRATI	338 62
6-LONDRINA	291 297
7-ORTIGUEIRA	310 202
8-PALMAS	200 -54
9-PARANAGUA	543 51
10-PARANAVAI	158 323

Supondo que a cidade inicial seja 2 (Curitiba), a sequência de cidades a visitar é 2 5 8 3 4 1 10 6 7 9 2 e a distância é de 1560.3 km

Algoritmos Para construir a matriz de distâncias entre pares de cidades, o algoritmo é

```
1: funcao ACHADIST (COORD: nome,xx,yy[nn])
   : matd [nn;nn]
2: i,j : inteiro
3: para i = 1 to nn
4:   para j = 1 to nn
5:     matd[i][j] ← COORD[i;2 3] distancia COORD[j;2 3]
6:   fim{para}
7: fim{para}
```

Note-se que a função *distancia* acima usada é a simples aplicação do Teorema de Pitágoras sobre o mapa. Note-se também que se o TSP deve ser feito sobre outro conceito que não a distância linear, esta função deve ser substituída pela que calcula o método desejado. Este pode ser: distância via estradas, tempo de voo, custo do pedágio, ...

Montada a matriz de distâncias, é hora de chamar a função TSP, mas antes, são necessárias algumas variáveis globais: MENOR: a menor distância até agora encontrada e inicializada com $+\infty$ e CMENOR: o caminho que dá origem à menor distância até agora encontrada.

```
1: funcao TSP (matd, dateaqui,futuro,passado)
2: se futuro[1] ≠ 1
3:   J ← 1
4:   enquanto J ≤ futuro[1]
5:     D1 ← dateaqui +
6:     matd[passado[1+passado[1]][futuro[1+J]]
7:     novofuturo ← excluivv (futuro[J+1], futuro)
8:     novopassado ← passado
9:     novopassado[1] ← novopassado[1]+1
10:    novopassado[1+novopassado[1]]← futuro[J+1]
11:    TSP (matd,D1,novopassado,novofuturo)
12:    J ← J + 1
```

```
13:   fim{enquanto}
14: senão
15:   D1 ← dateaqui +
16:   matd[passado[1+passado[1]][futuro[2]]
17:   D1 ← D1+matd[futuro[2]][passado[2]]
18:   passado[passado[1]+2]←futuro[2]
19:   passado[1]←passado[1]+2
20:   se D1 < MENOR
21:     MENOR ← D1
22:     CMENOR ← passado
23:   fim{se}
24: fim{se}
```

Os vetores *passado* e *futuro* ambos tem na primeira posição um contador de valores válidos (para usar parâmetros de tamanho fixo), e eles contém as cidades já visitadas (*passado*) e as cidades a visitar (*futuro*). A função *excluivv* subtrai um no contador do vetor e exclui no vetor a cidade citada no parâmetro.

TSP e o algoritmo guloso Se uma resposta degradada for aceitável, pode-se aplicar o algoritmo guloso neste problema. A estratégia aqui é *das cidades não visitadas, devo escolher a mais próxima*. Em um caso real, obtiveram-se os seguintes valores: Cidades: 1-Cascavel, 2-Fco Beltrao, 3-Guaíra, 4-Guarapuava, 5-Irati, 6-Ortigueira, 7-Paranavai, 8-Ponta Grossa, 9-Telêmaco Borba e 10-Umuarama. Inicial: Ponta Grossa: Solução exata: 1111.97 Km. Sequência: 8 5 4 2 1 3 10 7 6 9 8. Algoritmo Guloso: 1329.56 Km. Sequência: 8 5 4 6 9 7 10 3 1 2 8

Para você fazer

Suponha a seguinte lista de cidades

1-CAMPO MOURAO	164 216
2-CANDIDO ABREU	269 157
3-CURITIBA	467 63
4-IRATI	338 62
5-ORTIGUEIRA	310 202
6-PARANAGUA	543 51
7-PARANAVAI	158 323
8-PONTA GROSSA	377 102
9-UMUARAMA	72 246

Sendo que o início e o fim do ciclo de viagens deve ocorrer na cidade de PARANAGUA . Após calcular o menor caminho entre todas as cidades, responda:

1. Qual o comprimento em quilômetros do menor caminho ?
2. Qual o NOME da primeira/última cidade a ser visitada ?
3. Qual o NOME da última/primeira cidade a ser visitada ?

distância	prim/ult	ult/prim



Árvores e recursividade usando apontadores

As árvores são estruturas de dados das mais úteis. Presentes em qualquer índice de banco de dados, em corretores ortográficos, em compressores de dados, em dispositivos de acesso a dados, a lista é infundável. Em particular, no capítulo de árvores (outro exemplo: LISP) tem-se que, usar recursividade deixa os algoritmos mais simples, esbeltos e elegantes. Daí porque, mesmo os que torcem o nariz para a recursividade, precisarão abrir uma exceção para estudar árvores e reconhecer aqui a potência da técnica. Vamos usar a Árvore Binária de Pesquisa, pela sua simplicidade e potência. Nesta árvore binária (grau=2), as chaves determinam a regra de sua construção: menores ou iguais à esquerda e maiores à direita. A programação desta folha está baseada em material da prof^{ta} Divani Gavini, disponível na Internet.

Vai-se mostrar o algoritmo básico de árvores (conceito, criação, inclusão e exclusão, pesquisa e caminhamento) usando um programa Python orientado a objeto.

Elementos

```
class Nodo:
    def __init__(self, es, di, key):
        self.es = es
        self.di = di
        self.chave = key
```

```
class arvore:
    def __init__(self):
        self.raiz = Nodo(None, None, None)
        self.raiz = None
```

Aqui definem-se as 2 áreas de dados: o nodo da árvore que será composto por 3 números: os endereços do filho esquerdo e direito e o valor da chave do nodo. A árvore que, quando é criada, tem sua raiz vazia.

```
def insercao(self, valor):
    nn = Nodo(None, None, valor)
    if self.raiz == None:
        self.raiz = nn
    else:
        vamos = self.raiz
        while 1==1:
            antes = vamos
            if valor <= vamos.chave:
                vamos = vamos.es
                if vamos == None:
                    antes.es = nn
                    return
            else:
                vamos = vamos.di
                if vamos == None:
                    antes.di = nn
                    return
```

A inserção começa criando um novo nodo, que conterá (-1,-1,valor) Daí, se a árvore estava vazia (raiz=-1) a raiz passa a apontar para o nodo recém criado e estamos feitos. Senão, precisa-se navegar na árvore até achar o lugar correto de inserção do novo nodo. Usa-se a regra básica das ABPs: *se menor ou igual: vá a esquerda e se maior: vá a direita*. Antes de cada passo do laço, guarda-se o apontador do antecessor, pois ele será necessário para ligar o novo nodo na árvore (o atual será None).

```
def pesquisa(self, sujeito):
    if self.raiz == None:
        return None
    vamos = self.raiz
    while vamos.chave != sujeito:
        if sujeito < vamos.chave:
            vamos = vamos.es
        else:
            vamos = vamos.di
    if vamos == None:
        return None:
    return vamos
```

Esta é a pesquisa: sempre começa na raiz até achar a chave ou um endereço valendo -1 na busca. Neste caso o valor buscado não estará na árvore.

```
def exclue(self, quem):
    if self.raiz == None:
        return False
    vamos = self.raiz
```

```
pai = vamos
esqdir = 0
while vamos.chave != quem:
    pai = vamos
    if quem < vamos.chave:
        esqdir = 0
        vamos = vamos.es
    else:
        vamos = vamos.di
        esqdir = 1
    if vamos == None:
        return False
if vamos.es == None and vamos.di == None:
    if vamos == self.raiz:
        self.raiz = None
    else:
        if esqdir == 0:
            pai.es = None #1
        else:
            pai.di = None #1
elif vamos.es == None: # só tem f. direito
    if vamos == self.raiz:
        self.raiz = vamos.di
    else:
        if esqdir == 0:
            pai.es = vamos.di #2
        else:
            pai.di = vamos.di #2
elif vamos.di == None: # só tem f. esquerdo
    if vamos == self.raiz:
        self.raiz = vamos.es
    else:
        if esqdir == 0:
            pai.es = vamos.es #2
        else:
            pai.di = vamos.es #2
else: # tem os 2 filhos
    sucessor = vamos.di
    while sucessor.es != None:
        pai = sucessor
        sucessor = sucessor.es
    arv.exclue(sucessor.chave) #3
    vamos.chave = sucessor.chave #3
return True
```

A função de exclusão é a mais sofisticada, já que ao excluir um nodo, se ele tiver filhos, há que fazer algo com eles (*não dá para jogar fora a água do banho com o bebê dentro*). A coisa funciona assim, lembrando que após a exclusão efetuada, a árvore binária de pesquisa precisa continuar obedecendo a regra básica: **menores à esquerda e maiores à direita**.

- Se o nodo a excluir não tem filhos, a exclusão é trivial: basta anular o apontador do pai dele para ele. No código isto aparece com #1.
- Se o nodo a excluir tem 1 filho apenas: este filho passa a ocupar o lugar do nodo excluído. No código #2
- O nodo tem 2 filhos: Deve-se buscar na árvore o sucessor do nodo a excluir. Este sucessor não terá filho à esquerda (porque?). O sucessor deve ser excluído e o sucessor ocupa o lugar do nodo a excluir. Veja #3.

Caminhamentos

```
def emordem(self, atual):
    if atual != None:
        self.emordem(atual.es) #1
        print(atual.chave, end=" ") #2
        self.emordem(atual.di) #3
def largura(self, atual):
    if atual != None:
        fila=[atual]
        while len(fila)!=0:
            print(fila[0].chave, end=' ')
            if fila[0].es != None:
                fila.append(fila[0].es)
            if fila[0].di != None:
                fila.append(fila[0].di)
            fila.pop(0)
```

As definições usuais de caminhamento:

preordem Usa a ordem: raiz-esquerda-direita. Use #2, #1 e #3.

emordem Usa a ordem: esquerda-raiz-direita.

posordem Usa a ordem: esquerda-direita-raiz. Use #1, #3 e #2.

largura Fátia a árvore, usando uma fila para guardar os filhos.

```
def alt(self, eu):
    if eu==None or eu.es==None and eu.di==None:
        return 0
    else:
        if self.alt(eu.es)>self.alt(eu.di):
```

```
        return 1 + self.alt(eu.es)
    else:
        return 1 + self.alt(eu.di)
```

```
def folhas(self, eu):
    if eu == None:
        return 0
    if eu.es == None and eu.di == None:
        return 1
    return self.folhas(eu.es)+self.folhas(eu.di)
```

```
def contarNos(self, eu):
    if eu == None:
        return 0
    else:
        return 1+self.contarNos(eu.es)
        + self.contarNos(eu.di)
```

As funções acima mostram a altura da árvore, a quantidade de folhas e a quantidade de nodos na árvore.

Desenhando

```
def des(self, atual):
    global nu, fi
    if atual.es == None and atual.di == None:
        return
    if atual.es == None:
        fi.write('{0} -> null{1}; null{1} \
[shape=point]\n'.format(atual.chave, nu))
        nu=nu+1
        fi.write('{0} -> {1} \n' \
        .format(atual.chave, atual.di.chave))
        self.des(atual.di)
        return
    if atual.di == None:
        fi.write('{0} -> {1} \n' \
        .format(atual.chave, atual.es.chave))
        self.des(atual.es)
        fi.write('{0} -> null{1}; null{1} \
[shape=point]\n'.format(atual.chave, nu))
        nu=nu+1
        return
    fi.write('{0} -> {1} \n' \
    .format(atual.chave, atual.es.chave))
    self.des(atual.es)
    fi.write('{0} -> {1} \n' \
    .format(atual.chave, atual.di.chave))
    self.des(atual.di)
    return
def desenha(self, atual):
    global nu, fi
    fi = open("dese.dox", "w")
    nu=0
    fi.write("digraph BST { \n")
    self.des(atual)
    fi.write('}\n')
    fi.close()
    return
```

O código acima (junto com o aplicativo graphviz: <https://graphviz.org/>) permite desenhar a árvore ABP. Execute simplesmente o comando
> dot dese.dox -T png -o dese11.png
e use qualquer visualizador de imagens para ver a árvore.

Uma aplicação simples Para usar estes códigos...

```
arv = arvore()
arv.insercao(22)
arv.insercao(4)
...
arv.largura(arv.raiz)
...
```

🔗 Para você fazer

Escreva estes códigos no Python. Crie uma ABP com as chaves (nesta ordem)
63 39 61 4 20 42 92 11 59 87
22 16 78 27 79 77 56 14 67 76
Depois exclua as seguintes chaves (nesta ordem)
20 16 59 14 4 22

Anote aqui o décimo e o décimo terceiro elementos da árvore resultante no caminhamento em largura

10 ^o	13 ^o



Árvores usando cursores

Esta folha deve ser examinada vis-à-vis com a anterior (260b) uma vez que a temática é a mesma (árvores binárias com recursividade), mas ao invés de usar apontadores vão ser usados cursores. O balanço é o de sempre: apontadores são mais eficientes (menos indireções, mais próximo do baixo nível) enquanto os cursores são mais amigáveis na depuração, já que é muito fácil imprimir a matriz de cursores e acompanhar o que o programa fez-faz e fará.

Se não for possível abandonar os apontadores minha recomendação é: escreva o programa usando cursores, depure-o, certifique-o e só então substitua os cursores pelos apontadores. Você terá o melhor dos dois mundos.

Vai-se mostrar o algoritmo básico de árvores (conceito, criação, inclusão e exclusão, pesquisa e caminhamento) usando cursores em um programa Python. O terminador será -1, *as usual*.

```
import numpy as np
global arv, ultimo
arv=np.full((25,3),-1,int)
ultimo = 0
O nodo da árvore que será composto por 3 núme-
ros: os endereços do filho esquerdo e direito e o
valor da chave do nodo. A árvore que, quando é
criada, tem sua raiz vazia. ultimo aponta para a
última linha usada na matriz arv.
def insercao(valor):
    global arv,ultimo
    if ultimo == 0:
        arv[ultimo][2] = valor
        ultimo = 1
        return
    else:
        arv[ultimo]=[-1,-1,valor]
        ultimo=ultimo+1
        vamos = 0
        while 1==1:
            antes = vamos
            if valor <= arv[vamos][2]:
                vamos = arv[vamos][0]
                if vamos == -1:
                    arv[antes][0] = ultimo-1
                    return
            else:
                vamos = arv[vamos][1]
                if vamos == -1:
                    arv[antes][1] = ultimo-1
                    return
```

A inserção começa criando um novo nodo, que conterá (-1,-1,valor) Daí, se a árvore estava vazia (raiz=-1) a raiz é criada na posição 0 da matriz e estamos feitos. Senão, precisa-se navegar na árvore até achar o lugar correto de inserção do novo nodo. Usa-se a regra básica das ABPs: *se menor ou igual: vá a esquerda e se maior: vá a direita*. Antes de cada passo do laço, guarda-se o apontador do antecessor, pois ele será necessário para ligar o novo nodo na árvore (o atual será None).

```
def pesquisa(sujeito):
    if ultimo == 0:
        return -1
    vamos = 0
    while arv[vamos][2] != sujeito:
        if sujeito < arv[vamos][2]:
            vamos = arv[vamos][0]
        else:
            vamos = arv[vamos][1]
        if vamos == -1:
            return -1
    return vamos
```

Esta é a pesquisa: sempre começa na raiz (0) até achar a chave ou um endereço valendo -1 na busca. Neste caso o valor buscado não estará na árvore.

```
def exclue(quem):
    global arv,ultimo
    if ultimo == 0:
        return -1
    vamos = 0
    pai = vamos
    esqdir = 0
    while arv[vamos][2] != quem:
        pai = vamos
        if quem < arv[vamos][2]:
```

```
        esqdir = 0
        vamos = arv[vamos][0]
    else:
        vamos = arv[vamos][1]
        esqdir = 1
    if vamos == -1:
        return -1
    if arv[vamos][0]==-1 and arv[vamos][1]==-1:
        if vamos == 0:
            ultimo = 0
        else:
            if esqdir == 0:
                arv[pai][0] = -1
            else:
                arv[pai][1] = -1
    elif arv[vamos][0] == -1: # só tem f. direito
        if vamos == 0:
            ultimo = 0
        else:
            if esqdir == 0:
                arv[pai][0] = arv[vamos][1]
            else:
                arv[pai][1] = arv[vamos][1]
    elif arv[vamos][1] == -1: # só tem f. esquerdo
        if vamos == 0:
            ultimo = 0
        else:
            if esqdir == 0:
                arv[pai][0] = arv[vamos][0]
            else:
                arv[pai][1] = arv[vamos][0]
    else: # tem os 2 filhos
        sucessor = arv[vamos][1]
        while arv[sucessor][0] != -1:
            pai = sucessor
            sucessor = arv[sucessor][0]
        exclue(arv[sucessor][2])
        arv[vamos][2] = arv[sucessor][2]
    return True
```

A função de exclusão é a mais sofisticada, já que ao excluir um nodo, se ele tiver filhos, há que fazer algo com eles (*não dá para jogar fora a água do banho com o bebê dentro*). A coisa funciona assim, lembrando que após a exclusão efetuada, a árvore binária de pesquisa precisa continuar obedecendo a regra básica: **menores à esquerda e maiores à direita**.

- Se o nodo a excluir não tem filhos, a exclusão é trivial: basta anular o apontador do pai dele para ele. No código isto aparece com #1.
- Se o nodo a excluir tem 1 filho apenas: este filho passa a ocupar o lugar do nodo excluído. No código #2
- O nodo tem 2 filhos: Deve-se buscar na árvore o sucessor do nodo a excluir. Este sucessor não terá filho à esquerda (porque ?). O sucessor deve ser excluído e o sucessor ocupa o lugar do nodo a excluir. Veja #3.

Caminhamentos

```
def emordem(atual):
    if arv[atual][0] != -1:
        emordem(arv[atual][0]) #1
    print(arv[atual][2],end=" ") #2
    if arv[atual][1] != -1:
        emordem(arv[atual][1]) #3
```

```
def largura(atual):
    if atual != -1:
        fila=[atual]
        while len(fila)!=0:
            print(arv[fila[0]][2],end=' ')
            if arv[fila[0]][0] != -1:
                fila.append(arv[fila[0]][0])
            if arv[fila[0]][1] != -1:
                fila.append(arv[fila[0]][1])
            fila.pop(0)
```

As definições usuais de caminhamento:

preordem Usa a ordem: raiz-esquerda-direita. Use #2, #1 e #3.

emordem Usa a ordem: esquerda-raiz-direita.

posordem Usa a ordem: esquerda-direita-raiz. Use #1, #3 e #2.

largura Fatia a árvore, usando uma fila para guardar os filhos.

```
def altura(atual):
    if arv[atual][0] == -1 and arv[atual][1] == -1:
        return 0
    else:
        if altura(arv[atual][0])>altura(arv[atual][1]):
            return 1 + altura(arv[atual][0])
        else:
            return 1 + altura(arv[atual][1])
```

```
def folhas(atual):
    if atual == -1:
        return 0
    if arv[atual][0]==-1 and arv[atual][1]==-1:
        return 1
    return folhas(arv[atual][0]) + \
        folhas(arv[atual][1])
def contarNos(atual):
    if atual == -1:
        return 0
    else:
        return 1 + contarNos(arv[atual][0]) \
            + contarNos(arv[atual][1])
```

As funções acima mostram a altura da árvore, a quantidade de folhas e a quantidade de nodos na árvore.

Desenhando

```
def des(atual):
    global arv,ultimo,nu,fi
    if arv[atual][0] == -1 and arv[atual][1] == -1:
        return
    if arv[atual][0] == -1:
        fi.write('{0} -> null{1}; null{1} \
[shape=point]\n'.format(arv[atual][2],nu))
        nu=nu+1
        fi.write('{0} -> {1} \n'.format(arv[atual][2] \
,arv[atual][1][2]))
        des(arv[atual][1])
        return
    if arv[atual][1] == -1:
        fi.write('{0} -> {1} \n'.format(arv[atual][2] \
,arv[atual][0][2]))
        des(arv[atual][0])
        fi.write('{0} -> null{1}; null{1} \
[shape=point]\n'.format(arv[atual][2],nu))
        nu=nu+1
        return
    fi.write('{0} -> {1} \n'.format(arv[atual][2] \
,arv[atual][0][2]))
    des(arv[atual][0])
    fi.write('{0} -> {1} \n'.format(arv[atual][2] \
,arv[atual][1][2]))
    des(arv[atual][1])
    return
```

```
def desenha(atual,arquivo):
    global arv,ultimo,nu,fi
    fi = open(arquivo,"w")
    nu=0
    fi.write("digraph BST { \n")
    des(0)
    fi.write('}\n')
    fi.close()
    return
```

O código acima (junto com o aplicativo graphviz: <https://graphviz.org/>) permite desenhar a árvore ABP. Execute simplesmente o comando
> dot dese.dox -T png -o dese11.png
e use qualquer visualizador de imagens para ver a árvore.

Uma aplicação simples

Para usar estes códigos...

```
arv = arvore()
arv.insercao(22)
arv.insercao(4)
...
arv.largura(arv.raiz)
...
```

🔗 Para você fazer

Escreva estes códigos no Python. Crie uma ABP com as chaves (nesta ordem)

99 97 86 6 91 19 82 18 20 75
69 3 32 88 96 55 59 89 70 52

Depois exclua as seguintes chaves (nesta ordem)

96 88 75 55 18 52

Anote aqui o décimo e o décimo terceiro elementos da árvore resultante no caminhamento em largura

10°	13°



- 1 - /

Árvores k-d

Árvore k-d é uma estrutura muito boa para estudar distribuição de pontos em (hiper)espaços geométricos. Por simplicidade vai-se estudar a estrutura em espaços 2D, mas o conceito pode ser facilmente estendido para mais dimensões, quantas se quiser. A matemática agradece. Ou melhor dizendo, agradecemos à matemática.

A idéia do algoritmo subjacente é receber um conjunto de pontos, usualmente através de suas coordenadas e construir uma árvore subdividindo as regiões do espaço. Esta estratégia segue até que cada ponto esteja isolado em uma região. Nessa hora, analisar essas regiões pode permitir diversas leituras.

Existe um exercício nesta coleção (o vivx255) que introduz o conceito de árvore k-d no âmbito de compressão de imagens digitais.

Deve-se ressaltar que as árvores k-d surgem em inúmeros sabores sendo que todos compartilham a decomposição hierárquica de pontos em poucas células. A partição acaba após lg n níveis, onde n é o número de pontos.

Nesta implementação, a árvore terá grau=4, já que a região em análise será cortada por 2 planos ortogonais entre si, dividindo a região em 4 partes (quadrantes), que serão numerados como em

1	2
3	4

Para mostrar a liberdade do programador neste tipo de implementação, vão-se usar 2 estratégias de corte

- 1. Cortar as duas partes de maneira que cada uma tenha a metade dos pontos
- 2. Cortar na metade do espaço geográfico, fazendo com que as duas partes tenham a mesma área.

Para a primeira estratégia, deve-se ordenar os pontos presentes na área pela sua coordenada-X e selecionar a coordenada x do ponto mediano. Se a quantidade de pontos for par, a coordenada x do corte deve ser a média aritmética das coordenadas x dos dois pontos centrais. O mesmo procedimento deve ser feito para a coordenada y do corte.

Para a segunda estratégia a coisa é mais simples: basta tomar a média aritmética das coordenadas x e y da região e estão obtidos os dois cortes a efetuar.

Veja-se um exemplo do que se fala. Se a coleção de pontos for 1 2; 2 2; 3 3; 6 1; 6 6 e 9 7 com limites entre 0 e 10 (x) e 0 e 8 (y) nas respostas a seguir os primeiros 4 valores são x_{min}, x_{max}, y_{min} e y_{max}. O próximo número é a quantidade de pontos e depois vem a sequência de pontos. Pontos limítrofes são agregados pelo primeiro quadrante considerado: primeiro o 1, depois o 2 e assim por diante. A seguir, pode-se comparar as 2 estratégias acima descritas:

pontos3 gerafilhos1 0 10 0 8				
0	4.5	0	2.5	2
				2 2
4.5	10	0	2.5	1
				6 1
0	4.5	2.5	8	1
				3 3
4.5	10	2.5	8	2
				6 6
				9 7

enquanto a estratégia 2 gerará as regiões

pontos3 gerafilhos2 0 10 0 8				
0	5	0	4	3
				1 2
				2 2
				3 3
5	10	0	4	1
				6 1
0	5	4	8	0
5	10	4	8	2
				6 6
				9 7

A seguir vai ser descrita a função que cria a árvore k-d. É uma função recursiva o que vai facilitar muito a sua implementação. Lembrando, uma função recursiva precisa ter 3 características:

- 1. A função chama a si mesma
- 2. Precisa ter um caso básico onde deixa de haver a recursão
- 3. A cada chamada o universo (os parâmetros) deve ser menor (na direção do caso básico)

Aqui o caso básico é quando a função é chamada com zero pontos. O universo menor é um quarto da chamada original. Nos exemplos acima compare a região da chamada (valores ao lado direito da função gerafilhos1 ou gerafilhos2) com as chamadas “menores” nas quatro regiões. O nodo da árvore tem os seguintes atributos:

x_{min} coordenada x mínima deste nodo

x_{max} coordenada x máxima deste nodo

y_{min} coordenada y mínima deste nodo

y_{max} coordenada y máxima deste nodo

quantidade de pontos neste nodo

pai endereço do pai deste nodo na árvore

área deste nodo em unidades de coordenadas

F₁, F₂, F₃ e F₄ endereços dos 4 filhos numerados conforme a figura acima. Se algum dos filhos não tiver pontos não será gerado nodo para ele e o endereço no pai aparecerá como 0 que deve ser entendido como terminador, já que nesta árvore a origem dos índices é 1.

Exemplo completo

Seja um caso com 20 pontos a seguir

9 8, 8 1, 6 4, 10 3, 7 9, 6 10, 3 9, 1 8, 2 10, 2 2, 1 9, 3 2, 6 8, 10 5, 2 8, 7 7, 2 5, 8 8, 9 4 e 4 1

e gerou a seguinte árvore (usando a estratégia 1)

	xi	xf	yi	yf	qt	pai	area	f1	f2	f3	f4
1=	0	10	0	10	20	-1	100	2	9	16	26
2=	0	6	0	7.5	5	1	45	3	6	7	8
3=	0	3	0	2	2	2	6	4	5	0	0
4=	0	2.5	0	2	1	3	5	0	0	0	0
5=	2.5	3	0	2	1	3	1	0	0	0	0
6=	3	6	0	2	1	2	6	0	0	0	0
7=	0	3	2	7.5	1	2	16.5	0	0	0	0
8=	3	6	2	7.5	1	2	16.5	0	0	0	0
9=	6	10	0	7.5	5	1	30	10	13	14	15
10=	6	9	0	4	2	9	12	11	0	0	12
11=	6	8.5	0	2.5	1	10	6.25	0	0	0	0
12=	8.5	9	2.5	4	1	10	0.75	0	0	0	0
13=	9	10	0	4	1	9	4	0	0	0	0
14=	6	9	4	7.5	1	9	10.5	0	0	0	0
15=	9	10	4	7.5	1	9	3.5	0	0	0	0
16=	0	6	7.5	10	7	1	15	17	21	24	25
17=	0	2	7.5	9	3	16	3	18	19	20	0
18=	0	1	7.5	8	1	17	0.5	0	0	0	0
19=	1	2	7.5	8	1	17	0.5	0	0	0	0
20=	0	1	8	9	1	17	1	0	0	0	0
21=	2	6	7.5	9	2	16	6	0	22	23	0
22=	4.5	6	7.5	8.5	1	21	1.5	0	0	0	0
23=	2	4.5	8.5	9	1	21	1.25	0	0	0	0
24=	0	2	9	10	1	16	2	0	0	0	0
25=	2	6	9	10	1	16	4	0	0	0	0
26=	6	10	7.5	10	3	1	10	27	28	29	0
27=	6	8	7.5	8	1	26	1	0	0	0	0
28=	8	10	7.5	8	1	26	1	0	0	0	0
29=	6	8	8	10	1	26	4	0	0	0	0

Uma possível interpretação para estes dados é analisar os nodos folhas da árvore (ou seja os que descrevem um único ponto). Note que se a árvore está corretamente construída, deve haver tantas folhas quantos pontos originais havia. Uma informação interessante destes nodos é a área do nodo. Ordenando estes nodos por área os pontos serão agrupados conforme sua proximidade. Veja-se este processamento aplicado aos dados acima

x	y	área
1	8	0.5
2	8	0.5
9	4	0.75
3	2	1
1	9	1
8	8	1

9	8	1
3	9	1.25
6	8	1.5
2	10	2
10	5	3.5
10	3	4
6	10	4
7	9	4
2	2	5
4	1	6
8	1	6.25
7	7	10.5
2	5	16.5
6	4	16.5

Duas possíveis referências para este tema são: SKIENA, Steven. **The Algorithm Design Manual**. Springer, New York, 1998, pág. 194 e seguintes. DEWDNEY, A. K. **The Turing Omnibus**. W. Freeman, New York, 1989, pág 290 e seguintes.

Mais um exemplo Supondo os 20 pontos a seguir e construindo a árvore k-d de tipo 1, e analisando os nodos folha da árvore o nodo de menor área (0.02) é o ponto 7.30 3.20 e o nodo de maior área (9.765) é o ponto 2.30 5.

5.00, 1.00	6.00, 2.00
2.00, 1.10	7.20, 5.20
9.00, 3.00	7.10, 6.60
4.00, 6.00	5.50, 1.60
4.20, 1.20	2.10, 2.10
5.20, 6.30	4.30, 2.30
2.30, 5.00	2.10, 7.00
8.00, 3.10	6.30, 7.30
7.00, 1.30	6.20, 1.40
8.10, 4.00	7.30, 3.20

Para você fazer

Imagine um universo de 20 pontos cujas coordenadas são

3.00, 7.00	1.30, 1.00
1.00, 6.00	5.00, 7.40
6.00, 2.00	7.30, 3.00
8.00, 2.20	4.00, 7.60
3.30, 7.20	7.60, 6.30
2.00, 7.30	2.20, 4.10
3.20, 5.00	8.30, 2.40
7.00, 5.20	5.10, 6.20
6.20, 4.00	5.30, 2.30
1.20, 4.30	2.30, 4.20

Aplicando o algoritmo 1 (acima descrito) e analisando a área de cada nodo, o ponto cuja área é menor (que, a propósito vale 0.165 unidades de área ao quadrado) é o ponto cujas coordenadas são

x _{min}	x _{max}
e o ponto cuja área é maior (valendo 11.4125) é o ponto cujas coordenadas são	
x _{min}	x _{max}



Recursividade: Strassen e Quickselect

Algoritmo de Strassen O algoritmo de Strassen prevê uma pequena economia ao realizar a multiplicação matricial. A importância deste algoritmo é que durante anos, a ninguém ocorreu que alguma otimização pudesse ser feita sobre o algoritmo canônico. Strassen percebeu que em $r = a.b$ (a e b matrizes conformáveis) se ambas fossem divididas em 4 (com cortes ao meio), para obter a resposta, ao invés de fazer 8 multiplicações, eram necessárias apenas 7, acompanhe:

- a é dividida em 4: a_{11} , a_{12} , a_{21} e a_{22}
- idem para b
- agora são calculadas 7 sub-matrizes recursivamente, usando diversas parcelas envolvendo a_{ij} e b_{km} .
- finalmente, o resultado é obtido executando operações elementares (adição e subtração) sobre as 7 sub-matrizes.

Um detalhe importante é que a se a matriz a multiplicar não tiver dimensões iguais a uma potência de 2, basta acrescentar zeros nas linhas ou nas colunas para atingir esta condição. Tais linhas e colunas podem ser retiradas depois sem alterar o resultado numérico da operação.

Acompanhe o algoritmo e se quiser efetue a multiplicação matricial canônica para conferir o resultado correto do algoritmo de Strassen.

```
#strassen em Python, usando o pacote numpy
import numpy as np
import random as ra
import time
tam=512
a=np.zeros((tam,tam),float)
for i in range(0,tam):
    for j in range(0,tam):
        a[i][j]=ra.random()
b=np.zeros((tam,tam),float)
for i in range(0,tam):
    for j in range(0,tam):
        b[i][j]=ra.random()

def marra(a,b):
    tam=len(a)
    r=np.zeros((tam,tam),float)
    for i in range(tam):
        for j in range(tam):
            olha=0
            for k in range(tam):
                olha=olha+a[i][k]*b[k][j]
            r[i][j]=olha
    return(r)
```

```
def strassen(x,y):
    n=len(x)
    if n>4: # ou n>1
        me=int(n/2)
        a11=x[0:me,0:me]
        a12=x[0:me,me:n]
        a21=x[me:n,0:me]
        a22=x[me:n,me:n]
        b11=y[0:me,0:me]
        b12=y[0:me,me:n]
        b21=y[me:n,0:me]
        b22=y[me:n,me:n]
        m1=strassen((a11+a22),(b11+b22))
        m2=strassen((a21+a22),b11)
        m3=strassen(a11,(b12-b22))
        m4=strassen(a22,(b21-b11))
        m5=strassen((a11+a12),b22)
        m6=strassen((a21-a11),(b11+b12))
        m7=strassen((a12-a22),(b21+b22))
        c=np.zeros((n,n),int)
        c[0:me,0:me]=m1+m4-m5+m7
        c[0:me,me:n]=m3+m5
        c[me:n,0:me]=m2+m4
        c[me:n,me:n]=m1-m2+m3+m6
        return c
    else:
        return marra(x,y) # ou x*y
```

```
ini=time.time()
strassen(a,b)
fim=time.time()
print('Demora de strassen',fim-ini)
```

Dados reais: Eis um possível desempenho real destas funções: Para matrizes de 256×256 reais, o algoritmo de Strassen (com $n > 4$) demorou 13.8 seg e o algoritmo canônico 18.5 seg. Já para matrizes de 512×512 reais, os tempos foram de 89.6 e 150.3 segundos respectivamente. Fica de fora a análise da função primitiva do numpy-python `dot(a,b)` que tem um desempenho maravilhoso: para os casos citados demorou 0.002 e 0.015 segundos.

Quickselect Trata-se de descobrir a mediana de um conjunto de dados desordenados, sem ser necessário ordená-los. A referência é “Algoritmos para Leigos” de Mueller, John e Massaron, Luca, pág. 334.

Seja calcular a mediana de uma lista (desordenada) de números. Da estatística, sabe-se que a mediana de uma distribuição é o valor que ocupa a posição central DEPOIS que a distribuição é ordenada. (No caso da ordenação ter um número par de elementos, a mediana é a média entre os 2 valores centrais). Esta definição aponta para uma solução simples: ordenar a sequência e depois tomar o elemento central. Só que a ordenação é uma operação custosa – custa $O(n \cdot \log n)$ no mínimo, e o que se pretende aqui é achar a mediana sem precisar ordenar nada e pagando apenas $O(n)$. Pode-se batizar este algoritmo como Quickselect, ou o algoritmo do k-ésimo valor. Os passos do algoritmo recursivo são:

1. A chamada é $QS(lista, k)$
2. O algoritmo escolhe um pivot (aleatório) e divide a lista em 2: à esquerda os menores do que o pivot e à direita os maiores do que o pivot.
3. Achar o comprimento das 2 listas. Se o $tam(esquerda) > k$, então a mediana está na lista esquerda. Aplicar $QS(esquerda)$.
4. subtrair de k o tamanho da lista esquerda
5. Calcular o número de duplicações da mediana, fazendo $tam(lista) - (tam(esquerda) + tam(direito))$

- Se o número de duplicações é maior do que k , achou-se a mediana
- senão, remover o número de duplicações de k ($novok = k - duplicacoes$) e aplicar $QS(direito, novok)$.

Em python:

```
from random import randint, random, choice
import numpy as np
import sys
import time
tam=100000
a=np.zeros((tam+1),float)
for i in range(1,tam+1):
    a[i]=random()*100000
def quickselect(series,k):
    pivot=choice(series)
    #retorna um randomico da lista
    left, right = list(),list()
    for item in series:
        if item<pivot:
            left.append(item)
        if item>pivot:
            right.append(item)
    lenght_left=len(left)
    if lenght_left > k:
        return quickselect(left,k)
    k=k-lenght_left
    dupli=len(series)-(lenght_left+len(right))
    if dupli>k:
        return float(pivot)
    k=k-dupli
    return quickselect(right,k)
ini=time.time()
print(quickselect(a,(tam//2)+1))
fim=time.time()
print("tempo quickselect ",fim-ini)
```

```
def partition(arr, begin, end):
    pivot = begin
    for i in range(begin+1, end+1):
        if arr[i] <= arr[begin]:
            pivot = pivot + 1
            arr[i],arr[pivot]=arr[pivot],arr[i]
    arr[begin],arr[pivot]=arr[pivot],arr[begin]
    return pivot
```

```
def quicksort(array, begin=0, end=None):
    if end is None:
        end = len(array) - 1
    def _quicksort(array, begin, end):
        if begin >= end:
            return
        pivot = partition(array, begin, end)
        _quicksort(array, begin, pivot-1)
        _quicksort(array, pivot+1, end)
    return _quicksort(array, begin, end)
```

```
ini=time.time()
print(quicksort(a))
print(a[1+(tam//2)])
fim=time.time()
print("tempo canonico ",fim-ini)
```

Eis a chamada completa para obter a mediana:

```
def mediana(se):
    if len(se)%2 != 0:
        return quickselect(se,len(se),len(se)//2)
    else:
        le=quickselect(se,(len(se)-1)//2)
        ri=quickselect(se,(len(se)+1)//2)
        return (le+ri)/2
```

Para você fazer

Planeje uma execução comparativa entre dois ambientes: No primeiro o objetivo é achar a mediana de um vetor grande (digamos 100.001 reais) primeiro ordenando e depois listando o 50.000º número. Depois, faça a mesma medida de tempo usando esta função acima. Conclua sobre o resultado obtido.

Para seu controle em uma situação real, num computador meio lerdinho, em um vetor de 10.000 reais, o quickselect demorou 0.016 seg e o algoritmo canônico, 0.144 seg. Para um vetor de 50.000 reais os tempos foram 0.076 e 0.808 segundos.



Programação Linear: método Simplex

A programação linear apareceu na Matemática durante a II Grande Guerra. O objetivo era deslocar exércitos, armas, combustíveis, comida, e toda a logística associada. O matemático George Dantzig que era funcionário do Escritório de Controle Estatístico da Força Aérea Americana criou uma maneira inteligente de resolver este tipo de problema usando o algoritmo simplex. Note-se que a disponibilidade de computadores logo a seguir aumentou o interesse da coisa. Dantzig morreu em 2005 e o campo que ele inaugurou ainda está sob constante desenvolvimento.

Método Simplex Seja o caso: Um fazendeiro pode criar bois ou ovelhas para lã. Um boi rende 8000 por ano, e uma tonelada de lã rende 21.000/ano. Para criar 1 boi usa-se 11 ha de área. Para 1 ton de lã usa-se 60 ha de área. Um boi consome 1h/dia de manuseio e para gerar 1 ton de lã usa-se 2h/dia de manuseio. Considerando que o fazendeiro tem 500 ha de área e trabalham ele e dois filhos (o que dá 24 h/dia). Achar o melhor plano. Definindo as variáveis: x_1 =bois; x_2 =toneladas de lã; x_3 =área não utilizada (folga) e x_4 =horas não utilizadas (folga também). Z é a lucratividade do negócio.

Tableaux $Z - 8x_1 - 21x_2 = 0$
 $11x_1 + 60x_2 + x_3 = 500$
 $x_1 + 2x_2 + x_4 = 24$

Z	x_1	x_2	x_3	x_4	D
1	-8	-21	0	0	0
0	11	60	1	0	500
0	1	2	0	1	24

A coluna que deve entrar é a que tem o maior valor negativo na primeira linha, portanto a coluna x_2 .

Z	x_1	x_2	x_3	x_4	D
1	-8	-21	0	0	0
0	11	60	1	0	500
0	1	2	0	1	24
		↑			

Para determinar a linha que vai sair da base, Divida o lado direito (D) pelo coeficiente da coluna pivot (só se este for estritamente positivo (> 0)) e identifique a linha que possui a menor razão: $500/60 = 8.33$ e $24/2 = 12$, a linha menor é a linha do 500, e fica

Z	x_1	x_2	x_3	x_4	D
1	-8	-21	0	0	0
0	11	60	1	0	500 ←
0	1	2	0	1	24
		↑			

Agora temos a coluna pivot e a linha pivot e no cruzamento de ambos, o pivot.

Chegou o momento de operações elementares com linhas.

* Divida a linha pivot pelo pivot. Esta é a NOVA linha pivot. Fica

Z	x_1	x_2	x_3	x_4	D
1	-8	-21	0	0	0
0	0.183	1	0.016	0	8.33 ←
0	1	2	0	1	24
		↑			

* Para cada outra linha (inclusive a primeira)

- Se o coeficiente da coluna pivot é negativo:
 $L_{atual} = L_{atual} + |coeficiente| \times nova - linha - pivot$
- se positivo:
 $L_{atual} = L_{atual} - |coeficiente| \times nova - linha - pivot$

Primeira linha: 1, -8, -21, 000 +
[21] * 0, 0.183, 1, 0.016, 0, 8.33 e fica:
1, -4.157, 0, 0.336, 0, 174.93
Terceira linha: 0, 1, 2, 0, 1, 24 -
[2] * 0, 0.183, 1, 0.016, 0, 8.33 e fica:

0, 0.634, 0, -0.032, 1, 7.34					
Z	x_1	x_2	x_3	x_4	D
1	-4.157	0	-0.032	1	7.34
0	0.183	1	0.016	0	8.33
0	0.634	0	-0.032	1	7.34
	↑				

Linha pivot: candidatos: a segunda 8.33/0.183 ou a terceira 7.34/0.634 é a terceira.

A nova linha pivot é: 0, 1, 0, -0.0504, 1.577, 11.57

A nova linha 1 é: 1, 0, 0, -0.241, 7.555, 55.436

A nova linha 2 é: 0, 0, 1, 0.0252, -0.288, 6.212 e fica

Z	x_1	x_2	x_3	x_4	D
1	0	0	-0.2415	7.555	55.436
0	1	0	-0.0504	1.577	11.57
0	0	1	0.025	-0.288	6.212

Para finalizar, a resposta é $x_1 = 11.57$ e $x_2 = 6.21$. O valor da função objetivo é $L = 8 * x_1 + 21 * x_2 = 8 * 11.57 + 21 * 6.21 = 223.00$ (arredondando)

```
from pulp import *
import numpy as np
import matplotlib.pyplot as plt
prob=LpProblem("fazenda",LpMaximize)
X1=LpVariable('X1',0,None,None)
X2=LpVariable('X2',0,None,None)
prob+=8*X1+21*X2,"Lucro total"
prob+=11*X1+60*X2<=500,"restricao de area"
prob+=X1+2*X2<=24,"restricao de horas"
prob.writeLP("c:/python/lp03.lp")
prob.solve()
print("Status ",LpStatus[prob.status])
for v in prob.variables():
    print(v.name,' ',v.varValue)
print("Total ",value(prob.objective))
Status Optimal
print("Total ",value(prob.objective))
Status Optimal
X1 = 11.578947 X2 = 6.2105263
Total 223.05262829999998
```

O problema Começa com uma variável (custo, lucro, vazão, produção ou qualquer outra coisa similar) que se quer maximizar ou minimizar. Continua com uma lista de restrições que são inequações que o modelo deve obedecer. Tais desigualdades devem ser lineares e elas modelam o ambiente onde o problema se insere. O objetivo da programação linear é mostrar o valor otimizado e sobretudo as condições que vão permitir alcançá-lo.

Um exemplo Existe uma fábrica de 4 produtos: A,B,C e D. Ela deseja maximizar o lucro de seus quatro produtos A, B, C e D. Para fabricá-los usam-se as máquinas F=Furadeira e P=Prensa e dois tipos de mão de obra O=Operário e S=Supervisor. As disponibilidades mensais das máquinas são $F = 80$, $P = 20$. Quanto aos trabalhadores a disponibilidade é $O = 120$ e $S = 160$ horas por mês. As necessidades de produção por milhar de produto são:

Insumo	A	B	C	D
Furadeira	5	4	8	9
Prensa	2	6	-	8
Operário	2	4	2	8
Supervisor	7	3	-	7

O setor comercial da empresa informa que os potenciais de venda e os lucros unitários de cada produto são

Produto	Potencial vendas	L. unitário
argolas	70	10
botões	60	8
colchetes	40	9
decotes	20	7

O problema pede as quantidades que deverão ser produzidas dos quatro produtos a fim de que o lucro seja maximizado. Chamemo-las de A, B, C, D. Da tabela de lucro, obtemos que $L = 10 \times A + 8 \times B + 9 \times C + 7 \times D$.

As restrições do problema são:
 $A \leq 70$; $B \leq 60$; $C \leq 40$; $D \leq 20$ e mais
 $5A + 4B + 8C + 9D \leq 80$
 $2A + 6B + 8D \leq 20$
 $2A + 4B + 2C + 8D \leq 120$
 $7A + 3B + 7D \leq 160$,
e A, B, C, D ≥ 0 .

Usando o algoritmo simplex, (por exemplo através do programa LP88), obtem-se como resposta: A = 10; B = 0; C = 3,75; D = 0; com lucro de L = 133,75.

Eis a programação em Python

```
from pulp import *
prob=LpProblem("folha",LpMaximize)
A=LpVariable('A',0,None,None)
```

```
B=LpVariable('B',0,None,None)
C=LpVariable('C',0,None,None)
D=LpVariable('D',0,None,None)
prob+=10*A+8*B+9*C+7*D,"Lucro total"
prob+=A<=70,"potencial de vendas de A"
prob+=B<=60,"potencial de vendas de B"
prob+=C<=40,"potencial de vendas de C"
prob+=D<=20,"potencial de vendas de D"
prob+=5*A+4*B+8*C+9*D<=80,'restricao furadeira'
prob+=2*A+6*B+8*D<=20,'restricao prensa'
prob+=2*A+4*B+2*C+8*D<=80,'restricao operario'
prob+=7*A+3*B+7*D<=160,'restricao supervisao'
prob.writeLP("c:/python/lp02.lp")
prob.solve()
print("Status ",LpStatus[prob.status])
for v in prob.variables():
    print(v.name,' ',v.varValue)
print("Total ",value(prob.objective))
```

>>> Status Optimal
A=10.0 B=0.0 C=3.75 D=0.0 Total=133.75

Eis a matriz entregue ao programa simplex ca-seiro para este problema

-10	-8	-9	-7	0	0	0	0	0	0	0	0	0	0	0
5	4	8	9	0	0	0	0	1	0	0	0	80		
2	6	0	8	0	0	0	0	1	0	0	20			
2	4	2	8	0	0	0	0	0	1	0	120			
7	3	7	0	0	0	0	0	0	0	0	160			
1	0	0	1	0	0	0	0	0	0	0	0	70		
0	1	0	0	0	1	0	0	0	0	0	0	60		
0	0	1	0	0	0	1	0	0	0	0	0	40		
0	0	0	1	0	0	0	1	0	0	0	0	20		

Para você fazer

Procure em um livro de Pesquisa Operacional (como por exemplo *Introdução à Pesquisa Operacional* de HILLIER, Frederick e LIEBERMAN, Gerald, ou *Investigacion de Operaciones - una introduccion*, de Hamdy Taha ou ainda *Programação Linear* de Edgard Lanzer, ou qualquer outro), um problema qualquer de PL. Transcreva-o, arme as variáveis e as equações e depois resolva-o usando o pacote PULP do Python. Entregue tudo junto com esta folha.



Programação Dinâmica

Para muitos problemas quando a força bruta resulta inaceitável, a técnica da programação dinâmica pode ser uma alternativa. Esta técnica permite construir algoritmos que resolvem problemas computacionais, sobretudo na área de otimização combinatória. Tais problemas precisam ter 2 características para poderem usar a técnica:

- subestrutura ótima: quando a solução ótima contém em seu interior soluções ótimas para subproblemas e
- superposição de problemas: ocorre quando o algoritmo de força bruta reexamina o mesmo problema muitas vezes.

Exemplos de problemas que podem ser resolvidos usando programação dinâmica: multiplicação de séries de matrizes, problema da mochila, otimização combinatória, etc. Os problemas desta folha são os problemas 67, 81 e 82 do projeto Euler.

Maior soma em pirâmide

Suponha uma matriz como

3
7 4
2 4 6
8 5 9 3

Precisa-se calcular a soma máxima começando no vértice superior e descendo sempre para os 2 números adjacentes. Neste caso é fácil ver que a soma máxima é 3+7+4+9=23. Se a matriz tiver 15 linhas (e 15 números na última linha), já serão 16384 possíveis caminhos: o computador ainda os percorre muito rápido. Já em uma matriz de 100 linhas, se seu programa testar um trilhão de caminhos por segundo, serão necessários vinte bilhões de anos para testar todas as possibilidades.

Começando no alto do triângulo acima (no 3), deve-se escolher ir para a direita ou a esquerda. Para decidir isto, precisa-se resolver 2 problemas que são:

7 4
2 4 e 4 6
8 5 9 5 9 3

Pode-se ir quebrando a série de problemas até a última linha, cuja resposta é evidente: é o número que está lá. Resolvido, pode-se retornar para cima, usando a regra

a
b c

$a + \max(b, c)$. Entretanto, ao quebrar o problema original em 2, e cada um deste em outros 2, fica-se com 4 problemas. (As pirâmides: 7,2,4; 4,4,6 se quebram em 2,8,5; 4,5,9 e em 4,5,9; 6,9,3). Note que há problemas que se repetem (4,5,9 no exemplo). Ora, eles vão ter o mesmo resultado não importando como se chegou a eles. Logo, basta calcular tais problemas uma vez, e aqui a programação dinâmica mostra seu valor.

o algoritmo Começando de baixo para cima e aplicando a regra $a + \max(b, c)$, fica

3
7 4
2 4 6
8 5 9 3

fica

3
7 4
10 13 15

e depois

3
20 19

E finalmente 23, que é a resposta correta.

o algoritmo Neste caso o algoritmo é (origem=1):

```
1: leia(mat)
2: i ← número de linhas da mat
3: enquanto (i > 0)
4:   j ← 1
5:   enquanto (j ≤ i)
6:     mat[i][j] ← mat[i][j]+max(mat[i+1][j],
7:                               mat[i+1][j+1])
8:   j++
9:   fim(enquanto)
10: i--
11: fim(enquanto)
```

Menor soma em matriz

Suponha a matriz

131 673 234 103 18
201 96 342 965 150
630 803 746 422 111
537 699 497 121 956
805 732 524 37 331

Começando no alto à esquerda (no 131) e podendo apenas descer 1 linha ou andar à direita uma coluna, qual a menor soma que se pode obter ao chegar no último elemento à direita (o 331) ? Neste problema é fácil ver que a mínima soma é 131+201+96+342+746+422+121+37+331=2427. Mas, numa matriz quadrada de ordem 80, fica impossível a força bruta (serão 9×10^{46} caminhos). Para usar a programação dinâmica, vai-se quebrar o problema em probleminhas e depois juntar tudo. O ponto chave, é que se você estiver no final do problema, digamos: (o 188 é o número da última linha e última coluna da matriz)

270 312
416 188

Ao sair do 270 você deve escolher à direita ou para baixo: A decisão pode ser vista em (500=188+312 e 604=188+416)

270 500
604 188

Desta maneira a menor soma desde 270 até 188 é 270+min(500,604) que é 270+500=770. Usando esse raciocínio em toda a matriz, tem-se o algoritmo (origem=1):

```
1: leia mat
2: i ← ordem da matriz
3: enquanto (i > 0)
4:   mat[ord][i] ← mat[ord][i]+mat[ord][i+1]
5:   mat[i][ord] ← mat[i][ord]+mat[i+1][ord]
6:   i ← i - 1
7: fim(enquanto)
8: i ← ordem - 1
9: enquanto (i > 0)
10:   j ← ordem - 1
11:   enquanto (j > 0)
12:     mat[i][j] ← mat[i][j]+min(mat[i+1][j],
13:                               mat[i+1][j+1])
14:   j ← j - 1
15:   fim(enquanto)
16: i ← i - 1
17: escreva mat[1][1]
```

A mínima soma na matriz

Saindo de qualquer elemento da primeira coluna e chegando em qualquer elemento da última coluna, qual a soma mínima, podendo subir, descer ou andar à direita ? Por exemplo, em

131 673 234 103 18
201 96 342 965 150
630 803 746 422 111
537 699 497 121 956
805 732 524 37 331

é fácil ver que a menor soma é 201+96+342+234+103+18=994. No entanto, em uma matriz de ordem 80, não é mais tão fácil achar a resposta. A solução parte da premissa de que como não é permitido andar à esquerda, pode-se quebrar a matriz em colunas e resolver cada uma. Assim, o modelo da programação dinâmica continua válido.

o algoritmo (origem=1):

```
1: leia mat
2: int sol[ord]
3: sol ← última coluna de mat
4: i ← ord - 1
5: enquanto (i > 0)
6:   sol[i] ← sol[i]+mat[i][i]
7:   j ← 2
8:   enquanto (j ≤ ord)
9:     sol[j] ← min(sol[j-1]+mat[j][i],
10:                 sol[j]+mat[j][i])
11:   j ← j + 1
12:   fim(enquanto)
13:   j ← ord - 1
14:   enquanto (j > 0)
15:     sol[j] ← min(sol[j], sol[j+1]+mat[j][i])
16:   j ← j - 1
17:   fim(enquanto)
18: imprima o menor valor de sol
```

Exemplos

Depois que você criar e testar os algoritmos com as matrizes acima, e antes de executar esta folha às veras, teste-os com 3 exemplos cujos nomes são EXEF00E1, EXEF00E2 e EXEF00E3 cujas respostas esperadas são: 73218, 42729 e 27535. A propósito, as respostas corretas da sua instância darão números parecidos a estes. Peça estes arquivos para o professor.

📖 Para você fazer

Os dados referentes aos problemas estão em arquivos magnéticos que deverão estar publicados no portal na aula referente ao dia em que a folha for distribuída, ou deverão ser disponibilizados aos alunos em qualquer outro meio magnético.

1. Exercício da máxima soma em pirâmide. Use o arquivo

XXXXF01E1

que é uma matriz de ordem 100 parcialmente zerada (para simular uma pirâmide) e que contém em cada linha, 100 números menores do que 1000 e separados por um ou mais brancos. A linha ao final contém os caracteres CR+LF.

2. Exercício da mínima soma em matriz da esquerda superior até a direita inferior. Use o arquivo

XXXXF01E2

que é uma matriz de ordem 80 que contém em cada linha, 80 números menores do que 1000 e separados por um ou mais brancos. A linha ao final contém os caracteres CR+LF.

3. Exercício da mínima soma em matriz da esquerda até a direita. Use o arquivo

XXXXF01E3

que é uma matriz de ordem 80 que contém em cada linha, 80 números menores do que 1000 e separados por um ou mais brancos. A linha ao final contém os caracteres CR+LF.

1	2	3



- 1 - /

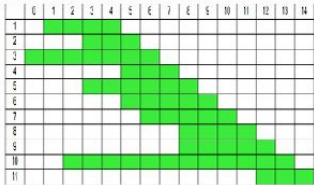
Seleção de atividades

Este é um problema comum em logística no qual há um recurso que deve ser compartilhado em diversas atividades. Tipicamente o recurso é escasso, pelo que diversas atividades deverão ser descartadas. Por exemplo: há um único laboratório na escola e muitas aulas querem utilizá-lo. Ou há poucos auditórios e diversas apresentações querem usá-lo. Ou há uma única pista de aterrissagem num aeródromo e diversos voos chegando querem pousar.

Para formalizar o problema: o recurso é único e enquanto uma atividade estiver usando-o nenhuma outra o pode fazer. Cada atividade tem um horário de início e de fim. O objetivo do problema é escolher um conjunto de atividades que maximize o uso do recurso (ou em outras palavras que minimize o tempo em que o recurso fica sem uso). Problema e imagens retiradas de <http://www.each.usp.br/digiampietri/SIN5013/13-algoritmosGulosos.pdf>. Há diversas estratégias aqui. Vamos usar um caso prático. Suponhamos 11 atividades em 14 unidades de tempo. A saber

- 1: começa em 1 e demora 4 unidades.
- 2: começa em 3 e demora 3.
- 3: começa em 0 e demora 7.
- 4: começa em 5 e demora 3.
- 5: começa em 3 e demora 6.
- 6: começa em 5 e demora 5.
- 7: começa em 6 e demora 5.
- 8: começa em 8 e demora 4.
- 9: começa em 8 e demora 5.
- 10: começa em 2 e demora 12.
- 11: começa em 12 e demora 3.

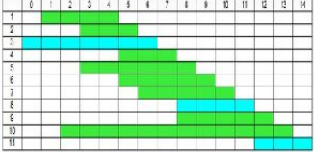
graficamente tem-se



Para uma possível solução pode usar:

Escolher as atividades que começam primeiro

Esta solução seria



Não foi muito bem, pois escolheu as atividades 3,8 e 11, quando uma solução melhor seria escolher 1,4,8 e 11.

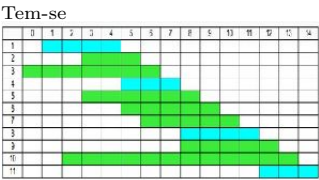
Escolher as atividades que demoram menos

Agora



Também não foi a melhor: escolheu 2,8 e 11, quando poderia ter escolhido 1,4,8 e 11.

Escolher atividades que terminam primeiro



Agora sim, esta solução é ótima. Agora pode-se desenvolver um algoritmo guloso para este problema:

- Ordene a lista de atividades por horário de termino.
- A cada iteração veja se a atividade atual pode ser incluída.
- sim ou não: a decisão é binária.

A estrutura de dados neste caso é

inicio = [1,3,0,5,3,5, 6, 8, 8, 2,12]
fim = [4,5,6,7,8,9,10,11,12,13,14]

para obter a duração, basta fazer (neste exemplo e em python):

```
def dura():  
    inicio = [1,3,0,5,3,5,6,8,8,2,12]  
    fim = [4,5,6,7,8,9,10,11,12,13,14]  
    dur=[0]*len(inicio)  
    for i in range(len(inicio)):  
        dur[i]=fim[i]-inicio[i]  
    print(dur)  
dura()
```

Para você fazer

- A seguir uma sequência de cirurgias previstas para ocorrerem em um único dia para um único centro cirúrgico. Cada cirurgia tem um horário de início e uma duração prevista. Você deve – usando o algoritmo guloso acima descrito – descobrir quais as cirurgias possíveis que otimizam o CC. Descubra 2 coisas:
1. Quantas cirurgias podem ser feitas ?
2. Quantos minutos do dia (máximo 1440) o CC ficará ocupado ?

cir.	hh:mm de início	hh:mm de duração
1	11 45	3 35
2	16 45	3 40
3	11 30	1 35
4	12 45	1 40
5	4 15	2 45
6	8 0	2 0
7	3 45	2 45
8	13 30	3 40
9	4 15	1 40
10	20 0	3 10
11	3 15	0 45
12	8 45	1 20
13	18 30	2 0
14	2 15	2 0
15	19 45	2 40
16	12 15	2 40
17	1 15	2 15
18	6 15	3 20
19	7 15	3 5
20	16 30	1 55
21	0 15	2 30
22	13 45	2 10
23	19 0	2 5
24	18 45	1 50
25	4 0	2 0
26	18 15	2 40
27	0 45	2 40
28	18 45	3 45
29	0 45	1 25
30	5 30	2 35

Responda aqui

quantas cirurgias ?	minutos de uso do CC

2. Agora, é um forno industrial em uma padaria e confeitaria. Precisa-se preparar o uso mais otimizado possível de um forno industrial em um período de 6 horas (360 minutos). Há muitas fornadas candidatas a usarem o forno e deve-se escolher o conjunto ótimo. Cada guloseima apresenta o minuto em que estará disponível para assar e a duração em minutos. Use o mesmo algoritmo.

f.	minuto em que fica pronto	duração
1	199	10
2	288	16
3	296	20
4	93	15
5	208	11
6	147	9
7	102	19
8	297	14
9	279	18
10	314	17
11	161	14
12	32	15
13	234	12
14	110	12
15	193	16
16	199	10
17	199	18
18	173	11
19	31	16
20	224	15
21	17	19
22	138	9
23	285	15
24	211	14
25	175	20

quantas fornadas ?	minutos de uso do forno industrial



Mecanismos de alocação

First Fit

Este algoritmo percorre a lista de áreas livres disponíveis na memória, até encontrar o primeiro espaço livre que acomode a requisição de memória que foi efetuada. Supondo que uma requisição de 120 inteiros tenha sido feita, este algoritmo entregaria a primeira área livre encontrada que fosse maior ou igual a 120. A sigla FIRST-FIT pode ser traduzida por o "primeiro que cabe". A principal vantagem dessa estratégia é a rapidez com que as solicitações são atendidas. Como desvantagem pode-se apontar o caráter randômico das alocações, o que deixa ao acaso o mapa geral de alocações.

Best Fit Este algoritmo percorre toda a lista de áreas livres fazendo a alocação pedida naquela área que minimize a sobra de espaço (a quantidade de inteiros perdidos ao final do bloco alocado). Por exemplo, digamos que uma requisição de 80 inteiros tenha sido feita, e ao percorrer a lista de áreas livres tenham sido encontradas as seguintes: 100, 280, 90, 500 e 320. Este algoritmo escolheria a terceira área, que é a que menor área desocupada deixaria ao final (apenas 10 inteiros). A sigla BEST-FIT pode ser traduzida por "o menor que cabe". A vantagem dessa estratégia é a conservação de blocos grandes. Por outro lado, ele tem como desvantagem a necessidade de pesquisar toda a lista a cada alocação, além de fragmentar a memória em muitos blocos pequenos, cada um deles podendo ter a sua sobra ao final. Este esquema privilegia as grandes alocações, no sentido de que requisições grandes tem boa chance de serem atendidas.

Worst Fit Este algoritmo percorre toda a lista de áreas livres e aloca a maior área que tenha sido encontrada na cadeia de áreas livres e que comporte a requisição de espaço solicitada. Usando o mesmo exemplo do caso anterior (solicitando-se 80, e comportando a lista os valores 100, 280, 90, 500 e 320) seria alocada a área de 500 inteiros, que passaria a contar agora apenas com 420 inteiros. A sigla WORST-FIT pode ser traduzida por "o pior que cabe". A vantagem desta estratégia é que ela não gera blocos muito pequenos, mas em compensação ela tende a fragmentar os blocos grandes, além é claro, de ter que percorrer toda a cadeia de áreas livres para achar "a pior".

Variantes Como toda algoritmo associado a uma estrutura de dados, as estratégias acima comportam diversas modificações. O objetivo sempre é encontrar um ponto de equilíbrio entre objetivos opostos quais sejam: minimizar o tempo gasto na alocação e maximizar o uso da memória. As variantes mais conhecidas são:

- Existência de um valor mínimo de alocação. Sempre que a diferença entre o valor pedido na alocação e o tamanho do bloco selecionado para atender aquela alocação for menor que um valor fixo, todo o bloco será alocado. Essa modificação, se de um lado desperdiça certas áreas ao final das alocações, por outro lado mantém a lista de blocos livres pequena - ou pelo menos, menor do que seria sem esse conceito.
- Ajuste de tamanho. Antes de se proceder ao atendimento da solicitação de alocação, o tamanho pedido é arredondado - para cima - para um valor múltiplo de alguma constante. Essa estratégia também garante a minimização de pequenos pedaços ao final dos blocos alocados.
- Encontro imediato. Nos algoritmos de BEST-FIT ou WORST-FIT, sempre que na pesquisa é encontrado um bloco com tamanho igual ao pedido (ou com diferença menor que uma dada constante) o bloco é imediatamente alocado, sem ter que se proceder à toda pesquisa na lista de blocos livres.
- Segregação de memória. Os blocos são previamente criados e posteriormente não são divididos. Por exemplo, cria-se uma coleção de blocos de 100 inteiros, outra de 500, outra de 1000 e assim por diante.

Estratégias de alocação: exemplo prático Seja o problema do empacotamento no supermercado: Existe uma compra de objetos cujos pesos variam entre 1Kg e 10Kg. Cada sacola de mercado agüenta 15 Kg, e obviamente há um interesse do mercado em minimizar o uso de sacolas. Lembre que: esta-se abstraindo a questão da dimensão da compra...

Obviamente este problema se aplica a objetos em containers, pessoas em elevadores, alunos em vans, etc etc

A solução ótima deste problema tem complexidade ... maior do que 2^n , mas existem algoritmos mais rápidos e que são satisfatórios. Por exemplo, Suponha uma sequência de M pacotes pesando $\frac{S}{2} - \epsilon$, seguida de outra sequência de M pacotes pesando $\frac{S}{2} + \epsilon$ (S =limite da sacola e ϵ =um pequeno valor, por exemplo 0,01 Kg). Está claro que a solução ótima é usar M sacolas. Mas, na sequência como foi dada, cada dois pacotes da primeira metade ocuparão uma sacola, e depois, cada pacote da segunda metade usará 1 sacola.

Um exemplo Seja uma instância do problema: Compra: 2, 5, 4, 7, 1, 3 e 8 Kg. Uma possível resposta para sacolas com limite máximo de 10Kg é $S_1=8$ e 2 Kg; $S_2=3$ e 7Kg e $S_3=5$, 1 e 4 Kg.

Existem 2 versões deste algoritmo, chamadas ON-LINE e OFF-LINE. Na primeira cada pacote deve ser colocado na sua sacola definitiva, ANTES que chegue o segundo pacote.

Na versão OFF-LINE, todos os pacotes estão disponíveis antes de começar a distribuição. **Estratégia 1: NEXT FIT** É a mais simples: ao chegar um pacote, verifica-se se ele cabe na sacola atual. Se couber, vai, senão fecha a sacola atual e começa outra. Esta estratégia nunca usa mais do que $2 \times X$ sacolas (X =número ótimo de sacolas). Questão: quantas sacolas esta estratégia usaria no exemplo acima ? $R = \frac{4 \cdot X + 1}{3}$.

Estratégia 2: FIRST FIT Verifica todas as sacolas já usadas para ver se o pacote cabe em alguma. A primeira possível é a escolhida. Se nenhuma serve, começa-se nova sacola. Esta estratégia nunca usa mais do que $1.7 \times X$ sacolas. Questão: idem. $R = \frac{4 \cdot X + 1}{3}$.

Estratégia 3: BEST FIT Verifica todas as sacolas abertas. Escolhe a que tem o menor espaço onde este pacote cabe. Nunca mais do que $1.7 \times X$, mas este número é quase nunca alcançado frequentemente. Questão: idem. $R = \frac{4 \cdot X + 1}{3}$.

Estratégia 4: WORST FIT Verifica todas as sacolas abertas. Escolhe a que tem o maior espaço onde o pacote cabe.

Estratégia 5: Versão OFFLINE Os pacotes devem ser ordenados em ordem decrescente. Daí aplica-se a estratégia FIRST ou BEST. Ambas dão o mesmo resultado, logo usar-se-á aqui a FIRST. O desempenho aqui nunca é maior do que $\frac{4 \cdot X + 1}{3}$ sacolas.

Acompanhe o exemplo Seja o seguinte conjunto de compras com o limite da sacola=15Kg.

1	3	3	8	3	10	4	6	8	3
7	2	7	10	2	7	5	6	4	6

Que dará o seguinte resultado em número de sacolas:

	NEXT	FIRST	BEST	WORST	OFF
1	1,3,3,8	1,3,3,8	1,3,3,8	1,3,3,8	10,5
2	3,10	3,10,2	3,10,2	3,10	10,4,1
3	4,6	4,6,3,2	4,6,3,2	4,6	8,7
4	8,3	8,7	8,7	8,3	8,7
5	7,2	7,7	7,7	7,2,6	7,6,2
6	7	10,5	10,5	7,2,4	6,6,3
7	10,2	6,4	6,4	10	4,3,3,3,2
8	7,5	6	6	7,5	
9	6,4			6	
10	6				

Um exemplo completo para você ver Seja o seguinte conjunto de compras com limite da sacola=15Kg)

9	7	3	8	1	2	5	1	10	5
2	3	8	1	9	9	4	1	2	3

Veja:

Para o método NEXTFIT, a quantidade de sacolas é: **9**
Para o método FIRSTFIT, a sacola número 3 terá peso: **15**
Para o método BESTFIT, a primeira sacola terá: **4 itens**
Para o método WORSTFIT, a última sacola terá peso: **11**
Para o método OFFFIRSTFIT, a sacola mais CHEIA terá: **4 itens**.
Obs: mais CHEIA significa com mais itens (não necess. a mais pesada)

👉 - Para você fazer

Seja o seguinte conjunto de compras com limite da sacola=15Kg)

4	6	6	9	8	7	5	7	4
7	8	6	9	2	5	5	1	2

- Para o método **NEXTFIT**, a quantidade de sacolas é de _____ sacolas.
- Para o método **FIRSTFIT**, a sacola número 2 terá peso de _____ kilos.
- Para o método **BESTFIT**, a primeira sacola terá _____ itens.
- Para o método **WORSTFIT**, a última sacola terá peso de _____ kilos.
- Para o método **OFFFIRSTFIT**, a sacola mais CHEIA terá _____ itens.

Obs: 1. mais CHEIA significa com mais itens (não necessariamente a mais pesada)
2. Se houver empate de tudo, pegue a sacola de menor número



Alocação em um teatro

- PARA ENDEREÇAMENTO DESTA MEMÓRIA, A ORIGEM DOS INDICES É ZERO

Imagine um teatro formado por 120 lugares. Um vetor de 120 inteiros é utilizado para controlar a venda de ingressos.

Os valores que fazem parte deste vetor são:

-2 indica que aquela poltrona está livre

-1 indica que a poltrona esta ocupada e foi vendida para um espectador solitário (isolado), isto é, que não faz parte de nenhum grupo.

0 a 119 indica que a poltrona foi vendida para um participante de um grupo qualquer. No local correspondente à poltrona no vetor, indica-se em que poltrona esta o próximo elemento do grupo. No elemento correspondente a esta próxima poltrona, está a poltrona do próximo elemento, e assim por diante. Este encadeamento não tem terminador. Sabe-se que um grupo terminou quando um número qualquer aparece pela segunda vez no encadeamento.

Vejamos no exemplo a seguir com apenas 20 poltronas

	0	1	2	3	4	5	6	7	8	9
0	1	0	17	8	-1	-2	-2	3	7	18
1	-2	-1	-2	-1	9	19	2	15	16	

Percebe-se neste teatro (vetor de 0..19), poltronas vazias em: 5, 6, 10 e 12. Ocupantes solitários, os há em: 4, 11, 13 e 14. Quanto aos grupos, há 5 grupos com 2, 3, 2, 2 e 3 elementos.

grupo 1 ocupa as cadeiras 1 e 0

grupo 2 17 e 2

grupo 3 3, 8 e 7

grupo 4 9, 18 e 15

grupo 5 16 e 19

PS: note o critério de numeração dos grupos. VALE A MENOR CADEIRA.

No exercício a seguir, dá-se um teatro com 120 lugares e se pede:

1. Quantas cadeiras estão livres ?
2. Quantas pessoas solitárias estão sentadas ?
3. Quantas pessoas de grupos estão sentadas ?
4. Quantos grupos há no teatro ?
5. Numerando os grupos considerando a menor poltrona que cada um tem, pergunta-se quantos elementos tem o grupo número 5 ?
6. Idem para o grupo de número 7 ?
7. Quantos elementos tem o maior grupo ?
8. Qual o primeiro grupo que poderia ter 4 pessoas gordas (que precisavam usar 2 cadeiras: uma originalmente alocada ao grupo e mais uma poltrona livre contígua) ?

Acompanhe a solução destas perguntas para o caso exemplo. Seja o teatro

	0	1	2	3	4	5	6	7	8	9
0	14	97	-2	16	76	-1	118	-1	-1	113
1	-2	-1	72	98	58	-2	12	55	-2	-1
2	-1	94	-2	110	0	103	37	93	-2	114
3	75	-2	21	26	86	29	101	99	109	69
4	-2	-1	66	-2	-2	-1	-2	115	53	-1
5	-2	83	-2	87	107	1	63	-1	64	108
6	-1	17	100	56	59	-2	119	-1	-2	80
7	-2	-1	4	-1	-1	6	42	-2	-1	-1
8	38	-2	116	84	117	48	34	51	30	-1
9	-2	35	47	82	39	-2	-2	23	27	36
10	91	33	-2	32	105	24	-2	61	104	25
11	54	-2	3	13	62	88	9	85	92	112

Para efeito de vizinhança, considere que se trata de um VETOR. Os elementos 0 e 119 só tem 1 vizinho. Todos os demais elementos do vetor tem 2 vizinhos.

O teatro acima tem os seguintes grupos:

- 1 58 64 59 108 104 105 24 0 14
- 2 23 110 54 107 61 17 55 1 97
- 3 112 3 16 12 72 4 76 42 66 119
- 4 6 118 92 47 115 88 30 75
- 5 82 116 9 113 13 98 27 93
- 6 109 25 103 32 21 94 39 69 80 38
- 7 33 26 37 99 36 101
- 8 29 114 62 100 91 35
- 9 34 86
- 10 51 83 84 117 85 48 53 87

11 56 63

Respostas às perguntas:

1. Quantas cadeiras estão livres: **24**
2. Quantas pessoas solitárias estão sentadas: **18**
3. Quantas pessoas de grupos estão sentadas: **78**
4. Quantos grupos há no teatro: **11**
5. Quantos elementos tem o quinto grupo: **8**
6. Idem para o sétimo grupo: **6**
7. Quantos elementos tem o maior grupo: **10**
8. Qual o primeiro grupo que poderia ter 4 pessoas gordas: **2**

Obs: atente que o primeiro grupo tem apenas 3 lugares vagos ao lado (cadeiras 64, 105 e 14). O grupo 2 tem 6 lugares vagos (cadeiras 23, 110, 107, 17, 1 e 97). Logo a resposta correta a pergunta 8 é: grupo 2

Para você fazer

Seja um teatro com as mesmas considerações feitas acima:

	0	1	2	3	4	5	6	7	8	9
0	30	-1	52	-1	34	-2	10	40	-1	55
1	50	-2	112	64	-2	44	32	-1	-2	28
2	111	-1	61	9	69	35	-2	-1	24	58
3	83	95	29	23	15	75	-1	85	93	71
4	105	12	53	78	39	-2	109	41	113	106
5	48	-1	116	80	70	49	37	90	54	114
6	-1	6	47	2	86	-2	-1	7	22	104
7	118	103	25	-2	-2	100	-2	-1	56	-1
8	97	87	117	98	72	43	16	20	68	-2
9	19	81	107	57	13	119	-2	0	31	63
10	101	82	-2	92	38	67	46	4	-2	59
11	-2	62	91	88	33	-2	99	84	94	42

Perguntas:

Quantas cadeiras estão livres: R: _____.

Quantas pessoas solitárias estão sentadas: R: _____.

Quantas pessoas de grupos estão sentadas: R: _____.

Quantos grupos há no teatro: R: _____.

Quantos elementos tem o quinto grupo: R: _____.

Idem para o sétimo grupo: R: _____.

Quantos elementos tem o maior grupo: R: _____.

Qual o primeiro grupo que poderia ter 4 pessoas gordas: (ter -2 ao lado de 4 cadeiras) R: _____.

Desafios Este exercício apresenta diversos desafios. Imagine um ambiente de programação (por exemplo o VG) no qual existe a variável global TEATRO na forma de um vetor de 120 inteiros corretamente preenchido de acordo com as especificações aqui listadas. O desafio é construir programas que:

1. Ache a quantidade de cadeiras livres.
2. Ache a quantidade de grupos existentes no teatro.
3. A quantidade de elementos por grupo

Para testar:

-2 85 39 -2 45 -2 -1 71 104 20 -2 -1 101 -2 115 75 78 37 -2 64
25 -2 23 30 32 80 -2 -1 48 -1 36 77 93 -2 14 -2 69 106 88 113
55 90 67 -1 42 116 28 -2 81 8 97 12 119 -1 -1 2 -2 7 -1 -1
57 24 98 68 109 87 1 62 100 51 117 44 102 4 -1 83 -1 19 40 34
79 105 73 16 107 46 -1 60 114 63 96 50 110 91 66 89 52 70 65 -2
31 92 84 112 22 94 61 38 -2 95 49 -2 82 15 72 41 103 17 -2 9

Respostas: 16, 9 e 9 10 7 11 13 13 12 10 7



- 1 - /

Como descrever uma família? Usando um problema derivado de uma coisa que todos conhecemos (as relações familiares) seja a questão de construir uma estrutura de dados para armazenar informações sobre uma família, que pode ser descrita por meio de uma série de declarações: (<=R sinaliza RAIZ).

```
Adrian P=Ronald M=Gisele I=Juliano<=R Paulo C=Alberta I=Fabiano
Antonio C=Elen I=Carmem Reginato P=Gilson M=Sara
Ari P=Moacir M=Maria C=Miriam I=Hugo C=Carmem I=Gisele UI
Benito C=Silvia I=Moacir Robson P=Sandro M=Camila UI
Bento P=Gilson M=Sara I=Reginato Ronald P=Heitor M=Liz
Daniel P=Sandro M=Camila C=Ana C=Gisele I=Adrian
I=Miriam, I=Marcos Sandro C=Camila I=Daniel
Fabiano P=Paulo M=Alberta I=Fabio Xavier P=Juca M=Bianca I=Jane
Fabio P=Paulo M=Alberta I=Ana Ana P=Paulo M=Alberta UI; bianca;
camila
Gilson C=Sara I=Bento Carmem P=Antonio M=Elen
Heitor P=Juca M=Bianca C=Liz I=Ronald Celia P=Ronald M=Gisele UI; elen
I=Xavier Gisele P=Reginato M=Carmem; helena
Hugo P=Ari M=Miriam I=Liz Jane P=Juca M=Bianca I=Sheila
Jose C=Helena I=Maria Liz P=Ari M=Miriam UI
Juca C=Bianca I=Heitor Maria P=Jose M=Helena
Juliano P=Ronald M=Gisele I=Celia Miriam P=Daniel M=Ana; sara
Marcos P=Sandro M=Camila I=Robson Sheila P=Juca M=Bianca UI; silvia
Moacir P=Benito M=Silvia C=Maria I=Ari
```

A sequência de nomes para o exemplo (na tabela) é: adrian, antonio, ari, benito, bento, daniel, fabiano, fabio, gilson, heitor, hugo jose, juca, juliano, marcos, moacir, paulo, reginato, robson, ronald, sandro, xavier, alberta, ana, bianca, camila, carmem, celia, elen, gisele, helena, jane, liz, maria, miriam, sara, sheila, silvia.

Cada regra no formato “x P=y, M=z, C=k, I=w, l=t”, deve ser assim entendida: a pessoa “x” tem um pai que é o “y” e uma mãe que é “z”. Se “x” for homem, poderá haver o nome da mulher dele (que seria “k”). Se x for mulher a informação do cônjuge não aparece, embora se x está casado com k, obviamente k está casada com x (propriedade comutativa do casamento).

A informação I=w refere-se ao próximo irmão de “x”. Os irmãos estão ordenados por ordem alfabética machista (os homens vêm antes das mulheres). Para reconstituir a lista de irmãos completa, deve-se buscar a informação “l=” dos pais de “x”. A partir desta pessoa, deve-se seguir o encadeamento mostrado em “I=”, até ser encontrada a informação UI, que significa, “último irmão”.

Finalmente, como se viu, “l=t” significa que o primeiro filho de “x” é o “t”. Informações marcadas com “*****” devem ser ignoradas.

De posse destas informações, o aluno deve montar uma árvore genealógica, na qual a raiz da árvore deve ser desenhada no centro inferior da folha, e daí devem ser colocados os ascendentes do raiz, sempre colocando o pai à esquerda e a mãe à direita.

Agora, responda:

1. Quem são os bisavós paternos de Liz ?
: _____, _____, _____
2. Quem são os bisavos homens de Adrian ?
: _____, _____, _____

Construída a matriz (cujo começo aqui aparece...)

Num	Nome	Paí	Mãe	Mulher	PFilh	ProxI
1	Adrian	20	30			14
2	Antonio			29	27	
3	Ari	16	34	35	11	
4	Benito			38	16	
5	Bento	9	36			18
6	Daniel	21	26	24	35	15
7	Fabiano	17	23			8
8	Fabio	17	23			24
9	Gilson			36	5	

Responda às perguntas

Na coluna PAI, em que linha aparece o primeiro MAIOR que 20 ? R1:
Na coluna MÃE, em que linha aparece o último MAIOR que 35 ? R2:
Na coluna PRIMF, em que linha aparece o mais próximo de 16.9 ? R3:
Na coluna PROXI, em que linha aparece o mais próximo de 10.9 ? R4:

☞ Para você fazer

A RAIZ = Julio
a)Aleixo C=Sheila I=Marcelo
b)Alfredo P=Ari M=Vera C=Tatiana I=Bento
c)Anderson C=Elisa I=Gisele
d)Ari C=Vera I=Alfredo
e)Augusto C=Paula I=Jean
f)Benito C=Iara I=Mara
g)Bento P=Alfredo M=Tatiana C=Daniela I=Roberto
h)Daniel C=Celia I=Juliano
i)Gustavo P=Ricardo M=Mara C=Debora I=Sonia
j)Hugo C=Andreia I=Ricardo
k)Jean P=Augusto M=Paula C=Gisele I=Ruth

l)Juliano P=Daniel M=Celia I=Tatiana
m)Julio P=Nelson M=Ruth I=Vinicius
n)Marcelo P=Aleixo M=Sheila C=Keli I=Debora
o)Nelson P=Roberto M=Sonia C=Ruth I=Julio
p)Ricardo P=Hugo M=Andreia C=Mara I=Gustavo
q)Roberto P=Bento M=Daniela C=Sonia I=Nelson I=Rodrigo
r)Rodrigo P=Bento M=Daniela I=Nelma
s)Vinicius P=Nelson M=Ruth UI
t)Xavier C=Ana I=Keli
u)Debora P=Marcelo M=Keli
v)Gisele P=Anderson M=Elisa
w)Keli P=Xavier M=Ana
x)Mara P=Benito M=Iara
y)Nelma P=Bento M=Daniela UI
z)Ruth P=Jean M=Gisele
aa)Sonia P=Gustavo M=Debora
ab)Tatiana P=Daniel M=Celia UI

Encerrado o desenho, pede-se: Pergunta: Quais os nomes de bisavos paternos de Sonia

Resposta: _____, _____, _____ e _____

Pergunta: Quais os nomes de bisavos mulheres de Vinicius

Resposta: _____, _____, _____ e _____

Depois, deve-se preencher a tabela a seguir, que já está com a primeira coluna preenchida.

A numeração das colunas é: 1. O nome das pessoas, em ordem alfabética machista. 2. O endereço do pai, das pessoas que o tiverem. 3. O endereço da mãe, idem. 4. O endereço da mulher, apenas para os homens casados. 5. O endereço do primeiro filho, que é colocado apenas no pai. 6. O endereço do próximo irmão. Quando este for o último irmão, colocar “-1”.

	nome	pai	mãe	mul	pFilh	proxI
1	1-Aleixo					
2	2-Alfredo					
3	3-Anderson					
4	4-Ari					
5	5-Augusto					
6	6-Benito					
7	7-Bento					
8	8-Daniel					
9	9-Gustavo					
10	10-Hugo					
11	11-Jean					
12	12-Juliano					
13	13-Julio					
14	14-Marcelo					
15	15-Nelson					
16	16-Ricardo					
17	17-Roberto					
18	18-Rodrigo					
19	19-Vinicius					
20	20-Xavier					
21	21-Ana					
22	22-Andreia					
23	23-Celia					
24	24-Daniela					
25	25-Debora					
26	26-Elisa					
27	27-Gisele					
28	28-Iara					
29	29-Keli					
30	30-Mara					
31	31-Nelma					
32	32-Paula					
33	33-Ruth					
34	34-Sheila					
35	35-Sonia					
36	36-Tatiana					
37	37-Vera					
38						
39						
40						

A partir da tabela preenchida, podem ser respondidas as outras questões do exercício, que são:

1. Na coluna “PAI”, em que linha aparece o primeiro número MAIOR que 17 ? R: _____
2. Na coluna “MÃE”, em que linha aparece o último número MAIOR que 33 ? R: _____
3. Na coluna “MULHER”, em que linha aparece o número mais próximo de 21.9 ? R: _____
4. Na coluna “PRIMEIRO FILHO”, em que linha aparece o número mais próximo de 27.9 ? R: _____



Alocação de área em disco sob WINDOWS (fat)

O algoritmo de alocação de área precisa conciliar algumas questões importantes e em certa medida contraditórias, a saber:

1. alocação encadeada, para permitir que arquivos não precisem estar em contiguidade física. Esta característica permite que um arquivo comece hoje, seja fechado e dentro de alguns dias, seja aberto para extensão, podendo crescer (ou diminuir) sem que haja necessidade de transferência física dos dados já gravados. Esta característica permite também um certo grau de (pseudo) paralelismo, através do qual dois ou mais processos podem estar gravando seus arquivos.
2. Eficiência de busca, tanto para buscar um dado quanto para alocar um bloco novo a um arquivo já existente, sem que o sistema perca muito tempo nessas operações. Isto implica em overhead mínimo possível.
3. Facilidade de acessar e alocar blocos, incluindo-se aqui a simplicidade necessária e suficiente para recuperar discos danificados e/ou arquivos excluídos por engano.

Surge agora o conceito de unidade de alocação: Trata-se do quantum de área em disco que é oferecido a qualquer processo sempre que há requisição de espaço. Este é o valor mínimo de alocação. Nunca um arquivo terá a si alocada uma área menor do que esta unidade (embora possa acabar não usando integralmente o que foi alocado). Os americanos chamam esta unidade de alocação de cluster, que foi traduzida para o português como grânulo.

Tipicamente um cluster pode ser de 2048 bytes ou de 1024 bytes. Clusters muito grandes causam fragmentação do disco (pelo desperdício de áreas não ocupadas em cada cluster) enquanto clusters pequenos implicam em grandes tabelas de alocação. Isto posto, e como regra geral, quanto menor o cluster mais eficiente é o sistema.

Como já descreveu Peter Norton, em seus livros, qualquer disco em Windows (fat) está organizado em 4 áreas (eventualmente discos rígidos tem uma quinta área que descreve o volume).

1. Registro de Boot: contém características físicas do disco e um pequeno programa de carga do núcleo do sistema operacional (SO) quando a máquina é ligada e este disco é o especificado para inicializar a máquina.
2. Área de diretório, que comporta diversas informações sobre cada um dos arquivos (nome, tamanho, data de atualização, atributos etc). Aqui interessam-nos duas informações: o nome do arquivo e o cluster em que ele se inicia.
3. Área de FAT (File allocation table). Trata-se de uma tabela que será explicada após ver-se a área 4.
4. Área de dados do disco. É a maior (mais de 95% do disco). Está dividida em clusters e cada cluster é numerado.

Agora pode-se voltar à definição da FAT. Ela faz a ligação entre os diversos clusters que compõem um arquivo.

Assim por exemplo, suponhamos que o arquivo AAAA ocupa 4 clusters, o 2, o 7, o 8 e o 10. No diretório, ocorre a entrada: AAAA começa em 2. Na segunda posição da FAT (que equivale ao cluster 2) o sistema coloca o número do próximo cluster do arquivo, no caso 7. Na 7 entrada da FAT está o número do próximo cluster que é o 8. Na oitava entrada consta 10 e finalmente na décima entrada tem-se um terminador de maneira a avisar que a cadeia acaba lá.

Vale ressaltar que o primeiro cluster do arquivo está identificado no diretório. A entrada da FAT correspondente a este cluster inicial contém o endereço do próximo cluster e assim por diante.

Neste exercício vai ser reproduzida a estratégia de alocação de áreas em disco do sistema operacional windows(fat), que veio sendo usada desde o longínquo 1981 no DOS, até as versões atuais do Windows, se bem que o esquema teve diversas atualizações, sobretudo devido aos discos enormes da atualidade. Para reproduzir a estratégia, considere:

Imagine um disco com 30 clusters, numerados de 1 a 30,

O tamanho de cada cluster é de 500 bytes,

Ocupa sempre os primeiros clusters livres,

Os clusters defeituosos, devem ser marcados com -2 no início do exercício e assim ficar até o final.

Informe ao final, qual a disposição da FAT

Significados: 0 = cluster livre; -1=terminador; -2=cluster defeituoso

Exemplo O enunciado é o mesmo. Suponha que os clusters defeituosos são:

CLUSTERS DEFEITUOSOS	3	4	12	23

Para cada ordem a seguir, será construída a FAT como sendo um vetor de 30 números

- 0 0 -2 -2 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
1. Cria arquivo A com 200 bytes
-1 0 -2 -2 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 2. Cria arquivo B com 1300 bytes
-1 5 -2 -2 6 -1 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 3. Cria arquivo C com 400 bytes
-1 5 -2 -2 6 -1 -1 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 4. Cria arquivo D com 800 bytes
-1 5 -2 -2 6 -1 -1 9 -1 0 0 -2 0 0 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 5. Expande arquivo C em 600 bytes

- 1 5 -2 -2 6 -1 10 9 -1 -1 0 -2 0 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
6. Expande arquivo A em 500 bytes
11 5 -2 -2 6 -1 10 9 -1 -1 -1 -2 0 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 7. Cria arquivo E com 800 bytes
11 5 -2 -2 6 -1 10 9 -1 -1 -1 -2 14 -1 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 8. Expande arquivo B em 100 bytes
11 5 -2 -2 6 -1 10 9 -1 -1 -1 -2 14 -1 0 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 9. Expande arquivo D em 1000 bytes
11 5 -2 -2 6 -1 10 9 15 -1 -1 -2 14 -1 16 -1 0 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 10. Expande arquivo B em 1100 bytes
11 5 -2 -2 6 17 10 9 15 -1 -1 -2 14 -1 16 -1 18 -1 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 11. Expande arquivo A em 600 bytes
11 5 -2 -2 6 17 10 9 15 -1 19 -2 14 -1 16 -1 18 -1 -1 0 0 0 -2 0 0 0 0 0 0 0 0 0 0 0
 12. Cria arquivo H com 1400 bytes
11 5 -2 -2 6 17 10 9 15 -1 19 -2 14 -1 16 -1 18 -1 -1 21 22 -1 -2 0 0 0 0 0 0 0 0 0 0 0
 13. Cria arquivo F com 1000 bytes
11 5 -2 -2 6 17 10 9 15 -1 19 -2 14 -1 16 -1 18 -1 -1 21 22 -1 -2 25 -1 0 0 0 0 0 0 0 0 0 0 0
 14. Expande arquivo C em 1100 bytes
11 5 -2 -2 6 17 10 9 15 26 19 -2 14 -1 16 -1 18 -1 -1 21 22 -1 -2 25 -1 27 28 -1 0 0 0
 15. Apaga arquivo D
11 5 -2 -2 6 17 10 0 0 26 19 -2 14 -1 0 0 18 -1 -1 21 22 -1 -2 25 -1 27 28 -1 0 0 0
 16. Apaga arquivo A
0 5 -2 -2 6 17 10 0 0 26 0 -2 14 -1 0 0 18 -1 0 21 22 -1 -2 25 -1 27 28 -1 0 0 0
 17. Cria arquivo A com 700 bytes
8 5 -2 -2 6 17 10 -1 0 26 0 -2 14 -1 0 0 18 -1 0 21 22 -1 -2 25 -1 27 28 -1 0 0 0
 18. Cria arquivo G com 200 bytes
8 5 -2 -2 6 17 10 -1 -1 26 0 -2 14 -1 0 0 18 -1 0 21 22 -1 -2 25 -1 27 28 -1 0 0 0

Para você fazer

CLUSTERS DEFEITUOSOS	2			

Eis as ordens a executar no disco:

- 1-Cria arquivo A com 500 bytes
- 2-Cria arquivo B com 100 bytes
- 3-Cria arquivo C com 800 bytes
- 4-Cria arquivo D com 600 bytes
- 5-Cria arquivo E com 1200 bytes
- 6-Expande (se existir) ou Cria (se nao) arquivo A em/com 1400 bytes
- 7-Apaga arquivo D
- 8-Expande (se existir) ou Cria (se nao) arquivo B em/com 200 bytes
- 9-Expande (se existir) ou Cria (se nao) arquivo A em/com 500 bytes
- 10-Cria arquivo F com 200 bytes
- 11-Expande (se existir) ou Cria (se nao) arquivo D em/com 500 bytes
- 12-Expande (se existir) ou Cria (se nao) arquivo A em/com 300 bytes
- 13-Expande (se existir) ou Cria (se nao) arquivo A em/com 300 bytes
- 14-Expande (se existir) ou Cria (se nao) arquivo B em/com 1100 bytes
- 15-Expande (se existir) ou Cria (se nao) arquivo C em/com 100 bytes
- 16-Expande (se existir) ou Cria (se nao) arquivo B em/com 100 bytes
- 17-Cria arquivo G com 1200 bytes
- 18-Cria arquivo H com 100 bytes

Para ajudar, eis um mapa do disco

Lista de arquivos		
Arquivo	Tamanho	Cluster inicial

FAT

Lembre que a FAT é um vetor de números.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Sua resposta Coloque aqui os valores das posições pedidas do vetor FAT. (apenas números).

4 _____, 11 _____, 17 _____, 19 _____.

Bibliografia: Desvendando o PC, Peter Norton



- 1 - /

Alocação de memória pelo interpretador LISP

Se você quer ir para Paris, precisa aprender francês. Se quer ir para a Inteligência Artificial, precisa aprender LISP

O LISP nasce como filho da comunidade de Inteligência Artificial. Na origem, 4 grupos de pessoas se inseriram na comunidade de IA: os lingüistas (na busca de um tradutor universal), os psicólogos (na tentativa de entender e estudar processos cerebrais humanos), matemáticos (a mecanização de aspectos da matemática, por exemplo a demonstração de teoremas) e os informáticos (que buscavam expandir os limites da ciência). O marco inicial é o encontro no Dartmouth College em 1956, que trouxe duas consequências importantes: a atribuição do nome (IA) e a possibilidade dos diferentes pesquisadores se conhecerem e terem suas pesquisas beneficiadas por uma interfecundação intelectual. A primeira constatação foi a de que linguagens existentes privilegiavam dados numéricos na forma de arrays. Assim, a busca foi a criação de ambientes que processassem símbolos na forma de listas. A primeira proposta foi IPL (Information Processing Language I, por Newel e Simon em 1956). Era um assemblão com suporte a listas. A IBM engajou-se no esforço, mas tendo gasto muito no projeto FORTRAN decidiu agregar-lhe capacidade de listas, ao invés de criar nova linguagem. Nasceu a FLPL (Fortran List Processing Language). Em 1958, McCarthy começou a pesquisar requisitos para uma linguagem simbólica. Foram eles: recursão, expressões condicionais, alocação dinâmica de listas, desalocação automática. O FORTRAN não tinha a menor possibilidade de atendê-las e assim McCarthy junto com Marvin Minsky desenvolveram o LISP. Buscava-se uma função Lisp Universal (como uma máquina de Turing Universal). A primeira versão (chamada LISP pura) era completamente funcional. Mais tarde, LISP foi sofrendo modificações e melhoramentos visando eliminar ineficiências e dificuldades de uso. Acabou por se tornar uma linguagem 100% adequada a qualquer tipo de resolução de problema. Por exemplo, o editor EMACS que é um padrão no mundo UNIX está feito em LISP. Inúmeros dialetos LISP apareceram, cada um queria apresentar a sua versão como sendo a melhor. Por exemplo, FranzLisp, ZetaLisp, LeLisp, MacLisp, Interlisp, Scheme, T, Nil, Xlisp, Autolisp etc. Finalmente, como maneira de facilitar a comunicação entre todos estes (e outros) ambientes, trabalhou-se na criação de um padrão que foi denominado Common Lisp. Como resultado o Common Lisp ficou grande e um tanto quanto heterogêneo, mas ele herda as melhores práticas de cada um de seus antecessores.

Listas Uma lista é uma composição de dados. Neste sentido um dado composto (lista) é o antônimo de átomo. Uma lista é um objeto fundamental em LISP. Tudo em LISP pode ser representado como uma lista. Inclusive funções são representadas como listas (são listas!) em LISP.

Listas são representadas de duas maneiras, a externa e a interna. Para o usuário de LISP apenas a representação externa interessa, mas para quem quer conhecer LISP e programá-lo, ambas são importantes.

A representação externa de uma lista começa com um "abre parênteses", segue por uma lista de átomos ou de listas separados por pelo menos um espaço e terminado por um fecha parênteses.

Neste exercício em particular sobre as possíveis maneiras de alocar memória, o que nos interessa mesmo é a representação interna.

Para entender a representação interna, precisa-se conhecer as funções fundamentais de LISP, que são: (CAR l_x), e que responde com o primeiro elemento de l_x ; (CDR l_y) que responde com a lista que sobra em l_y , após a retirada de seu CAR; (CONS a b) que constrói listas, como em (cons a nil) → (a) e (eval l_z) executa a expressão l_z . A partir delas é que a linguagem é desenvolvida.

Seja exemplos de funções lisp:

```
> (defun listasiguais (L1 L2) ; pergunta se duas listas são iguais
  (cond
    ((null L1) (null L2))
    ((null L2) nil)
    ((not (eql (car L1) (car L2))) nil)
    (t (listasiguais (cdr L1) (cdr L2)))))
```

LISTASIGUAIS

```
> (defun remove2 (ato lis)
  (cond
    ((null lis) nil)
    ((eql ato (car lis)) (remove2 ato (cdr lis)))
    (t (cons (car lis) (remove2 ato (cdr lis))))))
```

REMOVE2

```
> (defun contaels (lista)
  (cond
    ((null lista) 0)
    ((atom lista) 1)
    (t (+ (contaels (car lista)) (contaels (cdr lista)))))
  )
)
CONTAELS
```

Representação interna Os elementos de LISP são representados internamente por uma tabela de símbolos e por um "saco" de células CONS. As células CONS se unem com o auxílio da tabela de símbolos para formar as listas, que são o elemento fundamental em LISP.

Uma célula CONS é um conjunto de 2 endereços que apontam para a tabela de símbolos. Elementos da tabela, por sua vez apontam para células CONS que são encadeadas.

O primeiro endereço da célula CONS é chamado CAR, o segundo é chamado CDR. COMMON LISP possui os sinônimos "first" e "rest" mas parece que ninguém os usa.

Exercício 1 Acompanhe no quadro negro a representação da lista: LT=(TENHO 22 REAIS) Se imaginarmos que a tabela de símbolos começa no endereço 1000 (tamanho de cada entrada=12) e o "saco" de CONS começa em 2000, (tamanho de cada CONS=8), poderíamos ter a seguinte situação:

2000=LT	2000=1012,2008
1012=TENHO	2008=1024,2016
1024=22	2016=1036,1048
1036=REAIS	
1048=nil	

Exercício 2 Acompanhe no quadro negro o desenho e os endereços da lista L=(A (B C) D)

Exercício 3 Seja o seguinte o conteúdo de memória: (Obs: No endereço 1000 está a tabela de símbolos (tamanho=12) e no endereço 2000 está o "saco" de CONS (tamanho=8))

ENDE CONT	ENDE CAR CDR	ENDE CAR CDR
1000-0580	2000- 2008 2016	2088- 2096 2152
1012-0639	2008- 1060 1180	2096- 2104 2112
1024-1057	2016- 1048 2024	2104- 1144 1180
1036-3358	2024- 2032 2040	2112- 2120 2128
1048-3777	2032- 1168 1180	2120- 1012 1180
1060-4155	2040- 1024 2048	2128- 1036 2136
1072-4450	2048- 1156 2056	2136- 1084 2144
1084-4466	2056- 2064 2088	2144- 1000 1180
1096-JTYA	2064- 2072 2080	2152- 2160 1180
1108-KJFC	2072- 1108 1180	2160- 1096 2168
2000-L1	2080- 1072 1180	2168- 1132 1180
1132-OYGZ		
1144-QBDF		
1156-RZZH	Este exercício terá como resposta:	
1168-UVKJ	((4155) 3777 (UVKJ) 1057 RZZH ((KJFC) 4450)	
1180-nil	((QBDF) (0639) 3358 4466 0580) (JTYA OYGZ))	
1192-t		

Antes de continuar, vamos treinar achar o último elemento de uma lista
(a b c d) => d
((a b) (c d) (e f)) => (e f)
(a (b (c))) => (b (c))
(((a))) => ((a))

Para você fazer

Construa a lista L1, a partir dos dados abaixo:

tabela de símbolos		saco de CONS		
Endereço	Conteúdo	Endereço	CAR	CDR
1000	1604	2000	1048	2008
1012	2449	2008	2016	2120
1024	3168	2016	2024	2056
1036	3453	2024	1000	2032
1048	AICO	2032	2040	2048
1060	BTFA	2040	1036	1180
1072	CCOP	2048	1156	1180
1084	EIBH	2056	2064	2072
1096	FBJD	2064	1096	1180
1108	HRYM	2072	2080	2088
1120	ILAP	2080	1072	1180
2000	L1	2088	2096	2112
1144	RROF	2096	1120	2104
1156	XUSP	2104	1024	1180
1168	ZCNI	2112	1108	1180
1180	nil	2120	1084	2128
1192	t	2128	1168	2136
		2136	2144	2168
		2144	2152	2160
		2152	1144	1180
		2160	1012	1180
		2168	2176	1180
		2176	1060	1180

Depois de ter achado a representação externa (parentizada) da lista L1 acima fornecida, informe: O número de elementos de L1 é: _____.

O último elemento de L1 é: _____.



Simulação de pilhas

Suponha pilhas seqüenciais com as seguintes característi-
 cas:

```

1: estrutura PILHA
2: inteiro TOPO
3: caracter LET [5]
4: fim {estrutura} {A condição de inicialização é TOPO = 0}
1: função IP (PILHA P, caracter L)
2: se P.TOPO ≥ 5 então
3:     ...erro...
4: senão
5:     P.TOPO++
6:     P.LET[P.TOPO] ← L
7: fim se
8: fim {função}
1: caracter função EP (PILHA P)
2: caracter X
3: se P.TOPO ≤ 0 então
4:     devolva '*'
5: senão
6:     X ← P.LET[P.TOPO]
7:     P.TOPO-
8:     devolva X
9: fim se
10: fim função
    
```

Considere os seguintes exemplos: Sejam 2 pilhas 1 e 2:

a. IP (2,A)	P1= - - - - T=0,	P2= A - - - - T=1
b. IP (2,X)	P1= - - - - T=0,	P2= A X - - - T=2
c. IP (1,B)	P1= B - - - - T=1,	P2= A X - - - T=2
d. IP (2,Y)	P1= B - - - - T=1,	P2= A X Y - - T=3
e. EP (2)	P1= B - - - - T=1,	P2= A X Y - - T=2
f. IP (2,EP(1))	P1= B - - - - T=0,	P2= A X B - - T=3
g. IP (2,EP(1))	P1= B - - - - T=0,	P2= A X B * - T=4
h. IP (2,Y)	P1= B - - - - T=0,	P2= A X B * Y T=5
i. IP (2,W)	P1= B - - - - T=0,	P2= A X B * Y T=5
j. IP (1,EP(2))	P1= Y - - - - T=1,	P2= A X B * Y T=4
k. IP (1,K)	P1= Y K - - - T=2,	P2= A X B * Y T=4
l. IP (2,F)	P1= Y K F - - T=2,	P2= A X B * F T=5
m. IP (2,Z)	P1= Y K - - - T=2,	P2= A X B * F T=5
n. IP (1,EP(2))	P1= Y K F - - T=3,	P2= A X B * F T=4
o. IP (2,L)	P1= Y K F - - T=3,	P2= A X B * L T=5
p. IP (2,F)	P1= Y K F - - T=3,	P2= A X B * L T=5
q. EP (2)	P1= Y K F - - T=3,	P2= A X B * L T=4
r. EP (2)	P1= Y H F - - T=3,	P2= A X B * L T=3
s. IP (2,EP(1))	P1= Y K F - - T=2,	P2= A X B F L T=4
t. IP (2,EP(1))	P1= Y K F - - T=1,	P2= A X B F K T=5
u. IP (1,H)	P1= Y H F - - T=2,	P2= A X B F K T=5
v. IP (1,EP(2))	P1= Y H K - - T=3,	P2= A X B F K T=4
x. IP (1,EP(2))	P1= Y H K F - T=4,	P2= A X B F K T=3
y. IP (1,EP(2))	P1= Y H K F B T=5,	P2= A X B F K T=2
z. IP (2 S)	P1= Y H K F B T=5,	P2= A X S F K T=3

0 que é válido: P1=YHKFB, pois o topo é 5. P2=AXS, o topo é 3.

Para você fazer

Agora, simule a operação de 5 pilhas (1, 2, 3, 4 e 5):

1. IP(3,M)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
2. IP(2,U)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
3. EP(3)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
4. IP(4,EP(3))	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
5. IP(3,P)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
6. IP(4,L)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
7. IP(3,T)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
8. IP(4,N)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
9. IP(1,U)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
10. IP(4,L)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
11. EP(1)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
12. IP(1,EP(4))	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
13. IP(1,T)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
14. IP(2,EP(4))	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_

15. IP(2,EP(4))	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
16. IP(5,I)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
17. IP(5,EP(5))	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
18. EP(2)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
19. EP(1)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
20. IP(3,C)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
21. IP(3,D)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
22. IP(2,G)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
23. EP(4)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
24. IP(4,T)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
25. EP(3)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
26. EP(3)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
27. IP(1,B)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
28. IP(5,U)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
29. IP(3,B)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
30. IP(1,F)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
31. IP(3,M)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
32. IP(1,R)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
33. IP(5,I)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
34. IP(2,EP(1))	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
35. EP(1)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
36. IP(5,S)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
37. IP(2,M)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
38. IP(1,L)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
39. EP(3)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
40. EP(1)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
41. IP(4,Y)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
42. IP(2,J)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
43. IP(4,T)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
44. IP(2,T)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
45. EP(3)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
46. EP(1)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
47. IP(5,EP(3))	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
48. IP(2,0)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
49. IP(1,A)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_
50. IP(5,L)	1=_____T_ 2=_____T_ 3=_____T_ 4=_____T_ 5=_____T_

Para responder às questões abaixo, considere cada pilha como sendo um vetor linear de 5 posições. Ou seja, o exercício pode estar pedindo uma letra que está "morta". Assim P=4 2=___ está pedindo o segundo valor do vetor (pilha) 4, da esquerda para a direita e sem considerar o topo. Lembre que quando é feita uma exclusão a letra fica lá.

1. P=1	3_____	2. P=5	2_____	3. P=2	2_____	4. P=4	4_____	5. P=3	3_____
6. P=4	3_____	7. P=3	T_____	8. P=2	1_____	9. P=2	4_____	10. P=5	1_____



Pilhas em ASSEMBLY

Neste exercício serão examinadas implementações de pilhas seqüenciais e encadeadas em assembly (linguagem de baixo nível), ou seja usando apenas as funções básicas de manipulação de memória. Para poder estudar o funcionamento dos algoritmos usaremos a memória M, formada por 100 inteiros numerados (endereçados) de 00 a 99. A obtenção de memória é fornecida pela função OBTEMEMO que recebe como parâmetro a quantidade de inteiros pedida e devolve o endereço do primeiro deles (se houve sucesso) ou um número negativo, se não houve sucesso. Esta função trabalha baseada na variável global PPL que indica qual a Próxima Posição Livre. No início da simulação, PPL ← 0. M começa com zeros nas posições 0..79 e com 1..20 nas posições 80..99.

Alocação sequencial

```
1: inteiro função PILSCRI (inteiro TAM,QTD)
2: inteiro ONDE, TAPRE
3: ONDE ← OBTEMEMO ((TAPRE ← (TAM × QTD)) + 4)
4: se (ONDE < 0) então
5:   devolva -1 {não houve sucesso na alocação}
6: senão
7:   M[ONDE] ← ONDE + 4 {instala a BASE}
8:   M[ONDE+1] ← M[ONDE] + TAPRE {instala o LIMITE}
9:   M[ONDE+2] ← TAM {instala o TAMANHO}
10:  M[ONDE+3] ← M[ONDE] {instala o TOPO (= BASE, pilha vazia)}
11:  devolva ONDE
12: fim se
13: fim {função}

1: inteiro função PILSINS (inteiro P,E)
2: inteiro X
3: se M[P+3] + M[P+2] ≤ M[P+1] então
4:   para X de 0 até M[P+2]-1 faça
5:     M[X+M[P+3]] ← M[X+E]
6:   fim para
7:   M[P+3] ← M[P+3] + M[P+2]
8:   devolva 0
9: senão
10:  devolva -1
11: fim se
12: fim {função}

1: inteiro função PILSEXC (inteiro P)
2: se M[P+3] = M[P] então
3:   devolva -1
4: senão
5:   M[P+3] ← M[P+3] - M[P+2]
6:   devolva M[P+3]
7: fim se
8: fim {função}
```

Alocação encadeada

Para esta implementação, usar-se-á um descritor de pilha, contendo

&	topo	tamanho

```
1: inteiro função PILECRI (inteiro TAM)
2: inteiro ONDE
3: ONDE ← OBTEMEMO (2)
4: se (ONDE < 0) então
5:   devolva -1
6: senão
7:   M[ONDE] ← -1
8:   M[ONDE+1] ← TAM
9:   devolva ONDE
10: fim se
11: fim{função}

1: inteiro função PILEINS (inteiro P,E)
2: inteiro Y,ONDE
3: ONDE ← OBTEMEMO (M[P+1]+1)
4: se (ONDE < 0) então
5:   devolva -1
6: senão
7:   para Y de 0 até M[P+1]-1 faça
8:     M[ONDE+Y+1] ← M[Y+E]
9:   fim para
10:  M[ONDE] ← M[P]
11:  M[P] ← ONDE
12:  devolva 0
13: fim se
14: fim{função}

1: inteiro função PILEEXC (inteiro P)
2: se (M[P] = -1) então
3:   devolva -1
4: senão
5:   devolva M[P]+1
6:   M[P] ← M[M[P]]
7: fim se
8: fim{função}
```

Na prática

Suponha uma memória M, assim inicializada

M	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	1	2	3	4	5	6	7	8	9	10
9	11	12	13	14	15	16	17	18	19	20

Seja a seguinte seqüência de comandos de manipulação de pilhas

```
0:P1 = PILECRI( 4)
1:P2 = PILSCRI( 4, 2)
2:P3 = PILECRI( 2)
3:R2 = PILEINS(P3, 83)
4:R3 = PILEINS(P1, 86)
5:R4 = PILEINS(P3, 80)
6:R6 = PILEINS(P3, 80)
7:P4 = PILSCRI( 2, 2)
8:R12= PILSINS(P2, 85)
9:R27= PILEINS(P1, 80)
10:R19= PILEEXC(P1)
11:R24= PILEINS(P3, 83)
12:R8 = PILEINS(P3, 83)
13:R26= PILEINS(P1, 84)
14:R16= PILSEXC(P4)
15:R25= PILSINS(P2, 81)
16:R11= PILSINS(P2, 82)
17:R15= PILSEXC(P2)
18:R22= PILEINS(P3, 81)
19:R21= PILSINS(P2, 80)
```

Que dará origem à seguinte disposição de M

M	0	1	2	3	4	5	6	7	8	9
0	49	4	6	14	4	14	6	7	8	9
1	1	2	3	4	54	2	-1	4	5	-1
2	7	8	9	10	16	1	2	24	1	2
3	34	38	2	34					19	1
4	2	3	4	27	4	5	43	4	5	19
5	5	6	7	8	46	2	3			
6										
7										
8	1	2	3	4	5	6	7	8	9	10
9	11	12	13	14	15	16	17	18	19	20



Para você fazer

Inicialize M como no começo do exemplo. Depois, execute a seguinte seqüência de comandos de pilha. Ao final, informe as posições de M que são pedidas.

```
0:P1 =PILECRI( 4)
1:P2 =PILSCRI( 4, 2)
2:P3 =PILECRI( 3)
3:R2=PILEINS(P3, 83)
4:R3=PILEINS(P1, 84)
5:R4=PILEINS(P3, 82)
6:R6=PILEINS(P3, 80)
7:P4 =PILSCRI( 2, 2)
8:R15=PILSEXC(P2)
9:R19=PILEEXC(P1)
10:R21=PILSINS(P2, 81)
11:R9=PILEINS(P3, 82)
12:R11=PILSINS(P2, 83)
13:R14=PILEEXC(P3)
14:R17=PILEEXC(P3)
15:R27=PILEINS(P1, 80)
16:R12=PILSEXC(P2)
17:R10=PILEINS(P1, 80)
18:R16=PILSEXC(P4)
19:R26=PILEINS(P1, 82)
```

2	7	9	24	30

para quem quiser implementar

```
1: global M[0..79] ← 0;M[80..99] ← 1..20;PPL ← 0
2: inteiro função OBTEMEMO (inteiro X)
3: se PPL+X ≥ 99 então
4:   devolva -1
5: senão
6:   devolva PPL
7:   PPL ← PPL + X
8: fim se
```



Notação Polonesa Reversa

Seres humanos escrevem suas sentenças matemáticas com o uso de chaves, colchetes, parênteses e ainda por cima tem uma hierarquia de operações. Se isto não for usado, temos problemas. Por exemplo:

$5 + 4 * 3$ é igual a 17, ou é igual a 27 ???

Todo compilador precisa entender expressões aritméticas escritas por humanos, logo, todo compilador precisa entender tais sentenças. Será que não haveria uma maneira de se escrever estas sentenças, sem NENHUM parênteses, e SEM nenhuma hierarquia ?

Esta maneira foi inventada por **Jan Lukiziewicz**, em 1951 e passou a ser denominada NOTAÇÃO POLONESA. Agora, mediante a introdução de um caracteres separador de operandos, as sentenças podem ser escritas sem parênteses e sem ambigüidade.

Na notação polonesa, escrevem-se os operandos e (antes = préfixada ou depois (pósfixada) a operação que eles sofrerão.

Por exemplo, a expressão, $5 * (6 + 2) - 12/4$, ficaria 5,6,2,+,*,12,4,/,-. Para que qualquer consiga achar o valor de uma expressão com parênteses, ele sempre:

1. Precisa transformar a expressão em RPN
2. Calcular a RPN

Algoritmo de Normal para RPN

Este algoritmo recebe uma cadeia de caracteres (denominada Q) representando a expressão aritmética escrita de maneira convencional, eventualmente com o uso de parênteses. A saída é outra cadeia (denominada R) e que é equivalente a cadeia original (a Q) mas EXPRESSA EM RPN.

Para este algoritmo funcionar, consideraremos a seguinte escala de prioridades: 3=E, 2=*/, 1=+- . Quando duas operações de mesma prioridades existirem faremos da esquerda para a direita, a menos que haja parenteses.

```

1: cadeia função NORMAL2RPN (cadeia Q)
2: pilha PILHA
3: empilhe (PILHA,"(")
4: insira ")" ao final de Q
5: enquanto houver elementos em Q faça
6:   se é operando então
7:     coloque-o na resposta R
8:   senão
9:     se é "(" então
10:      empilhe-o
11:     senão
12:      se é o operador (*) então
13:        retire da pilha todos os operadores com prioridade igual ou maior a (*) colocando-os na resposta R
14:        empilhe (*)
15:      senão
16:        se é ")" então
17:          desempilhe da PILHA e coloque na resp R cada operador ate achar um "("
18:          remova o "("
19:        fim se
20:      fim se
21:    fim se
22:  fim se
23: fim enquanto
```

Mini chinês

Fazer chinês com a expressão $5 * (6 + 2) - 12/4$

Deve dar: 5,6,2,+,*,12,4,/,-

A propósito compare o resultado encontrado ao analisar uma expressão parecida com a acima: $5*(6+2)E12/4$, que dá como resposta 5,6,2+12E*4/.

Algoritmo de resolução de RPN

A entrada deste algoritmo é a cadeia já convertida pelo algoritmo anterior e a saída é um número real representando o resultado final numérico da expressão.

```

1: real função ResolveRPN (cadeia R)
2: pilha PILHA
3: real A,B
4: enquanto houver elementos em R faça
5:   se é um operando então
6:     empilhe-o
7:   senão
8:     se é um operador (*) então
9:       desempilhe B
10:      desempilhe A
11:      Calcule A (*) B
12:      empilhe o resultado
13:    fim se
14:  fim se
15: fim enquanto
```

O valor final na pilha é o resultado desejado

Mini Chinês

Faça um chinês com: $5 * (6 + 2) - 12/4$

Transformado em 5,6,2,+,*,12,4,/,-

E dará:

5,6,2; depois 5,8; depois 40,12,4; depois 40,3 e 37

Exemplos

1. $((4 - 8) + (((8 * 8) - 3) * 2)) / (3 + 5)$
2. $((6 * 1) + ((5 * ((9 - 8) + (2/1))) * (2 + 2))) + 5$
3. $((((6 * ((8 + 7) - (5 * (2E1)))) - (4/2)) - (6 - 1)) / 8)$
4. $(1 + 1) + (((((6 - 5) + ((3 - 3) + ((4 - 6) * (6/3)))) * (3 + 7)) + (5 + 1)) - 9)$
5. $((((8 - 1) * (6 + 9)) - (9 * 7)) - 1) / (5 - 3)$
6. $((9 - ((3 * 3) + (3/(4 * 2)))) + 1) + (8 - 3)$
7. $((0E2) + ((7 + (((6/2) + 9) - (6/3))) + 8)) - (4 + 8)$
8. $((((4 + 1) + (((9 + 7)/((9 * 5) * (7 + 6)))) + (1 - 1)) + 8)) - (5 - 4)) + (1 * 4)$

Respostas

1. $48 - 88 * 3 - 2 * +35 + /$, com resposta=14.75
2. $61 * 598 - 21/ + *22 + * + 5+$, com resposta=71.00
3. $687 + 521E * - * 42/ - 61 - -8/$, com resposta=2.88
4. $11 + 65 - 33 - 46 - 63/ * + + 37 + * 51 + + 9 - +$, com resposta=-31.00
5. $81 - 69 + * 97 * - 1 - 53 - /$, com resposta=20.50
6. $933 * 342 * / + - 1 + 83 - +$, com resposta=5.63
7. $02E762/9 + 63/ - + 8 + + 48 + -$, com resposta=13.00
8. $41 + 97 + 95 * 76 + * / 11 - + 8 + + 54 - - 14 * +$, com resposta=16.03

☞ - Para você fazer

Nos exercícios a seguir, converta a expressão para RPN e depois calcule o resultado correto. A letra *E* significa *elevado a* e indica a operação potência.

1. $((((8 * 2)/(4 + 6)) - (9 * 2)) * 1) - 3$
2. $(1 - 8) - ((3 - 2) + ((7 - 9) + (((8/4) * (1 - 9)) + 1)))$
3. $(((((2/1) - ((3 - 6)/(8 + 9))) * (1 + 2)) + (7 + 7)) - 2) + (2E2)$
4. $7 + ((6 * 1) + (1 * ((1 + 3) - ((5 * 3) + (8 - ((1 * 8) + (6 - 4))))))$

Para responder este exercício, você deve fazer duas coisas. Primeiro, deve localizar - na expressão RPN - qual o caracter pedido (usando o algoritmo 1). Depois, deve apresentar o resultado numérico dessa expressão RPN. (usando o algoritmo 2).

1. Caracter [9] _____. Resultado=_____.
2. Caracter [11] _____. Resultado=_____.
3. Caracter [14] _____. Resultado=_____.
4. Caracter [11] _____. Resultado=_____.



- 1 - /

O passeio do rato no laberinto

Considere o algoritmo ANDA descrito abaixo. Ele ajuda um ratinho a sair de um laberinto. O laberinto pode ser deste tipo:

```
x..xxx..x.x.xxx
x.x.x..xxxx..x.
xxxxxx.x..x.xxx
.....x.x..x.x..
xxxxxx.x..x.x..
..x....x..x.xxx
..xxxxxx..x.x.x
..x.....x.x.x
.....xxxxxxx.x
...xxx.....x
```

Nesta representação o caminho livre é representado por 'x', enquanto o obstáculo é representado por um ponto.

A saída do ratinho é sempre na posição 1,1, no alto, à esquerda do tabuleiro, e a chegada sempre é na posição mais inferior à direita.

No desenho a seguir, está representado o caminho que o rato deve seguir para escapar do laberinto. Considere a sequência dos números naturais 1234567890123...

```
1..xxx..x.x.xxx
2.x.x..5678..x.
345678.4..9.xxx
.....9.3..0.x..
xx3210.2..1.x..
..4....1..2.012
..567890..3.9.3
..x.....4.8.4
.....xxxx567.5
...xxx.....6
```

O laberinto sempre terá as dimensões 10,15. Entretanto para facilitar o algoritmo, trabalha-se com variáveis de 12,17, nas quais se construiu uma parede de pontos em torno do laberinto. Neste caso, agora, a saída é em 2,2 e a chegada é em 11,16.

O rato é metódico. Ele sempre tenta seguir as 4 direções nesta ordem: cima, direita, baixo e esquerda. Sempre que há caminho disponível nesta direção ela é seguida. Quando o rato encontra um beco sem saída, ele retorna até a última decisão.

A variável maze tem 12,17 e começa zerada. A medida em que o ratinho vai andando ele marca cada quadradinho andado como '1'. mark também é 12,17 contendo zeros (caminho) e uns (parede).

```
1: função ANDA (LABERINTO, LINHAS)
2: MAZE ← LABERINTO {cercado por 111111, acima, abaixo, a esquerda e a direita. 1 é parede, 0 é caminho}
3: MARK ← 0 {MARK tem as mesmas dimensões de MAZE. Começa zerado e cada andada é marcada com 1}
4: CRIAPILHA (P,3) {cria pilha P com elementos de comprimento=3}
5: MARK [ 2 ; 2 ] ← 1
6: CONTADOR ← 0
7: EMPILHA (P,(2,2,0))
8: enquanto (não vazia(P)) faça
9:   ALFA ← DESEMPILHA(P)
10:   CONTADOR++
11:   se (CONTADOR=10) então
12:     imprima ALFA {imprima na decima desempilhada}
13:   fim se
14:   I ← ALFA[1]
15:   J ← ALFA[2]
16:   MOV ← ALFA[3] + 1
17:   enquanto (MOV ≤ 4) faça
18:     G ← I + DIRE[MOV;1]
19:     H ← J + DIRE[MOV;2]
20:     se ((I = 11) ∧ (J = 16)) então
21:       ...ACABOU...
22:   fim se
23:   se ((MAZE[G ; H] = 0) ∧ (MARK[G ; H] = 0)) então
24:     MARK[G ; H] ← 1
25:     EMPILHA (P,(I,J,MOV))
26:     MOV ← 0
27:     I ← G
28:     J ← H
29:   fim se
30:   MOV++
31: fim enquanto
32: fim função
33: fim função
```

para efeito deste algoritmo considere a matriz DIRE com 4 linhas e 2 colunas e com os seguintes valores

-1	0
0	1
1	0
0	-1

Para o laberinto acima mostrado, os valores de ALFA são:

cont	mov linha	mov coluna	mov feito(1..4)
1	2	2	0
2	4	4	1
3	2	6	2
4	2	6	4
5	3	6	1
6	4	6	1
7	3	10	1
8	3	12	1
9	2	15	2
10	2	15	4
11	3	15	1
12	4	15	1
13	4	15	2
14	4	14	2
15	5	14	1
16	6	14	1
17	7	14	1

Para você fazer

Considere o seguinte laberinto (1=parede 0=passagem)

```
0 0 0 0 0 1 1 1 0 1 1 0 1 1 1
1 1 1 1 0 1 1 1 0 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 0 1 1 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 0 0 0 0 1 1 0 0 0
1 0 0 0 0 1 0 1 1 0 1 1 0 1 0
1 0 1 1 1 1 0 1 1 0 0 1 0 1 0
1 0 1 1 1 1 0 1 1 1 0 1 0 1 0
1 0 0 0 0 0 0 1 1 1 0 0 0 1 0
1 1 1 1 1 1 0 0 0 1 1 1 1 1 0
```

Considere DIRE como sendo a matriz 4 x 2 acima desenhada. A Pilha P tem tamanho suficientemente grande para o problema Considere o seguinte valor para LINHA: 10

Responda: Quantidade de passos do ratinho: _____.

alfa para linha pedida:

: _____ e _____.

1

Envoltória Convexa: algoritmo de Graham

Geometria Computacional Um capítulo importante da programação é a chamada geometria computacional. Ela é usada sempre que se precisa trabalhar com entidades geométricas. Jogos, diagramas de engenharia de software, redes de petri, quadros e mapas são apenas alguns exemplos onde é importante conhecer entes da geometria computacional. Este exercício, além disso ilustra a operação de uma pilha.

Pontos e produto cruzado

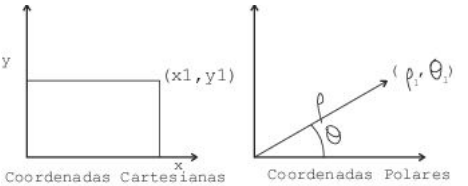
Sejam pontos distribuídos no espaço bi-dimensional. Cada ponto será identificado por uma dupla (x, y) . Dados 3 pontos $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ e $P_3(x_3, y_3)$, supondo que eles não sejam colineares, podemos imaginar a existência de 2 segmentos de reta: (p_0, p_1) e (p_1, p_2) . A pergunta que se busca responder aqui, é se, ao percorrer os pontos p_0, p_1 e p_2 nesta ordem, ao chegar ao ponto p_1 , vai-se para sua direita ou para a sua esquerda ao ir para p_2 . Para responder a essa questão, basta calcular o produto cruzado $(p_2 - p_0) \times (p_1 - p_0)$. Se este valor for negativo, fazemos uma volta à esquerda em p_1 . Se o valor for positivo, a volta é para a direita, e se for zero, os 3 pontos são colineares.

O cálculo de $p_1 \times p_2$ levado a suas coordenadas é igual a $x_1y_2 - x_2y_1$ que por sua vez é igual a $-p_2 \times p_1$.

Com as duas fórmulas acima, pode-se obter o produto cruzado dos 3 pontos: $((x_2 - x_0)(y_1 - y_0)) - ((x_1 - x_0)(y_2 - y_0))$

Coordenadas Polares

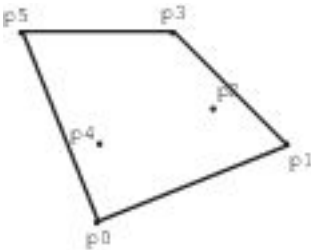
Embora a maneira mais comum de representar pontos no espaço bi-dimensional, seja através de duplas (x, y) , em certos momentos será necessário usar o sistema de coordenadas polares.



A conversão entre os dois sistemas é feita usando-se as seguintes relações: $\rho = \sqrt{x^2 + y^2}$ e $\theta = \arctg \frac{y}{x}$. O ângulo polar de um ponto p_1 em relação a um ponto de origem p_0 é o ângulo do vetor $p_1 - p_0$ no sistema de coordenadas polares habitual. Por exemplo, o ângulo polar $(3, 5)$ com relação a $(2, 4)$ é o ângulo do vetor $(1, 1)$ que é 45 graus ou $\pi/4$ radianos. O ângulo polar $(3, 3)$ com relação a $(2, 4)$ é o ângulo do vetor $(1, -1)$ que é 315 graus ou $7\pi/4$ radianos.

Envoltória Convexa

A envoltória convexa de um conjunto K de pontos é o menor polígono convexo M para o qual cada ponto de K está no limite de M ou no seu interior. Intuitivamente podemos imaginar cada ponto em K como um prego que se projeta para fora de uma tábua. Nesse caso, a envoltória seria a figura produzida por um elástico que envolva todos os pregos. Por exemplo, suponhamos o conjunto formado pelos pontos $\{(2, 1), (7, 3), (5, 4), (4, 6), (2, 3), (0, 6)\}$. A envoltória convexa seria:



Note que p_2 e p_4 não pertencem à envoltória.

Algoritmo de Graham

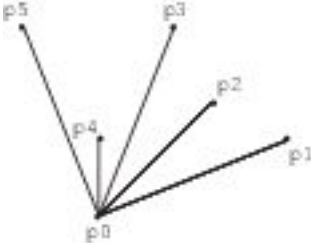
Usa uma pilha auxiliar para manter os pontos enquanto decide se eles pertencem ou não à envoltória.

- 1: Algoritmo Envoltória Convexa de Graham
- 2: $p_0 \leftarrow$ ponto em K com ordenada (y) mínima
- 3: calcular θ dos demais pontos, sempre em relação a p_0
- 4: ordenar os pontos $p_1 \dots p_m$ por θ crescente
- 5: empilhe (E, p_0)
- 6: empilhe (E, p_1)
- 7: empilhe (E, p_2)
- 8: para $i \leftarrow 3; i \leq m$
- 9: enquanto angulo formado por $NEXT, TOP$ e P_i for a direita
- 10: desempilhe (E)
- 11: fim{enquanto}
- 12: empilhe (E, P_i)
- 13: fim{para}
- 14: retorne E

Observações sobre o algoritmo Na linha 2, se houver mais de um ponto com ordenada (a medida "y") igual, deve-se pegar aquele que estiver mais à esquerda.

Na linha 3, deve-se observar que se o ângulo formado pelos 3 pontos for de $\pi/2$, a tangente é infinito (já que o denominador da fórmula do arctg seria zero). Nesse caso, deve-se usar a definição auxiliar: $\theta = \arcsen(\frac{x}{\sqrt{x^2+y^2}})$. Neste caso, no final das contas, colocar $\pi/2$.

Na linha 4, eis como ficariam os ângulos dos pontos



As linhas 5, 6 e 7 colocam os três primeiros pontos do polígono (que sempre formarão, nessa altura do problema, a envoltória convexa).

A variável $NEXT$ contém o ponto que está abaixo do $TOPO$ na pilha. $TOPO$ contém o último elemento incluído. Consultar estes dois pontos não altera a pilha.

Na linha 14, retorna-se a pilha E que contém apenas os pontos que fazem parte da envoltória convexa, listados no sentido horário.

Exemplo

Seja o seguinte conjunto de pontos: $H = \{(1, 1), (4, 1), (2, 2), (3, 4), (2, 4), (1, 5), (0, 4)\}$. Calculando os ângulos teríamos:

x	y	theta
4	1	0.00
2	2	0.79
3	4	0.98
2	4	1.25
1	5	1.57
0	4	1.89

Note que $p_0 = (1, 1)$

Executando o algoritmo teríamos a seguinte envoltória $E = \{(0, 4), (1, 5), (3, 4), (4, 1), (1, 1)\}$.

Note que dentro do *para* existe um *enquanto*. Isto significa que eventualmente haverá a retirada de um **ou mais de um** ponto(s).

Veja-se, por exemplo, o que aconteceria ao processar os pontos: $(1,1)$ $(4,5)$ $(2,8)$ $(2,9)$ $(2,10)$ e $(3,15)$. Neste caso, antes de empilhar o $(3,15)$ devem ser desempilhados o $(2,8)$, o $(2,9)$ e o $(2, 10)$, antes. Se não tiver entendido, desenhe estes pontos no plano e siga o algoritmo.

Existe um applet java que calcula e mostra a envoltoria em

<http://www.ime.usp.br/~freitas/gc/java/GrahamApp.html>

Para você fazer

Seja o seguinte conjunto de pontos:
 $\{(23,13);(16,11);(21,27);(25,3);(13,7);(9,29);(15,21);(2,30);(27,17);(12,1);(7,6);(29,25);(24,20);(14,2);(22,10)\}$
Desenhe a envoltória

e usando o algoritmo de Graham calcule quantos pontos tem a envoltória ?

A construção de um compilador

A programação estruturada Sempre é possível programar usando apenas 3 blocos básicos:

- A ordem simples (lea, imprima, atribua)
- A alternativa simples ou composta (se...então e se...então...senão...)
- A repetição (enquanto e repita)

A tese dos pesquisadores do início dos anos 70, Dijkstra – por exemplo, é que o desvio tendia (e tende) a gerar caminhos confusos e difíceis de seguir. Embora na época eles tenham sido muito questionados, o tempo acabou mostrando que eles tinham toda a razão do mundo: as linguagens mais modernas sequer têm o comando GOTO. Por esta razão todas as linguagens posteriores simplesmente baniram o comando.

Um computador real O caso é que se formos olhar a linguagem de máquina dos processadores que existem por aí, arriscamos a tomar um susto. Qualquer processador tem as seguintes ordens:

na teoria	na linguagem objeto deste exercício
Atribuição	<nome> ← expressão
Macros para fazer E/S, uma vez que do ponto de vista da máquina estas são as mais complexas operações possíveis	lea e escreva
Testes de condições. Mas, ao contrário do que estamos acostumados, a única resposta possível a um teste é um desvio	→(condição)/rotulo se a condição é verdadeira, ocorrer o desvio para < rótulo >. Senão não.
Desvios incondicionais	→rotulo <rótulo>: estabelece um local de chegada do desvio

Um compilador de Programação Estruturada Aqui chega-se ao âmago da arte de programar. Tem-se uma camada superior que define regras apropriadas ao ser humano e que visa diminuir a chance dele (ser humano) errar. Uma camada inferior que obrigatoriamente tem que obedecer à arquitetura de hardware disponível. Entre elas: um compilador que nós vamos construir.

O objetivo deste exercício é pensar como funcionaria um compilador que recebesse programas escritos usando os blocos da Programação estruturada e os convertesse em uma implementação apropriada a rodar em um hardware real.

A linguagem fonte

cabeçalho
Bem simples, formado apenas pela palavra “algoritmo” seguida do nome do mesmo
Variáveis
As variáveis do algoritmo devem ser especificadas logo após o cabeçalho do algoritmo. Haverá apenas o tipo inteiro, que por ser único não precisa ser especificado.
LEIA
O comando leia ao ser compilado interromperá o fluxo do programa, gerará uma mensagem padrão “LEITURA DA VARIÁVEL nnn”, onde nnn é o nome da variável da lista, aceitará o valor a ela fornecido via teclado, assinalará este valor a ela e continuará o processamento.
IMPRIMA
Aqui, ao ser traduzido este comando gerará uma impressão do programa, colocando antes dos valores a mensagem opcional e listando as variáveis cujos nomes fazem parte da lista. Não há interrupção no fluxo.
ASSINALAMENTO

O comando de assinalamento tem o formato
receptor <- expressão compatível. O símbolo “< -” pode ser substituído pelo símbolo ← se este estiver disponível.
SE
Comando condicional. Tem o formato
se condição ações [senão ações2] fim{se} A condição é avaliada. Se ela for verdadeira, a lista de ações é executada. Se ela for .F. e houver a lista do senão (a ações2) esta é executada. O comando sempre se encerra por um fim{se}.
ENQUANTO
Comando condicional. Tem o formato
enquanto condição lista-ações fim{enquanto} A condição é avaliada. Se ela for verdadeira, a lista-ações é executada. Ao chegar ao final da lista, há um desvio ao começo do laço. Se ela for .F. há um desvio para depois do fim{enquanto}.
REPITA
Comando condicional. Tem o formato
repita lista-ações ate {condição} A lista é executada. Ao se localizar a cláusula ate, a condição é avaliada. Se ela for verdadeira, o fluxo segue normalmente. Se ela for falsa, há um desvio ao início do laço.
PARA
Comando de repetição. Tem o formato
para var de valor1 ate valor2
[passo valor3] lista-ações fim{para} Controla laços finitos e previamente conhecidos.

Linguagem objeto

Haverá uma pilha cujo elemento tem 22 caracteres (6 de identificação, 8 de variável do para e 8 de incremento do para. Estes dois últimos campos só são usados no para) Haverá uma coleção de variáveis locais, com nomes V00, V01, V02, etc. Elas são numéricas inteiras e servem tanto para efetuar cálculos, como para receber condições (0 é falso, 1 é verdadeiro), como para estabelecer pontos de chegada de desvios, o que em linguajar de programação se conhece como “label”, desde que apareçam sozinhas no código seguido por “:”.
Na descrição a seguir, os números das variáveis são quaisquer. Usaram-se números diferentes apenas para facilitar o entendimento pelo leitor.

SE simples:
a condição é V01. Se V01 for falsa, desvie para V02. Empilhe V02. Traduza normalmente. Ao achar fim{se}, desempilhe V02 e aplique o rótulo no código.
SE composto:
a condição é V01. Se V01 for falsa, desvie para V02. Empilhe V02. Traduza normalmente. Ao achar senão, desempilhe V02 guardando-o em SALVA. Empilhe V03 e desvie para ela. Aplique SALVA no código. Traduza normalmente. Ao achar fim{se}, desempilhe V03 e aplique-o no código
ENQUANTO:
Empilhar V04, Aplicar V04 no código. Empilhar V05. Calcular V06 com a condição. Se V06 é falso, desviar para V05. Traduzir normalmente. Ao achar fim{enquanto}, desempilhar o V05, guardando-o em SALVA, desempilhar o V04, desviar para V04, e aplicar SALVA no código.
REPITA:
aplicar V07 no código e empilhar V07. Traduzir. Ao achar “ate”, empilhar V08. Colocar a condição em V09. Se V09 for verdadeira, desempilhe V08 e desvie para V08. Desempilhe o V07 e desvie para V07. Aplique V08 no código.
PARA:
Fazer variável ← valor1. Empilhar V10 e aplicar V10 no código. Empilhar V11,variável, incremento. Colocar em V12 a condição variável ≤ valor2. (Se o passo é negativo, a condição é ≥). Se V12 for falso, desviar para V11. Traduzir normalmente. Ao achar o fim{para} consultar o topo, incrementar a variável, desempilhar V11 guardar em SALVA, desempilhar V10, desviar para lá. Aplicar SALVA no código.

Um exemplo Seja o seguinte código aleatório

```
1: algoritmo TESTE
2: inteiro X,Y,Z,T
3: repita
4:   para Z de 4 ate 7 passo 2
5:     Y← 8
6:     X← Z
7:     Z ← X-Y
```

```
8:   fim{para}
9: até T = 12
10: fim{algoritmo}
Que seria transformado em
1: TESTE
2: inteiro X Y Z T V001 V002 V003 V004 V005
   V006
3: V001:
4: Z ← 4
5: V002:
6: V004← Z≤ 7
7: → (∼ V004)/V003
8: Y← 8
9: X← Z
10: Z← X-Y
11: Z ← Z + 2
12: → V002
13: V003:
14: V006← T=12
15: → (V006)/V005
16: → V001
17: V005:
```

Outro exemplo

```
1: algoritmo TESTE
2: inteiro X,Y,Z,T
3: se Y> 4
4:   se T<> 7
5:     X<-Y-Z
6:     Z<-19
7:     T<-X*X
8:   fim{se}
9: fim{se}
10: fimalgoritmo
que deu origem ao seguinte código objeto
```

```
1: TESTE X Z T V001 V002 V003 V004 V005
2: V001←Y>4
3: →(∼V001)/V002
4: V003←T<>7
5: →(∼V003)/V004
6: X←Y-Z
7: Z←19
8: T←X*X
9: V004:
10: V002:
```

Para você fazer

Usando a proposta de tradução acima, converta o seguinte trecho de código escrito em linguagem fonte:

```
algoritmo TESTE-66840
inteiro X,Y,Z,T
enquanto Z>=15
  repita
    X<-X*13
    Y<-Z
    T<-T+Y
  ate X<>15
fim{enquanto}
fim{algoritmo}
```

Responda no verso



Engenharia reversa de um compilador

linguagem fonte Formada por: cabeçalho (Bem simples, formado apenas pela palavra “algoritmo” seguida do nome do mesmo); Variáveis (As variáveis do algoritmo devem ser especificadas logo após o cabeçalho do algoritmo. Haverá apenas o tipo inteiro, que por ser único não precisa ser especificado) e Comandos, como se especifica a seguir.

LEIA

O comando leia ao ser compilado interromperá o fluxo do programa, gerará uma mensagem padrão “LEITURA DA VARIÁVEL nnn”, onde nnn é o nome da variável da lista, aceitará o valor a ela fornecido via teclado, assinalará este valor a ela e continuará o processamento.

IMPRIMA

Aqui, ao ser traduzido este comando gerará uma impressão do programa, colocando antes dos valores a mensagem opcional e listando as variáveis cujos nomes fazem parte da lista. Não há interrupção no fluxo.

ASSINALAMENTO

O comando de assinalamento tem o formato

receptor <- expressão de tipo compatível com o receptor. O símbolo “<-” pode ser substituído pelo símbolo ← se este estiver disponível.

SE

Comando condicional. Tem o formato

se condição lista-ações [senão lista-ações2] fim{se} A condição é avaliada. Se ela for verdadeira, a lista-ações é executada. Se ela for .F. e houver a lista do senão (a lista-ações2) esta é executada. O comando sempre se encerra por um **fim{se}**.

ENQUANTO

Comando condicional. Tem o formato

enquanto condição lista-ações fim{enquanto} A condição é avaliada. Se ela for verdadeira, a lista-ações é executada. Ao chegar ao final da lista, há um desvio ao começo do laço. Se ela for .F. há um desvio para depois do fim{enquanto}.

REPITA

Comando condicional. Tem o formato

repita lista-ações ate {condição} A lista é executada. Ao se localizar a cláusula **ate**, a condição é avaliada. Se ela for verdadeira, o fluxo segue normalmente. Se ela for falsa, há um desvio ao início do laço.

PARA

Comando de repetição. Tem o formato

para variável de valor1 ate valor2 [passo valor3] lista-ações fim{para} Controla laços finitos e previamente conhecidos.

Linguagem objeto

Haverá uma pilha cujo elemento tem 22 caracteres (6 de identificação, 8 de variável do para e 8 de incremento do para. Estes dois últimos campos só são usados no para) Haverá uma coleção de variáveis locais, com nomes V0000, V0001, V0002, etc. Elas são numéricas inteiras e servem tanto para efetuar cálculos, como para receber condições (0 é falso, 1 é verdadeiro), como para estabelecer pontos de chegada de desvios, o que em linguagem de programação se conhece como “label”, desde que apareçam sozinhas no código seguido por “:”.

Resumo dos desvios

Se simples SE simples: a condição é 0001. Se 0001 for falsa, desvie para 0002. Empilhe SE0002. Traduza normalmente. Ao achar fim{se}, desempilhe 0002 e aplique o rótulo no código.

Se composto SE composto: a condição é 0001. Se 0001 for falsa, desvie para 0002. Empilhe SE0002. Traduza normalmente. Ao achar senão, desempilhe 0002 guardando-o em SALVA. Empilhe SE0003 e desvie para ela. Aplique SALVA no código. Traduza normalmente. Ao achar fim{se}, desempilhe 0003 e aplique-o no código

Enquanto ENQUANTO: Empilhar EN0004, Aplicar 0004 no código. Empilhar EN0005. Calcular 0006 com a condição. Se 0006 é falso, desviar para 0005. Traduzir normalmente. Ao achar fim{enquanto}, desempilhar o 0005, guardando-o em SALVA, desempilhar o 0004, desviar para 0004, e aplicar SALVA no código.

Repita REPITA: aplicar 0007 no código e empilhar RE0007. Traduzir. Ao achar “ate”, empilhar RE0008. Colocar a condição em 0009. Se 0009 for verdadeira, desempilhe 0008 e desvie para 0008. Se 0009 for falsa, desempilhe e desvie para 0007. Aplique 008 no código.

Para PARA: Empilhar PA0010 e aplicar 0010 no código. Empilhar PA0011, variável, incremento. Fazer variável ← valor1. Colocar em 0012 a condição variável ≤ valor2. Se 0012 for falso, desviar para 0011. Traduzir normalmente. Ao achar o fim{para} consultar o topo, incrementar a variável, desempilhar 0011 guardar em SALVA, desempilhar 0010, desviar para lá. Aplicar SALVA no código.

Um exemplo

```
1: algoritmo TESTE
2: inteiro X,Y,Z,T
3: enquanto X > 10 faça
4:   se X = 14 então
5:     Z ← 10
6:     T ← Y
7:     T ← Z × 6
8:   fim se
9: fim enquanto
10: finalgoritmo
```

Que foi convertido em

```
1: TESTE ;Z;T;V001;V002;V003;V004;V005;V006
2: V001:
3: V003 ← X > 10
4: → ( ~ V003)/V002
5: V004 ← X = 14
6: → ( ~ V004)/V005
7: Z ← 10
8: T ← Y
9: T ← Z × 6
10: V005:
11: → V001
12: V002:
```

Para você fazer

Exercício 1

A seguir, uma função já compilada. Faça a engenharia reversa e escreva o código fonte original.

```
1: TESTE-82985 ;Z;T;X;V001;V002;V003;V004;V005;V006
2: Y ← 2
3: V001:
4: V003 ← Y ≤ 8
5: → ( ~ V003)/V002
6: V004:
7: Z ← X-X
8: T ← Y
9: X ← X-T
10: V006 ← X > 15
11: → (V006)/V005
12: → V004
13: V005:
14: Y ← Y + 2
15: → V001
16: V002:
```

Responda no verso

Exercício 2

Suponha a seguinte função fonte

```
1: algoritmo TESTE-53877
2: inteiro X,Y,Z,T
3: repita
4:   se T ≤ 2 então
5:     T ← T+X
6:     Y ← X+15
7:     Z ← Z- 3
8:   fim se
9: até T ≠ 3
10: finalgoritmo
```

Que foi compilada na seguinte função objeto

```
1: TESTE-53877 ;T;Y;Z;V001;V002;V003;V004;V005
2: =====
3: V002 ← T ≤ 2
4: → ( ~ V002)/V003
5: T ← T+X
6: Y ← X+15
7: Z ← Z-3
8: V003:
9: V005 ← T ≠ 3
10: → (V005)/V004
11: → V001
12: =====
```

Informe qual deveria ser o comando que aparece nas linhas duplamente sublinhadas.

1

Filas seqüenciais

Suponha filas seqüenciais com as seguintes características:

- 1: estrutura FILA
- 2: inteiro ENT, SAI
- 3: caracter LET [5]
- 4: fim estrutura {Condição de inicialização: ENT=0 e SAI=0}

```

1: função IF (FILA F, caracter L)
2: inteiro X,Y
3: se F.ENT < 5 então
4:   F.ENT++
5:   F.LET[F.ENT] ← L
6: senão
7:   se F.SAI > 0 então
8:     X ← F.ENT - F.SAI
9:     para Y de 1 até X faça
10:      F.LET[Y] ← F.LET[F.SAI+Y]
11:     fim para
12:     F.ENT ← F.ENT - F.SAI
13:     F.SAI ← 0
14:     IF (F, L)
15:   senão
16:     erro (F cheia)
17:   fim se
18: fim se
19: fim {função}
1: caracter função EF (FILA F)
2: caracter X
3: se F.SAI ≥ F.ENT então
4:   retorne '*' erro (F vazia)
5: senão
6:   F.SAI++
7:   X ← F.LET[F.SAI]
8:   se F.SAI = F.ENT então
9:     F.SAI ← F.ENT ← 0
10:   fim se
11:   devolva X
12: fim se
13: fim {função}
  
```

Siga os exemplos (supondo duas Filas: 1 e 2)

a. IF(1,A)	F1= A - - - - E1=1, S1=0	F2= - - - - - E2=0, S2=0
b. IF(1,B)	F1= A B - - - E1=2, S1=0	F2= - - - - - E2=0, S2=0
c. IF(2,EF(1))	F1= A B - - - E1=2, S1=1	F2= A - - - - E2=1, S2=0
d. IF(2,X)	F1= A B - - - E1=2, S1=1	F2= A X - - - E2=2, S2=0
e. IF(2,Y)	F1= A B - - - E1=2, S1=1	F2= A X Y - - E2=3, S2=0
f. EF(1)	F1= A B - - - E1=0, S1=0	F2= A X Y - - E2=3, S2=0
g. EF(2)	F1= A B - - - E1=0, S1=0	F2= A X Y - - E2=3, S2=1
h. IF(2,EF(1))	F1= A B - - - E1=0, S1=0	F2= A X Y * - E2=4, S2=1
i. IF(1,J)	F1= J B - - - E1=1, S1=0	F2= A X Y * - E2=4, S2=1
j. IF(2,K)	F1= J B - - - E1=1, S1=0	F2= A X Y * K E2=5, S2=1
k. IF(2,M)	F1= J B - - - E1=1, S1=0	F2= X Y * K M E2=5, S2=0
l. IF(1,EF(2))	F1= J X - - - E1=2, S1=0	F2= X Y * K M E2=5, S2=1
m. IF(1,EF(2))	F1= J X Y - - E1=3, S1=0	F2= X Y * K M E2=5, S2=2
n. IF(2,W)	F1= J X Y - - E1=3, S1=0	F2= * K M W M E2=4, S2=0
o. EF(1)	F1= J X Y - - E1=3, S1=1	F2= * K M W M E2=4, S2=0
p. EF(1)	F1= J X Y - - E1=3, S1=2	F2= * K M W M E2=4, S2=0
q. IF(1,EF(2))	F1= J X Y * - E1=4, S1=2	F2= * K M W M E2=4, S2=1
r. IF(1,EF(2))	F1= J X Y * K E1=5, S1=2	F2= * K M W M E2=4, S2=2
s. IF(1,V)	F1= Y * K V K E1=4, S1=0	F2= * K M W M E2=4, S2=2
t. IF(1,J)	F1= Y * K V J E1=5, S1=0	F2= * K M W M E2=4, S2=2
u. IF(2,P)	F1= Y * K V J E1=5, S1=0	F2= * K M W P E2=5, S2=2
v. IF(2,A)	F1= Y * K V J E1=5, S1=0	F2= M W P A P E2=4, S2=0
w. IF(1,T)	F1= Y * K V J E1=5, S1=0	F2= M W P A P E2=4, S2=0
x. EF(1)	F1= Y * K V J E1=5, S1=1	F2= M W P A P E2=4, S2=0
y. IF(2,EF(1))	F1= Y * K V J E1=5, S1=2	F2= M W P A * E2=5, S2=0

Para você fazer

Agora, simule a existência de 5 filas, numeradas de 1 a 5.

1. IF(2,I)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
2. IF(1,D)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
3. IF(1,EF(4))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
4. IF(5,T)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
5. IF(2,U)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
6. IF(1,U)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
7. IF(2,E)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_

8. EF(2)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
9. IF(4,Z)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
10. IF(4,E)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
11. IF(3,J)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
12. IF(3,R)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
13. IF(4,EF(4))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
14. IF(5,K)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
15. IF(4,V)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
16. IF(2,F)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
17. IF(5,L)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
18. EF(4)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
19. IF(1,I)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
20. IF(1,N)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
21. IF(4,EF(4))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
22. IF(5,K)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
23. IF(2,EF(1))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
24. IF(3,EF(5))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
25. EF(3)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
26. IF(4,EF(3))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
27. IF(3,EF(3))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
28. IF(3,M)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
29. IF(4,P)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
30. IF(3,Q)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
31. IF(5,Z)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
32. EF(3)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
33. IF(5,S)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
34. IF(4,C)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
35. IF(5,M)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
36. IF(4,F)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
37. IF(1,EF(5))	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
38. IF(2,Q)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
39. IF(2,U)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
40. IF(1,Q)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
41. IF(1,Z)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
42. EF(5)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
43. IF(3,N)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
44. EF(1)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
45. EF(1)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_
46. IF(1,K)	1_____E_S_ 2_____E_S_ 3_____E_S_ 4_____E_S_ 5_____E_S_

Para responder às questões abaixo, considere cada fila como sendo um vetor linear de 5 posições. Assim, F=3 2=___ pede a segunda letra da fila 3, da esquerda para a direita. E é a entrada e S a saída. Lembre que quando uma exclusão é feita a letra fica lá.

1.F=3	1_____	2.F=4	4_____	3.F=1	4_____	4.F=4	1_____	5.F=2	4_____
6.F=1	1_____	7.F=2	2_____	8.F=3	E_____	9.F=5	2_____	10.F=2	5_____



Filas Circulares

Suponha filas circulares com as seguintes características:

- 1: estrutura FILA
- 2: inteiro ENT, SAI, QT
- 3: caracter LET [5]
- 4: fim estrutura

Condição de inicialização: ENT=0, SAI=0 e QT=0

Quantidade de elementos válidos: QT

```
1: função IC (FILA C, caracter L)
2: se C.QT < 5 então
3:   C.ENT++
4:   se C.ENT ≥ 6 então
5:     C.ENT ← 1
6:   fim se
7:   C.QT++
8:   C.LET[C.ENT] ← L
9: senão
10:   ... erro ...
11: fim se
12: fim função
1: caracter função EC (FILA C)
2: caracter X
3: se C.QT > 0 então
4:   C.SAI++
5:   se C.SAI ≥ 6 então
6:     C.SAI ← 1
7:   fim se
8:   X ← C.LET[C.SAI]
9:   C.QT--
10:   devolva X
11: senão
12:   devolva '*'
13: fim se
14: fim função
```

Siga os exemplos (supondo duas Filas Circulares: 1 e 2)

a. IC(1,A)	F1= A - - - - E=1 S=0 Q=1	F2= - - - - - E=0 S=0 Q=0
b. IC(1,B)	F1= A B - - - - E=2 S=0 Q=2	F2= - - - - - E=0 S=0 Q=0
c. IC(2,EC(1))	F1= A B - - - - E=2 S=1 Q=1	F2= A - - - - - E=1 S=0 Q=1
d. IC(2,X)	F1= A B - - - - E=2 S=1 Q=1	F2= A X - - - - E=2 S=0 Q=2
e. IC(2,Y)	F1= A B - - - - E=2 S=1 Q=1	F2= A X Y - - - E=3 S=0 Q=3
f. EC(1)	F1= A B - - - - E=2 S=2 Q=0	F2= A X Y - - - E=3 S=0 Q=3
g. IC(2,EC(1))	F1= A B - - - - E=2 S=2 Q=0	F2= A X Y * - E=4 S=0 Q=4
h. IC(1,J)	F1= A B J - - - E=3 S=2 Q=1	F2= A X Y * - E=4 S=0 Q=4
i. IC(2,K)	F1= A B J - - - E=3 S=2 Q=1	F2= A X Y * K E=5 S=0 Q=5
j. EC(2)	F1= A B J - - - E=3 S=2 Q=1	F2= A X Y * K E=5 S=1 Q=4
k. IC(1,EC(2))	F1= A B J X - E=4 S=2 Q=2	F2= A X Y * K E=5 S=2 Q=3
l. IC(2,W)	F1= A B J X - E=4 S=2 Q=2	F2= W X Y * K E=1 S=2 Q=4
m. IC(2,EC(1))	F1= A B J X - E=4 S=3 Q=1	F2= W J Y * K E=2 S=2 Q=5
n. IC(1,EC(2))	F1= A B J X Y E=5 S=3 Q=2	F2= W J Y * K E=2 S=3 Q=4
o. IC(2,G)	F1= A B J X Y E=5 S=3 Q=2	F2= W J G * K E=3 S=3 Q=5
p. IC(2,A)	F1= A B J X Y E=5 S=3 Q=2	F2= W J G * K E=3 S=3 Q=5
q. EC(2)	F1= A B J X Y E=5 S=3 Q=2	F2= W J G * K E=3 S=4 Q=4
r. EC(2)	F1= A B J X Y E=5 S=3 Q=2	F2= W J G * K E=3 S=5 Q=3
s. IC(2,K)	F1= A B J X Y E=5 S=3 Q=2	F2= W J G K K E=4 S=5 Q=4
t. IC(1,EC(2))	F1= W B J X Y E=1 S=3 Q=3	F2= W J G K K E=4 S=1 Q=3
u. IC(1,EC(1))	F1= W X J X Y E=2 S=4 Q=3	F2= W J G K K E=4 S=1 Q=3
v. IC(2,U)	F1= W X J X Y E=2 S=4 Q=3	F2= W J G K U E=5 S=1 Q=4
w. IC(2,E)	F1= W X J X Y E=2 S=4 Q=3	F2= E J G K U E=1 S=1 Q=5
x. IC(1,T)	F1= W X E X Y E=3 S=4 Q=4	F2= E J G K U E=1 S=1 Q=5
y. IC(1,T)	F1= W X E T Y E=4 S=4 Q=5	F2= E J G K U E=1 S=1 Q=5
z. EC(1)	F1= W X E T Y E=4 S=5 Q=4	F2= E J G K U E=1 S=1 Q=5

Simule as seguintes operações supondo as filas 1, 2, 3, 4 e 5. Responda:

1. EC(5)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
2. IC(3,A)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
3. IC(5,EC(4))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
4. IC(3,Y)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
5. IC(4,A)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
6. EC(3)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
7. EC(2)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
8. IC(2,B)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
9. IC(3,EC(3))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
10. IC(3,G)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_

11. IC(4,EC(4))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
12. EC(3)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
13. IC(3,0)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
14. EC(5)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
15. EC(2)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
16. IC(3,EC(5))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
17. IC(2,E)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
18. EC(2)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
19. IC(3,B)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
20. IC(1,EC(2))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
21. EC(5)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
22. IC(1,EC(5))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
23. EC(4)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
24. IC(3,Y)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
25. IC(1,EC(3))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
26. IC(5,N)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
27. IC(2,V)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
28. IC(1,V)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
29. IC(3,EC(1))	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
30. IC(4,Y)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
31. EC(3)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
32. IC(5,Q)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
33. IC(2,U)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
34. IC(1,A)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
35. IC(1,P)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
36. EC(3)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
37. IC(1,Z)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
38. EC(4)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
39. IC(2,U)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
40. IC(4,R)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
41. IC(5,M)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
42. IC(1,E)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
43. IC(4,V)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
44. IC(5,F)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
45. IC(3,F)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_
46. IC(5,Q)	1...E_S_Q_ 2...E_S_Q_ 3...E_S_Q_ 4...E_S_Q_ 5...E_S_Q_

Para responder as questões abaixo, considere cada fila circular como sendo um vetor linear de 5 posições. Assim, F=3, 4=___ pede qual o valor do quarto elemento da fila 3, da esquerda para a direita. "E" é a entrada, "S" a saída e "Q" a quantidade. Lembre que quando a exclusão é feita, a letra permanece lá.

1.F=4	3_	2.F=4	5_	3.F=2	5_	4.F=3	Q_	5.F=4	4_
6.F=4	E_	7.F=2	4_	8.F=5	4_	9.F=5	1_	10.F=2	E_



Filas em Assembly

Neste exercício serão examinadas implementações de filas circulares sequenciais e encadeadas com 2 apontadores em baixo nível, ou seja usando apenas as funções básicas de manipulação de memória. Para poder estudar o funcionamento dos algoritmos usaremos a memória M, formada por 100 inteiros numerados (endereçados) de 00 a 99. A obtenção de memória é fornecida pela função OBTEMEMO que recebe como parâmetro a quantidade de inteiros pedida e devolve o endereço do primeiro deles (se houve sucesso) ou um número negativo, se não houve sucesso. Esta função trabalha baseada na variável global PPL que indica qual a Próxima Posição Livre. No início da simulação PPL ← 0.

Alocação sequencial, fila circular Haverá um descritor composto por 6 inteiros

Lim _s	Entrada	Lim _i	Saída	Tamanho	Quantidade

```
1: inteiro função FILCCRI (inteiro TAMANHO, QTD)
2: inteiro X
3: X ← OBTEMEMO (6 + TAMANHO × QTD)
4: se X ≥ 0 então
5:   M[X+1] ← M[X+2] ← M[X+3] ← X + 6
6:   M[X] ← 6 + X + (TAMANHO × QTD)
7:   M[X+4] ← TAMANHO
8:   M[X+5] ← 0
9:   devolva X
10: senão
11:   devolva -1
12: fim se
13: fim função
1: inteiro função FILCINS (inteiro F, ENDEREÇO)
2: se M[F+5] < (M[F]-M[F+2]) ÷ M[F+4] então
3:   se M[F+1] ≥ M[F] então
4:     M[F+1] ← M[F+2]
5:   fim se
6:   para Y de 0 até M[F+4]-1 faça
7:     M[M[F+1]+Y] ← M[ENDEREÇO + Y]
8:   fim para
9:   M[F+1] ← M[F + 1] + M[F+4]
10:  M[F+5]++
11:  devolve 0
12: senão
13:  devolva -1
14: fim se
15: fim função
1: inteiro função FILCEXC (inteiro F)
2: se M[F+5] > 0 então
3:   SALVA ← M[F+3]
4:   M[F+3] ← M[F+3] + M[F+4]
5:   se M[F+3] ≥ M[F] então
6:     M[F+3] ← M[F+2]
7:   fim se
8:   M[F+5]-
9:   devolva SALVA
10: senão
11:   devolva -1
12: fim se
13: fim função
```

Alocação encadeada, fila com 2 apontadores

& saída	& entrada	tamanho

```
1: inteiro função FILTCRI (inteiro TAMANHO)
2: inteiro ONDE
3: ONDE ← OBTEMEMO(3)
4: se ONDE ≥ 0 então
5:   M[ONDE] ← M[ONDE+1] ← -1
6:   M[ONDE+2] ← TAMANHO
7:   devolva ONDE
8: senão
9:   ... erro ...
10: fim se
1: função FILTINS (inteiro F, DADOS)
2: inteiro TAM, ONDE
3: TAM ← M[F+2]
4: ONDE ← OBTEMEMO(TAM+1)
5: se ONDE ≥ 0 então
6:   se M[F] = -1 ∧ M[F+1] = -1 então
7:     M[F] ← M[F+1] ← ONDE
8:     M[ONDE] ← -1
9:     para Y de 0 até TAM-1 faça
10:      M[ONDE+Y+1] ← M[DADOS+Y]
11:   fim para
12:   senão
13:     M[ONDE] ← -1
14:     M[M[F+1]] ← ONDE
15:     M[F+1] ← ONDE
```

```
16:   para Y de 0 até TAM-1 faça
17:     M[ONDE+Y+1] ← M[DADOS+Y]
18:   fim para
19:   fim se
20: senão
21:   ...erro...
22: fim se
1: inteiro função FILTEXC (inteiro F)
2: se M[F] = -1 ∧ M[F+1] = -1 então
3:   devolva -1 {erro, fila vazia}
4: senão
5:   devolva M[F]+1
6:   se M[F] = M[F+1] então
7:     M[F] ← M[F+1] ← -1
8:   senão
9:     M[F] ← M[M[F]]
10:  fim se
11: fim se
```

Na prática Suponha uma memória M de 100 inteiros, endereçados de 00 a 99, devidamente inicializada com zeros nas primeiras 80 posições e contendo de 1 a 20 nas posições 80 a 99.

Seja a seguinte sequência de comandos de manipulação de filas

```
P1 <- FILTCRI(2)                      R8 <- FILTINS(P3,90)
P2 <- FILCCRI(2,4)                    R9 <- FILTINS(P3,86)
P3 <- FILTCRI(4)                      R10 <- FILTINS(P1,85)
R2 <- FILTINS(P3,84)                  R11 <- FILCINS(P2,83)
R3 <- FILTINS(P1,87)                  R12 <- FILCEXC(P2)
R4 <- FILTINS(P3,92)                  R13 <- FILCEXC(P2)
R5 <- FILTEXC(P1)                    R14 <- FILTEXC(P3)
P4 <- FILCCRI(2,3)                    R15 <- FILCEXC(P2)
P5 <- FILTCRI(3)                      R16 <- FILCEXC(P4)
R6 <- FILTINS(P3,82)                  R17 <- FILTEXC(P3)
R7 <- FILCEXC(P2)
```

Que dará origem à seguinte disposição de M

M	0	1	2	3	4	5	6	7	8	9
0	63	63	2	17	11	9	11	2		4
1	5							48	58	4
2	28	5	6	7	8	-1	8	9	48	13
3	14	15	16	45	39	39	39	2		
4						-1	-1	3	53	3
5	4	5	6	58	11	12	13	14	-1	7
6	8	9	10	-1	6	7				
7										
8	1	2	3	4	5	6	7	8	9	10
9	11	12	13	14	15	16	17	18	19	20

👉 Para você fazer

Inicialize M como no começo do exemplo. Depois, execute a seguinte sequência de comandos de pilha. Ao final, informe as posições de M que são pedidas.

```
0:P1 =FILTCRI( 3)
1:P2 =FILCCRI( 3, 3)
2:P3 =FILTCRI( 5)
3:R2=FILTINS(P3, 82)
4:R3=FILTINS(P1, 86)
5:R4=FILTINS(P3, 84)
6:R5=FILEXC(P1)
7:P4 =FILCCRI( 2, 2)
8:R25=FILCINS(P2, 81)
9:R6=FILTINS(P3, 82)
10:R11=FILCINS(P2, 80)
11:R8=FILTINS(P3, 80)
12:R15=FILEXC(P2)
13:R26=FILTINS(P1, 87)
14:R20=FILCINS(P4, 84)
15:R10=FILTINS(P1, 80)
16:P5 =FILTCRI( 2)
17:R14=FILEXC(P3)
18:R7=FILEXC(P2)
19:R9=FILTINS(P3, 83)
```

5	33	34	64	74

para quem quiser implementar

```
1: global M[0..79]← 0;M[80..99]← 1..20;PPL← 0
2: inteiro função OBTEMEMO (inteiro X)
3: se PPL+X ≥ 99 então
4:   devolva -1
5: senão
6:   devolva PPL
7:   PPL ← PPL + X
8: fim se
```



Ordenação topológica usando filas

Ordenação Topológica

Suponha o problema de se vestir pela manhã. Existe uma lista de roupas/objetos que devem ser usados e certamente deve haver alguma ordem na sua apropriação. Suponhamos a seguinte lista:

- * meia esquerda
- * meia direita
- * sapato esquerdo
- * sapato direito
- * galocha esquerda
- * galocha direita
- * cueca ou calcinha
- * calça ou saia
- * camiseta
- * camisa
- * paletó
- * pulseira
- * anéis
- * luva direita
- * luva esquerda
- * relógio
- * óculos
- * tiara
- * chapéu ou gorro
- * cinto
- * sutiã (se for o caso)

O que se pode afirmar sobre este problema é a existência de duplas de tarefas, onde a primeira deve ser feita antes da segunda. Por exemplo: (calça,cinto) ou (camiseta,camisa) ou ainda (meia esquerda, sapato esquerdo). Obtida a lista completa de dependências, escrever uma sequência de tarefas que as obedecam (às dependências) é obter uma ordenação topológica das tarefas. A ordenação topológica é usada sempre que se precisa colocar objetos/eventos em ordem parcial.

x e y estão em ordem parcial se x deve ser feito antes de y

A ordem parcial apresenta as seguintes propriedades:

- 1. Se x deve ser feito antes de y e y deve ser feito antes de z, então: x deve ser feito antes z (propriedade transitiva).
- 2. Se x dsfa y então y NÃO dsfa x (propriedade assimétrica).
- 3. x NÃO dsfa x (propriedade irreflexiva).

São exemplos de atividades que devem sofrer ordenação parcial:
Montagem dos componentes de um veículo;
Execução de diversas disciplinas de um curso universitário com pré-requisitos;
Construção de uma ponte;
Enchimento do carrinho de supermercado,

O algoritmo de ordenação topológica trata de extrair uma ordenação linear de um conjunto de objetos/eventos que deve ser parcialmente ordenado. Isto é: Se A_j dsfa A_k então $j < k$.
Note-se que a ordenação parcial não necessariamente é única, razão pela qual o exercício a seguir sugere uma ordenação inicial, a fim de que o gabarito possa ser gerado e usado na correção.

Um pouco de história Antes de estudar o algoritmo, vale a referência de que ele está por trás do método PERT/CPM, uma das grandes conquistas da logística humana, desenvolvido na indústria bélica americana na década de 50 (especificamente para o projeto POLARIS, que tinha as seguintes características: 250 empreiteiros; 9.000 sub-empreiteiros; 70.000 peças diferentes nunca antes fabricadas; Atraso no projeto inaceitável, e que foi concluído com sucesso notável).
Para os que não lembram, (ou não estudaram, já que não eram nascidos ainda), o mundo vivia

em sobressalto, motivado pelo conflito leste oeste (capitalismo versus comunismo), origem da guerra fria que tinha seu epicentro em Berlim Ocidental, cujo mapas vemos aqui:



Chamou-se guerra (pois foi um efetivo confronto humano, no qual morreram milhões de seres humanos) mas foi também fria, já que os belicosos sempre foram terceirizados, nunca os dois blocos se confrontaram diretamente, (mas foi por muito pouco). Alguns episódios desta guerra: Revolução Chinesa (1949); Guerra da Coreia (1950-1953); Hungria (1956) Cuba (1958); Guerra do Vietnam (1961-73); Golpe no Brasil (1964); Primavera de Praga (1968); Golpe no Chile (1973); Golpe na Argentina (1976); Invasão no Afeganistão (1980), para ficarmos só nos principais.

Este conflito foi diferente de todos os havidos anteriormente na história da humanidade, porque neste apareceram as armas atômicas. Começando com a bomba de Hiroshima em 1945, pelo desenvolvimento americano da bomba H, cerca de 500 vezes mais potente, em 1951 e soviético em 1954. Peritos calcularam que nos anos 80, o arsenal americano e soviético juntos poderiam destruir o nosso planeta 1200 vezes. A situação só se descompriu um pouco no final da década de 80, quando o sistema soviético ruuiu por completo e a URSS desapareceu, surgindo em seu lugar cerca de 15 repúblicas na região. A maior é a Rússia.

Algoritmo de ordenação topológica

O funcionamento do algoritmo é simples: Busca-se um elemento que não tenha predecessor (sempre o haverá), retira-se-o do conjunto, colocando-o na saída. O conjunto restante ainda segue uma ordem parcial.

Repete-se o processo até que todos os objetos/eventos estejam na saída. A entrada do algoritmo é um conjunto de pares (a,b) que deve ser lido como "a deve ser feito antes de b".

Construa uma estrutura de dados na forma de uma tabela formada por

coluna 1 identificação da tarefa na ordem sugerida pelo exercício

coluna 2 quantidade de vezes que esta tarefa é "b"nos pares (a,b)

cols 3-10 valores de "b" quando o valor da coluna 1 é "a" em (a,b)

Por exemplo (12 nodos e 15 arestas):

(6 9) (2 4) (3 8) (2 3) (2 5) (8 9) (5 7) (6 7) (9 12) (8 11) (1 3) (3 4) (1 2) (5 10) (5 6)

Com estes dados a matriz ficaria (col 1 sugerida: números naturais)

1	0	3	2						
2	1	4	3	5					
3	2	8	4						
4	2								
5	1	7		10	6				
6	1	9	7						
7	2								
8	1	9	11						
9	2	12							
10	1								
11	1								
12	1								

Por exemplo, acompanhemos a montagem da primeira linha. Ela se refere à tarefa 1:

A coluna 2 é igual a zero, não há nenhuma antecessora para esta tarefa (não há nenhum par x,1)

Já as colunas 3..10 têm os valores 3 e 2, pois existem os pares (1,3) e (1,2), nesta ordem. (ou seja, 1 deve ser feita antes da 3 e da 2)

Obs: Para os algoritmos de ENFILEIRA e DESENFILEIRA use aqueles tradicionais vistos em sala de aula (linear/circular/encadeada, tanto faz).

```
1: Algoritmo RESOLVE-ORDENAÇÃO-TOPOLOGICA
2: para I de 1 até TAM
3:   se TAB[I;2] = 0
4:     ENFILEIRA (F,TAB[I;1])
5:   fim{se}
6: fim{para}
7: enquanto não vazia(F)
8:   VEZ ← DESENFILEIRA (F)
9:   imprima VEZ
10:  enquanto VEZ tiver sucessores
11:    subtraia 1 do sucessor {os sucessores estão nas cols 3..10}
12:    se TAB[sucessor;2] = 0
13:      ENFILEIRA (F,sucessor)
14:    fim{se}
15:  fim{enquanto}
16: fim{enquanto}
17: fim algoritmo
```

Resposta para este caso:

1, 2, 3, 5, 8, 4, 10, 6, 11, 9, 7, 12

Outro caso exemplo: (12 nodos, 15 arestas)

(13 8) (8 7) (20 6) (19 3) (4 1) (7 11) (19 11) (3 20) (15 6) (6 14) (1 6) (7 19) (8 4) (4 15) (3 14)

Sequência sugerida:

13, 8, 7, 19, 11, 3, 20, 4, 15, 1, 6, 14

terá como resposta:

13, 8, 7, 4, 19, 1, 15, 3, 11, 20, 6, 14

OBS: 1. A sequência sugerida, JA É A COLUNA 1.
2. Como se pode ver no exemplo, em caso de empate (mais de uma tarefa com zero na coluna 2), enfileire ANTES a tarefa de baixo na tabela.

Para você fazer

Conjunto com 12 nodos e 15 arestas. Sejam os pares:

(15,14) (17,4) (5,13) (15,3) (8,17) (4,12) (14,10) (13,15) (3,9) (12,13) (4,5) (12,1) (13,14) (17,12) (15,9)

Coluna 1 sugerida:

8 17 4 12 5 13 15 3 9 1 14 10

Resposta:



Bibliografia recomendada: Szwarcfiter, J e Markenzon L. ED e seus algoritmos, pag. 47-51 Knuth, D. The Art of Computer Programming. v1. pags 258-265

O jogo do TRUCO Considere uma representação possível para um baralho de truco. São 4 naipes ($\diamond$ =ouros, $\spadesuit$ =espadas, $\heartsuit$ =copas e $\clubsuit$ =paus) e 10 cartas em cada naipe: 4, 5, 6, 7, Q, J, K, A, 2, 3, da menos válida para a mais válida. Lembre-se que na memória encadeada, existirão OUTRAS coisas além do baralho. Cuidado com as coisas que podem ser assumidas. *Muito pouca coisa pode.*

Cada carta está identificada por um número do tipo NC, onde N=naipe (1= $\diamond$ (ouros), 2= $\spadesuit$ (espadas), 3= $\heartsuit$ (copas) e 4= $\clubsuit$ (paus)). As cartas usam o código 4=0, 5=1, 6=2, 7=3, Q=4, J=5, K=6, A=7, 2=8 e 3=9. Assim por exemplo, a carta 25 é o valete de espadas...

Qual a carta 17 _____ e a carta 28 _____ e a 42 _____?

A memória M é aquela convencional, já vista em sala. Existirão na memória abaixo, diversas listas. J_1 , J_2 , J_3 e J_4 são as listas de cada jogador. Cada uma conterá 3 cartas, como é próprio do truco. Haverá uma lista RESTO que conterá o resto do baralho. (28 cartas).

Vamos localizar as cartas no seguinte exemplo										
M	0	1	2	3	4	5	6	7	8	9
0	93	37	-1	12	1	44	3	16	5	24
1	7	26	-1	34	-1	36	9	18	15	14
2	17	28	19	15	21	31	11	33	23	13
3	-1	49	27	40	31	22	33	47	35	38
4	37	27	39	29	41	42	13	23	43	48
5	47	46	49	20	51	25	53	10	25	32
6	29	43	-1	17	55	30	63	19	61	21
7	45	39	59	35	67	41	65	45	75	11
8	77	42	64	6	55	49	44	87	73	36
9	46	47	45	57	71	73	69	79	82	99

Os apontadores das listas (J_1 , J_2 , J_3 , J_4 , RE) estão num bloco de 5 inteiros apontados pelo &0. Cada apontador de lista é 1 inteiro. Cada carta será composta de 2 inteiros: CARTA + & do próximo. Terminador é -1.

O bloco de descritores começa no endereço apontado por 0 e é 57,71,73,69 e 79.

As cartas do J_1 são: 10, 31 e 26 (4 $\diamond$, 5 $\heartsuit$ e K $\spadesuit$).
As cartas do J_2 são: 39, 32 e 13 (3 $\heartsuit$, 6 $\heartsuit$ e 7 $\diamond$).
As cartas do J_3 são: 35, 19 e 43 (J $\heartsuit$, 3 $\diamond$ e 7 $\clubsuit$).
As cartas do J_4 são: 21, 42 e 34 (5 $\spadesuit$, 6 de $\clubsuit$ e Q $\heartsuit$).
A carta que virou (a segunda da lista RESTO) é 45 (J de $\clubsuit$).

Exemplo completo

M	0	1	2	3	4	5	6	7	8	9
0	93	32	-1	14	-1	20	-1	30	3	34
1	-1	29	5	38	1	22	11	43	15	47
2	9	48	17	49	21	45	23	40	25	10
3	-1	44	19	11	13	12	27	13	7	25
4	35	15	39	17	29	39	41	41	45	19
5	47	37	43	35	49	36	53	42	55	16
6	57	26	59	21	61	23	63	18	65	28
7	67	46	69	27	71	31	73	24	75	33
8	77	64	55	13	9	39	38	72	42	35
9	82	48	90	33	51	31	37	79	54	89

Escreva aqui as cartas das listas				carta vira
J_1	J_2	J_3	J_4	

Rodada	J_1	J_2	J_3	J_4
1	3	2	1	1
2	1	1	2	3
3	2	3	3	2

cartas efetivamente jogadas				
Rodada	J_1	J_2	J_3	J_4
primeira				
segunda				
terceira				

Resultado do Exemplo

Dupla vencedora da rodada 1: _____ (1,2,E=empate)

Dupla vencedora da rodada 2: _____ (1,2,E)

Dupla vencedora da rodada 3: _____ (1,2,E)

Vencedor final _____ (1,2,E)

Para você fazer: Exercício 1

A configuração para a partida que você jogará é:

M	0	1	2	3	4	5	6	7	8	9
0	92	31	-1	48	-1	13	-1	46	5	33
1	7	19	9	49	11	43	13	10	15	34
2	-1	38	17	25	21	32	19	11	23	14
3	27	16	-1	39	29	40	33	18	25	20
4	3	44	35	30	41	35	43	15	45	42
5	31	26	47	28	1	45	51	36	55	24
6	57	47	39	37	59	12	63	29	65	23
7	67	21	69	17	71	22	49	41	73	27
8	53	75	87	29	57	79	25	18	2	16
9	81	50	75	79	37	61	77	48	83	28

O truco é jogado em duplas. A dupla "1" é composta por J_1 e J_3 que joga contra a dupla "2" (de J_2 e J_4). Cada partida de truco tem 3 rodadas. (daí os jogadores terem 3 cartas). A cada rodada, cada jogador joga 1 de suas cartas. A dupla que tiver o jogador que jogar a maior carta, será a vencedora da rodada. A dupla que vencer 2 rodadas, vence a partida. (Há casos raros de empate). Mesmo resolvido o jogo na rodada 2, você DEVE CALCULAR A RODADA 3

Escreva aqui as cartas das listas				carta vira
J_1	J_2	J_3	J_4	

Neste jogo, cada jogador jogará as seguintes cartas:

Rodada	J_1	J_2	J_3	J_4
1	3	2	2	1
2	1	3	3	2
3	2	1	1	3

cartas efetivamente jogadas				
Rodada	J_1	J_2	J_3	J_4
primeira				
segunda				
terceira				

Você deve
Descobrir qual a carta que virou (é a segunda da lista RESTO)
Descobrir quais as manilhas (a carta seguinte a da que virou)
Determinar quem ganhou e responder as questões abaixo.

Resultado
Dupla vencedora da rodada 1: _____ (1,2,E=empate)
Dupla vencedora da rodada 2: _____ (1,2,E)
Dupla vencedora da rodada 3: _____ (1,2,E)
Vencedor final _____ (1,2,E)

Para você fazer: Exercício 2

Para o próximo exercício, são dadas as cartas em cada uma das listas. O bloco de apontadores continua sendo apontado pelo endereço ZERO de M. Cada carta continua sendo representada por um bloco de 2 inteiros: (carta + &próximo). Preencha a memória a seguir, manuseando APENAS OS APONTADORES. As cartas já estão colocadas na memória.

As cartas de cada lista são:
 J_1 = 46,31,30
 J_2 = 41,35,21
 J_3 = 27,32,17
 J_4 = 40,24,26
RESTO= 22,15,19,11,12,20,37,33,44,49,43,10,42,25,14,34,16,39,47,36,13,48,18,38,23,29,28,45

Preencha a tabela a seguir:										
M	0	1	2	3	4	5	6	7	8	9
0	92	45		28		29		23		38
1		18		48		13		24		36
2		17		47		27		30		39
3		16		34		14		25		42
4		10		43		26		49		44
5		46		33		37		41		20
6		12		11		31		19		32
7		15		35		22		21		40
8										
9			51	57	25	79	75			

Não esqueça que na memória acima, já estão preenchidas as cartas. Só faltam os endereços, que você deve achar.

Os conteúdos da memória em cada endereço especificado:

& 22 de M= _____ & 62 de M= _____.



- 1 - /

criação e manuseio de 5 listas encadeadas

Suponha uma lista de livros com 5 encadeamentos: título do livro, idioma de publicação, preço, quantidade de páginas e cidade onde foi editado.
Exemplo (igual para todos): Suponha ainda que a lista possui 25 livros, com terminador valendo -1 e com os seguintes endereços iniciais:

título	18	idioma	10	preço	2	pags	5	cidade	22

Seja a seguinte lista de livros:

	título	&p	idioma	&p	preço	&p	pags	&p	cidade	&p
1	BLUSPD	6	SERVO	19	10.56	14	181	8	MADRID	23
2	OTTKVJ	23	VALAO	24	9.09	3	87	21	BLUMENAU	5
3	FQCZYQ	11	FINLANDES	5	9.94	1	60	4	CAIOBA	6
4	ODMLMA	2	HUNGARO	11	14.16	13	63	24	FLORIPA	10
5	GOMOUR	15	FINLANDES	12	17.15	20	19	9	BLUMENAU	7
6	BPTQMN	22	CROATA	14	31.81	22	108	16	CASCADEL	15
7	LQXIFJ	17	ALBANES	9	42.74	23	176	1	BRASILIA	3
8	SULHER	12	NORUEGUES	18	47.26	21	181	19	VALENCIA	-1
9	CEEEAP	16	ALEMAO	16	45.04	16	24	11	POMERODE	24
10	YFVVDKH	13	AFRIKANER	25	26.56	15	96	25	FLORIPA	14
11	FUVMXI	5	INGLES	8	47.12	8	31	14	ORLANDO	9
12	SZMUYG	10	FLAMENGO	22	36.95	25	163	15	DUBLIN	4
13	YXDFAM	-1	TUPI	21	14.68	19	134	12	SAO PAULO	8
14	AXEZR	1	ESPANHOL	3	12.82	4	35	17	GENEBRA	19
15	KFYIVQ	21	SUECO	13	26.59	18	174	7	COLUMBUS	16
16	DPCXLE	20	ARABE	17	45.69	11	132	13	CURITIBA	12
17	NAJCHM	19	CATALAO	6	31.49	6	49	3	SALVADOR	13
18	AFCXGS	25	NORUEGUES	20	29.18	17	214	-1	MATINHOS	25
19	NNDOLA	4	SERVO	15	16.62	5	186	22	LONDRINA	21
20	FJKAAQ	3	PORTUGUES	23	18.04	10	213	18	LYON	1
21	KTDSHQ	7	TUPI	2	47.68	-1	87	23	LONDRINA	20
22	BRSTATM	9	FRANCES	4	36.50	24	211	20	BARCELONA	2
23	PAXEEB	24	RUSSO	1	45.03	9	92	10	MARINGA	18
24	PXLSRI	8	VALAO	-1	36.86	12	78	2	POMERODE	17
25	ALKXIN	14	AFRIKANER	7	39.71	7	101	6	NEW YORK	11

Se nesta lista, você quisesse incluir o livro:
26. Livro=KGUGEI, Idioma=VALAO, Preço=42.76, Pags=131, Cidade=CURITIBA
quais seriam os seus 5 indicadores ?
21 -1 23 16 12
E após incluir o livro
27. Livro=GNPTXO, Idioma=FINLANDES, Preço=17.66, Pags=110, Cidade=NEW YORK
5 12 20 26 11

🔗 Para você fazer

endereços iniciais: T= 19 I= 7 P= 4 Pa= 16 C= 1

Seja a seguinte lista de livros:

	título	&p	idioma	&p	preço	&p	pags	&p	cidade	&p
1	IBSKNL	20	CATALAO	8	37.20	24	131	4	BLUMENAU	2
2	COMXFN	21	HOLANDES	24	41.03	8	191	19	BLUMENAU	3
3	EDZTIN	6	ALEMAO	9	13.43	25	210	-1	BLUMENAU	4
4	QFPLDK	24	CHINES	20	8.13	10	157	18	BRASILIA	21
5	KOKDXS	9	LATIM	15	45.47	-1	59	15	CASCADEL	11
6	HQLYUB	15	INGLES	5	27.05	11	204	9	VALENCIA	-1
7	SREBLY	13	AFRIKANER	3	36.47	1	200	6	DUBLIN	15
8	LVJUJM	22	CATALAO	22	41.40	17	175	10	CURITIBA	7
9	KOMKRG	8	ALEMAO	14	40.64	2	205	20	CUIABA	8
10	PFUPCI	14	CROATA	17	8.21	14	188	2	MARINGA	16
11	NKCMZT	12	NORUEGUES	16	27.92	21	199	7	CASCADEL	20
12	OQBOVN	10	VALAO	-1	34.91	7	40	25	MORRETES	25
13	TYROOI	16	GUARANI	2	24.21	22	108	22	SAO PAULO	6
14	PHUGGO	4	ALEMAO	11	9.07	23	171	8	MADRID	18
15	HVIYTC	1	LATIM	11	18.76	13	82	21	DUBLIN	17
16	UIAIRE	-1	PORTUGUES	12	11.80	3	29	12	MATINHOS	12
17	CKZCEN	2	CROATA	19	41.83	5	122	23	DUBLIN	14
18	SNEZDA	7	FINLANDES	25	17.48	20	161	14	MADRID	19
19	CCVLLR	17	CROATA	21	30.53	12	197	11	MADRID	10
20	JBBMZO	5	COREANO	10	17.59	15	208	3	CASCADEL	9
21	DYHLED	3	CROATA	18	29.09	19	84	13	CAIOBA	23
22	MPVHDV	25	CATALAO	4	26.88	6	116	17	POMERODE	24
23	RVRQEP	18	IDICHE	6	10.21	16	125	24	CAIOBA	5
24	RLVJQG	23	HOLANDES	23	40.06	9	129	1	SALVADOR	13
25	NHZEPG	11	FLAMENGO	13	15.58	18	46	5	PARIS	22

Se nesta lista, você quisesse incluir o livro:

Livro=ULPHUE Idioma=HOLANDES Preço=44.78 Pags= 47
Cidade=GENEBRA , quais seriam os seus 5 indicadores ?

E após incluir o livro
Livro=NRXNXE Idioma=FINLANDES Preço=28.46 Pags=113
Cidade=SALVADOR , quais seriam os seus 5 indicadores ?

:_____ , _____ :_____ , _____ :_____ , _____ :_____

OBSERVAÇÃO: Na inclusão, se já existe uma série igual ao elemento que está sendo incluído, a nova inclusão vai ao final desta série.
Exemplo: Já há 2 Valências no arquivo e vai-se incluir outra. A que está entrando entra depois das duas que já há.
Após incluir o livro 26, seus apontadores CONGELAM. Na inclusão do 27, os do 26 ainda que sejam alterados ficam como eram antes.



Listas duplamente encadeadas em Assembly

ATENÇÃO: PARA ENDEREÇAMENTO DESTA MEMÓRIA, A ORIGEM DOS INDICES É ZERO.

Imagine uma memória fictícia, composta por um vetor de 100 inteiros. Embora visualizada como uma matriz 10 x 10, ela é um vetor[0..99]

M	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
...										
9	0	0	0	0	0	0	0	0	0	0

Suponha que desejamos criar nesta memória listas duplamente encadeadas. Cada lista terá um descritor composto dos seguintes campos:

endereço da cabeça	endereço da cauda	tamanho do elemento (sem os endereços)	quantidade de elementos existentes

Cada elemento será formado por 3 inteiros, como se pode ver:

endereço próximo	endereço anterior	elemento da lista

1. Criando duas destas listas, inicialmente vazias, nos endereços 0 e 4,

M	0	1	2	3	4	5	6	7	8	9
0	-1	-1	1	0	-1	-1	1	0	0	0

2. Inserindo o número 33 na lista 0, fica

M	0	1	2	3	4	5	6	7	8	9
0	8	8	1	1	-1	-1	1	0	-1	-1
1	33	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0

3. Inserindo o número 25 na lista 4, fica

M	0	1	2	3	4	5	6	7	8	9
0	8	8	1	1	11	11	1	1	-1	-1
1	33	-1	-1	25	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0

4. Inserindo 40 ao final da lista 0, fica

M	0	1	2	3	4	5	6	7	8	9
0	8	14	1	2	11	11	1	1	14	-1
1	33	-1	-1	25	-1	8	40	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0

5. Inserindo 30 no início da lista 4, fica

M	0	1	2	3	4	5	6	7	8	9
0	8	14	1	2	17	11	1	2	14	-1
1	33	-1	17	25	-1	8	40	11	-1	30
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0

6. Inserindo 73 entre o primeiro e o segundo elemento na lista 0

M	0	1	2	3	4	5	6	7	8	9
0	8	14	1	3	17	11	1	2	20	-1
1	33	-1	17	25	-1	20	40	11	-1	30
2	14	8	73	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0

7. Inserindo 58 entre o 1 e o 2 na lista 4, fica

M	0	1	2	3	4	5	6	7	8	9
0	8	14	1	3	17	11	1	3	20	-1
1	33	-1	23	25	-1	20	40	23	-1	30
2	14	8	73	11	17	58	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0

8. Inserindo 90 ao final da lista 0, fica

M	0	1	2	3	4	5	6	7	8	9
0	8	26	1	4	17	11	1	3	20	-1
1	33	-1	23	25	26	20	40	23	-1	30
2	14	8	73	11	17	58	-1	14	90	0
3	0	0	0	0	0	0	0	0	0	0

9. Inserindo 10 entre o 1 e 0 2 da lista 0

M	0	1	2	3	4	5	6	7	8	9
0	8	26	1	5	17	11	1	3	29	-1
1	33	-1	23	25	26	20	40	23	-1	30
2	14	29	73	11	17	58	-1	14	90	20
3	8	10	0	0	0	0	0	0	0	0

10. Inserindo 52 ao final da lista 4

M	0	1	2	3	4	5	6	7	8	9
0	8	26	1	5	17	32	1	4	29	-1
1	33	32	23	25	26	20	40	23	-1	30
2	14	29	73	11	17	58	-1	14	90	20
3	8	10	-1	11	52	0	0	0	0	0

11. Inserindo 11 entre o 1 e o 2 na lista 0

M	0	1	2	3	4	5	6	7	8	9
0	8	26	1	6	17	32	1	4	35	-1
1	33	32	23	25	26	20	40	23	-1	30
2	14	29	73	11	17	58	-1	14	90	20
3	35	10	-1	11	52	29	8	11	0	0

12. Excluindo agora o segundo elemento da lista 0, fica

M	0	1	2	3	4	5	6	7	8	9
0	8	26	1	5	17	32	1	4	29	-1
1	33	32	23	25	26	20	40	23	-1	30
2	14	29	73	11	17	58	-1	14	90	20
3	8	10	-1	11	52	29	8	11	0	0

Para você fazer

Inicialize a Memória M toda ela com zeros e execute as seguintes operações:

- Inicialmente crie 2 listas duplamente encadeadas (lista 0 e lista 4)
- Insira, na lista 4, o elemento 11 ao final da lista
- Insira, na lista 4, o elemento 73 no início da lista
- Insira, na lista 0, o elemento 10 ao final da lista
- Insira, na lista 0, o elemento 14 ao final da lista
- Insira, na lista 0, o elemento 56 entre 1 e 2
- Insira, na lista 0, o elemento 52 no início da lista
- Insira, na lista 4, o elemento 82 entre 1 e 2
- Insira, na lista 4, o elemento 95 no início da lista
- Insira, na lista 4, o elemento 79 no início da lista
- Insira, na lista 0, o elemento 59 ao final da lista
- Insira, na lista 0, o elemento 46 entre 4 e 5
- Insira, na lista 0, o elemento 60 no início da lista

Para ajudar, eis uma memória M vazia:

M	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

E, agora responda qual o valor na célula pedida de M:

M[18]	M[22]	M[23]	M[36]	M[39]

1

1. Excluir MATINHOS da lista 3
2. Excluir CHINES da lista 2
3. Excluir HOLANDA da lista 1
4. Excluir PARAGUAI da lista 1
5. Incluir GENEBA na lista 3, com k = 5
6. Incluir IEMEN na lista 1, com k = 2
7. Incluir MORRETES na lista 3, com k = 4
8. Incluir FINLANDIA na lista 1, com k = 10
9. Incluir CHICAGO na lista 3, com k = 2
10. Incluir ALBANES na lista 2, com k = 1

Eis como ficaram as listas do exemplo acima:

cab:7, cau:12			cab:5, cau:7			cab:1, cau:4		
1.	HOLANDA	2 7	1.	PORTUGUES	10 2	1.	CUIABA	9 -1
2.	SUECIA	11 7	2.	SUECO	1 11	2.	CASCADEL	12 8
3.	CROACIA	4 6	3.	BASCO	4 9	3.	LONDRINA	4 5
4.	CHILE	9 3	4.	VALAO	8 3	4.	VALENCIA	-1 3
5.	SERVIA	6 11	5.	LATIM	11 -1	5.	CURITIBA	3 6
6.	BELGICA	3 5	6.	CHINES	9 10	6.	PARIS	5 11
7.	GABAO	2 -1	7.	AFRIKANER	-1 8	7.	MATINHOS	2 8
8.	B. FASO	12 9	8.	NORUEGUES	7 4	8.	LYON	2 13
9.	FRANCA	8 4	9.	HOLANDES	3 10	9.	MARINGA	13 1
10.	PARAGUAI	5 2	10.	TUPI	9 1	10.	CAIOBA	11 12
11.	IEMEN	5 2	11.	ALBANES	2 5	11.	GENEBRA	6 10
12.	FINLANDIA	-1 8	12.			12.	MORRETES	10 2
13.			13.			13.	CHICAGO	8 9

Manuseio de LDEs

Onde manusear ?

No exemplo a seguir, a ordenação dos elementos em uma Lista Duplamente Encadeada (LDE) não é determinada por um atributo intrínscio ao dado e sim por um indicador externo à lista. Assim, para inclusão LÓGICA, será fornecida uma chave K que indicará o local da inclusão. Para uma LDE contendo n elementos, tem-se

Se $k = 0$ a inclusão deve ser no início da LDE (à esquerda de tudo)

Se $k > 0$ e $k \leq n$ a inclusão deve ser à direita do elemento n

Se $k > n$ a inclusão deve ser feita ao final da LDE (à direita de tudo)

Note que fisicamente as inclusões SEMPRE se dão ao final da LDE Para exclusão LÓGICA, k indica qual o elemento da LDE deve ser excluído. Fisicamente o elemento excluído fica lá.

Um exemplo

1. Seja uma lista de pessoas, (*=terminador, H=head, T=tail) cuja ordem física é: Z,A,C,D,L e cuja ordem lógica é: C,Z,D,L,A Eis como ficaria

H=3; T=2;
1=Z,4,3; 2=A,*,5; 3=C,1,*; 4=D,5,1; 5=L,2,4

2. Incluindo a pessoa E com k = 3, fica
Ordem física: Z,A,C,D,L,E e ordem lógica: C,Z,D,E,L,A
Eis como ficaria:

H=3; T=2;
1=Z,4,3; 2=A,*,5; 3=C,1,*; 4=D,6,1; 5=L,2,6; 6=E,5,4

3. Excluindo a pessoa Z, fica
Ordem física: ZACDLE, ordem lógica: CDELA

H=3; T=2; 1=Z,4,3; 2=A,*,5; 3=C,4,*; 4=D,6,3;
5=L,2,6; 6=E,5,4, ou
H=3; T=2; 2=A,*,5; 3=C,4,*; 4=D,6,3; 5=L,2,6;
6=E,5,4

4. Incluindo S com k = 10, fica
Ordem física: ZACDLES, ordem lógica: CDELAS

H=3; T=7; 2=A,7,5; 3=C,4,*; 4=D,6,3; 5=L,2,6;
6=E,5,4; 7=S,*,2

5. Excluindo o C, fica
Ordem física: ZACDLES, ordem lógica: DELAS

H=4; T=7; 2=A,7,5; 3=C,4,*; 4=D,6,*; 5=L,2,6;
6=E,5,4; 7=S,*,2 ou
H=4; T=7; 2=A,7,5; 4=D,6,*; 5=L,2,6; 6=E,5,4;
7=S,*,2

6. Incluindo C com k = 2, fica
Ordem física: ZACDLESC, ordem lógica: DECLAS

H=4; T=7; 2=A,7,5; 4=D,6,*; 5=L,2,8; 6=E,8,4;
7=S,*,2; 8=C,5,6

7. Incluindo X com k = 0
Ordem física: ZACDLESCX, ordem lógica: XDECLAS

H=9; T=7; 2=A,7,5; 4=D,6,9; 5=L,2,8; 6=E,8,4;
7=S,*,5; 8=C,5,6; 9=X,4,*

8. Excluindo A, fica
Ordem física: ZACDLESCX, ordem lógica: XDECLS

H=4; T=7; 2=A,7,5; 4=D,6,9; 5=L,7,8; 6=E,8,4;
7=S,*,5; 8=C,5,6; 9=X,4,*
H=4; T=7; 4=D,6,9; 5=L,7,8; 6=E,8,4; 7=S,*,5;
8=C,5,6; 9=X,4,*

Outro exemplo

Seja agora um exemplo mais palpável. Existem 3 LDEs, a primeira de países, a segunda de idiomas e a terceira de cidades.

cab:7, cauda:8			cab:5, cauda:7			cab:1, cauda:4		
país	pr	an	idioma	pr	an	cidade	pr	an
1-HOLANDA	2	7	1-PORTUGUES	10	2	1-CUIABA	9	-1
2-SUECIA	10	1	2-SUECO	1	5	2-CASCADEL	10	7
3-CROACIA	4	6	3-BASCO	4	9	3-LONDRINA	4	5
4-CHILE	9	3	4-VALAO	8	3	4-VALENCIA	-1	3
5-SERVIA	6	10	5-LATIM	2	-1	5-CURITIBA	3	6
6-BELGICA	3	5	6-CHINES	9	10	6-PARIS	5	10
7-GABAO	1	-1	7-AFRIKANER	-1	8	7-MATINHOS	2	8
8-B. FASO	-1	9	8-NORUEGUES	7	4	8-LYON	7	9
9-FRANCA	8	4	9-HOLANDES	3	6	9-MARINGA	8	1

Para você fazer

Lista 1: países			Lista 2: idiomas			Lista 3: cidades		
cab:7, cauda: 8			cab:5, cauda: 4			cab:5, cauda: 8		
país	pr	an	idioma	pr	an	cidade	pr	an
-----	--	--	-----	--	--	-----	--	--
RUSSIA	10	7	INGLES	4	3	SAO PAULO	2	6
NORUEGA	4	10	IIDICHE	3	7	BRASILIA	10	1
URUGUAI	9	4	ALEMAO	1	2	MORRETES	9	10
EGITO	3	2	FLAMENGO	-1	1	POMERODE	7	9
JAPAO	6	9	BASCO	8	9	BERLIM	6	-1
R. TCHECA	8	5	NORUEGUES	9	10	PARIS	1	5
ALEMANHA	1	-1	HUNGARO	2	8	ORLANDO	8	4
CHILE	-1	6	VALAO	7	5	COLUMBUS	-1	7
CHINA	5	3	HOLANDES	5	6	SALVADOR	4	3
LIBERIA	2	1	AFRIKANER	6	-1	GENEBRA	3	2

TAREFAS:

0. Listar as 3 listas na ordem direta e inversa para ver como ficaram

1. Excluir URUGUAI da lista 1
2. Excluir PARIS da lista 3
3. Excluir JAPAO da lista 1
4. Excluir INGLES da lista 2
5. Incluir CROATA na lista 2, com k = 1
6. Incluir MARINGA na lista 3, com k = 5
7. Incluir DUBLIN na lista 3, com k = 9
8. Incluir JAPONES na lista 2, com k = 10
9. Incluir CHICAGO na lista 3, com k = 7
10. Incluir CAIOBA na lista 3, com k = 1

Elemento	PR Ximo	ANTErior
CROATA	6	10
MARINGA	1._____	2._____
DUBLIN	3._____	4._____
JAPONES	5._____	6._____
CHICAGO	7._____	8._____
CAIOBA	9._____	10._____

Em resumo

1. Para incluir
 - (a) Inclua fisicamente o registro ao final (NOVO)
 - (b) Para cada atributo, localize o PRÉVIO e o POSTERIOR
 - (c) Faça
 - 1: PROX(NOVO) ← PROX(PREVIO)
 - 2: ANTE(NOVO) ← ANTE(POSTERIOR)
 - 3: PROX(PREVIO) ← NOVO
 - 4: ANTE(POSTERIOR) ← NOVO
2. Para excluir
 - (a) Achar o sujeito a excluir (QUEM)
 - (b) Para cada atributo, localize o PREVIO e o POSTERIOR
 - (c) Faça
 - 1: PROX(PREVIO) ← PROX(QUEM)
 - 2: ANTE(POSTERIOR) ← ANTE(QUEM)



Bancos de Dados usando encadeamento duplo A organização que vai ser vista aqui, era o único modelo possível na década de 80, quando surgiram os primeiros SGDBs (sistemas gerenciadores de bancos de dados). Recordemos que era uma época em que as memórias dos mainframes raramente ultrapassavam 500KB. Micros e minis ainda não existiam.

Diante de tamanha escassez de recursos, o esquema a seguir era o único que podia ser usado. Era muito eficiente e só deixou de ser usado porque não muito robusto e para o usuário a mescla entre dados e apontadores foi quase sempre uma combinação mortal. O modelo relacional que veio no começo do século seguinte resolveu esse fato (embora seja muito menos eficiente, ou seja mais gastador de recursos de máquina).

A única diferença de implementação, era que os bancos mantinham os apontadores em arquivos de apoio separados dos dados (buscando melhorar um pouco o aspecto acima citado). No mais, era exatamente como aqui descrito.

Suponha bancos de dados, contendo diversos atributos e organizando os dados de maneira a poder acessar os dados originais segundo diversas ordenações. Por exemplo, seja o arquivo de animais

	animal	↑	↓	nome	↑	↓	cor	↑	↓	idade	↑	↓
		4	6		1	3		6	7		6	7
1	bola	7	3	canari	5	0	cinza	2	4	08	5	3
2	mimi	5	7	peixe	3	4	claro	3	1	03	3	4
3	binho	1	4	pomba	0	2	louro	5	2	04	1	2
4	alex	3	0	gato	2	7	branco	1	6	02	2	6
5	neto	6	2	cao	6	1	marrom	7	3	09	7	1
6	piupiu	0	5	cobra	7	5	bege	4	0	01	4	0
7	fifi	2	1	furao	4	6	pardo	0	5	11	0	5
8												

Aqui, o terminador (ao estilo do compilador C) é ZERO. Como neste exercício os índices sempre começam em 1, não há risco de confusão.

Agora, deve-se incluir o animal feio minhoca ruivo 05. Verifique que para incluir um novo registro em um encadeamento duplo, em termos físicos o novo registro sempre entra ao final, neste caso, na linha 7, que é o único local adequado para receber um novo item sem precisar alterar os itens já existentes. Quando à inclusão lógica, para tal sempre é necessário alterar 4 cursores (ou apontadores) para cada um dos atributos. Neste caso, como há 4 atributos, é necessário alterar 16 cursores para cada inclusão.

Após acertar os cursores, eis como fica o resultado

	animal	↑	↓	nome	↑	↓	cor	↑	↓	idade	↑	↓
		4	6		1	3		6	8		6	7
1	bola	8	3	canari	5	0	cinza	2	4	08	5	8
2	mimi	5	7	peixe	3	8	claro	3	1	03	3	4
3	binho	1	4	pomba	0	2	louro	5	2	04	8	2
4	alex	3	0	gato	8	7	branco	1	6	02	2	6
5	neto	6	2	cao	6	1	marrom	7	3	09	7	1
6	piupiu	0	5	cobra	7	5	bege	4	0	01	4	0
7	fifi	2	8	furao	4	6	pardo	8	5	11	0	5
8	feio	7	1	minhoc	2	4	ruivo	0	7	05	1	3

Note que a ordenação sempre é estável, ou seja, havendo duplicidade de chave, a ordem original é preservada.

Tendo preenchido corretamente o quadro, perceba-se que o arquivo pode ser percorrido em 8 ordens diferentes. Aquela estabelecida por cada um dos 4 atributos do arquivo e nestes sempre em ordem ascendente ou descendente. Perceba-se como o encadeamento é poderoso: para 1 ordem física, providenciaram-se 8 ordens lógicas distintas.

Finalmente, o segundo exercício manda retirar o animal número 3, que vem a ser o pombo chamado binho. Note que ele NÃO é retirado fisicamente e também seus descritores intrínsecos não se alteram, mas após a exclusão este registro DEIXA DE FAZER parte dos encadeamentos. Dois cursores (apontadores) precisam ser alterados para cada atributo que sofre duplo encadeamento. Neste exemplo, como são 4 atributos, a exclusão demandará alterar 4 cursores (já que os cursores do excluído ficam inalterados). Fica

	animal	↑	↓	nome	↑	↓	cor	↑	↓	idade	↑	↓
		4	6		1	2		6	8		6	7
1	bola	8	4	canari	5	0	cinza	2	4	08	5	8
2	mimi	5	7	peixe	0	8	claro	5	1	03	8	4
3	binho	1	4	pomba	0	2	louro	5	2	04	8	2
4	alex	1	0	gato	8	7	branco	1	6	02	2	6
5	neto	6	2	cao	6	1	marrom	7	2	09	7	1
6	piupiu	0	5	cobra	7	5	bege	4	0	01	4	0
7	fifi	2	8	furao	4	6	pardo	8	5	11	0	5
8	feio	7	1	minhoc	2	4	ruivo	0	7	05	1	2

☞ Para você fazer

Nos exercícios a seguir, você receberá um conjunto de registros contendo 4 campos cada um. Deve construir os 8 encadeamentos. Depois disso, deve incluir o novo registro solicitado e depois deve excluir o registro indicado. Para cada exercício escreva a sequência de cursores aqui indicada, sempre sobre a ÚLTIMA tabela gerada.

	↑	↓	↑	↓	↑	↓	↑	↓
	•							
1		•						
2			•					
3				•				
4					•			
5						•		
6							•	
7								•

Carros

Seja um arquivo de carros com os atributos abaixo

	placa	↑	↓	marca	↑	↓	comb	↑	↓	km	↑	↓
1	7001			combi			alc			190		
2	4476			gm			cacha			332		
3	7003			corsa			gasol			230		
4	7002			dkw			canaa			111		
5	8000			fiat			eletr			340		
6	2324			uno			diese			167		
7	6000			jeep			butan			980		
8												

Incluir o registro 2212 vw milho 132

E fica:

	placa	↑	↓	marca	↑	↓	comb	↑	↓	km	↑	↓
1	7001			combi			alc			190		
2	4476			gm			cacha			332		
3	7003			corsa			gasol			230		
4	7002			dkw			canaa			111		
5	8000			fiat			eletr			340		
6	2324			uno			diese			167		
7	6000			jeep			butan			980		
8	2212			vw			milho			132		

E depois, excluir o carro de número 4

e fica

	placa	↑	↓	marca	↑	↓	comb	↑	↓	km	↑	↓
1	7001			combi			alc			190		
2	4476			gm			cacha			332		
3	7003			corsa			gasol			230		
4	7002			dkw			canaa			111		
5	8000			fiat			eletr			340		
6	2324			uno			diese			167		
7	6000			jeep			butan			980		
8	2212			vw			milho			132		

Finalmente, responda os cursores pedidos para este exercício:

head	car=1	car=2	car=3	car=4	car=5	car=6	car=7
pla↑	pla↓	mar↑	mar↓	com↑	com↓	km↑	km↓

livros

Seja um conjunto de livros com os atributos

	autor	↑	↓	titulo	↑	↓	pags	↑	↓	preco	↑	↓
1	giz			licor			403			67.00		
2	ze2			anao			200			33.00		
3	lapis			aril			450			48.00		
4	alba			cafe			341			18.60		
5	ze1			sou			401			70.00		
6	dji			fe			765			18.50		
7												

Incluir o registro xico ceva 980 20.50

e fica:

	autor	↑	↓	titulo	↑	↓	pags	↑	↓	preco	↑	↓
1	giz			licor			403			67.00		
2	ze2			anao			200			33.00		
3	lapis			aril			450			48.00		
4	alba			cafe			341			18.60		
5	ze1			sou			401			70.00		
6	dji			fe			765			18.50		
7	xico			ceva			980			20.50		

E depois, excluir o livro de número 2

E fica:

	autor	↑	↓	titulo	↑	↓	pags	↑	↓	preco	↑	↓
1	giz			licor			403			67.00		
2	ze2			anao			200			33.00		
3	lapis			aril			450			48.00		
4	alba			cafe			341			18.60		
5	ze1			sou			401			70.00		
6	dji			fe			765			18.50		
7	xico			ceva			980			20.50		

Finalmente, responda os cursores pedidos para este exercício:

head	liv=1	liv=2	liv=3	liv=4	liv=5	liv=6	liv=7
aut↑	aut↓	tit↑	tit↓	pag↑	pag↓	pre↑	pre↓



Equações diferenciais

Uma hipótese bastante plausível é que o crescimento de uma população varia a uma taxa proporcional a seu tamanho. Por exemplo, Curitiba pode ter uma taxa de de 35.000 crianças/ano (34.258 em 2022) enquanto uma cidade menor como Astorga pode ter taxa de 350 nascimentos/ano (329 em 2022). Neste exemplo a primeira taxa é 100 vezes maior que a segunda. As populações devem ter uma proporcionalidade semelhante.

Ao modelar algebricamente esse fenômeno, pode-se escrever que a variação α é proporcional à população, ou $\alpha = kP$. Mas a taxa de variação nada mais é que a derivada da população em relação ao tempo ou $\alpha = \frac{dP}{dt}$ e desta maneira a equação original fica como $\frac{dP}{dt} = kP$.

Este é um exemplo acabado de equação diferencial: trata-se de uma equação que envolve as operações algébricas E operações de diferenciação (derivação).

A maneira de resolver esta equação é

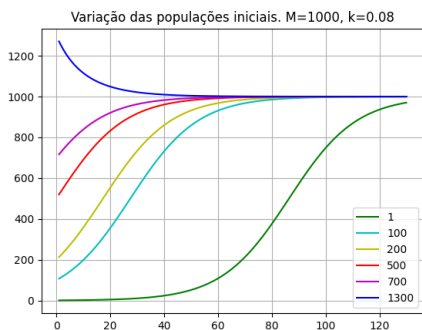
$$\begin{aligned} \int \frac{dP}{P} &= \int k dt \text{ que é} \\ \ln|P| &= kt + C \text{ ou} \\ |P| &= e^{kt+C} = e^C e^{kt} \text{ ou} \\ P &= Ae^{kt} \end{aligned}$$

A equação acima é chamada Lei do crescimento natural. Nela, se k é positivo a população cresce e se k é negativo ela diminui. Como a equação é separável, ela pode ser facilmente resolvida, como o foi.

Para obter mais realismo, vamos incluir uma capacidade de suporte M que indica quando a população se aproxima de um limite de suporte (basicamente comida, ou espaço, ou qualquer coisa parecida). Ele indica a população máxima que o ambiente é capaz de sustentar a longo prazo. A expressão que agrega esta hipótese é

$$\begin{aligned} \frac{dP}{dt} &= kP \left(1 - \frac{P}{M}\right) \\ \frac{1}{P} \frac{dP}{dt} &= k \left(1 - \frac{P}{M}\right) \\ \frac{dP}{P(1-P/M)} &= k dt \\ \int \left(\frac{1}{P} + \frac{1}{M-P}\right) dP &= \int k dt \\ \frac{M-P}{P} &= Ae^{-kt} \\ \frac{M}{P} &= \frac{M}{1+Ae^{-kt}} \quad A = \frac{M-P_0}{P_0} \end{aligned}$$

Observe como esta função se comporta em um exemplo com $M = 1000$ e $k = 0.08$ (Note que k sinaliza quão rapidamente a população tende ao limite M).



Construído pelo programa

```
k=0.08; M=1000
def P(t,M,P0,k):
    A=(M-P0)/P0
    return M/(1+A*np.exp(-k*t))
t=np.arange(1,130,0.1)
plt.plot(t,P(t,M,1,k),'g-',label='1')
plt.plot(t,P(t,M,100,k),'c-',label='100')
plt.plot(t,P(t,M,200,k),'y-',label='200')
plt.plot(t,P(t,M,500,k),'r-',label='500')
plt.plot(t,P(t,M,700,k),'m-',label='700')
plt.plot(t,P(t,M,1300,k),'b-',label='1300')
plt.title('Variação... M=1000, k=0.08')
```

Deve-se lembrar que em uma equação diferencial a resposta procurada é uma função ou uma família de funções (ao contrário de uma equação algébrica onde se acha como solução um número ou um conjunto de números). Vamos estudar mais um exemplo

Seja resolver a equação $y'' + 2y' = 3y$. Veja que esta equação pode ser escrita de maneiras alternativas: $f''(x) + 2f'(x) = 3f(x)$ ou ainda $\frac{d^2y}{dx^2} + 2\frac{dy}{dx} = 3y$. Ainda sem saber como resolvê-la vamos estudar uma possível solução achada na cartola do mágico: Se $y_1(x) = e^{-3x}$, então $y_1'(x) = -3e^{-3x}$ e $y_1''(x) = 9e^{-3x}$ e se você fizer as substituições verá que $y_1(x) = e^{-3x}$ é uma solução. Mas, $y_2 = e^x$ também é uma solução para a mesma equação diferencial.

Já uma equação diferencial linear como

$$\frac{dy}{dx} = -2x + 3y - 5$$

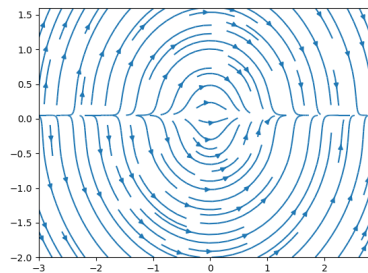
tem como solução uma reta cuja função é $y = mx + b$, restando a tarefa de achar m e b . Após manipular a expressão, acha-se $m = \frac{2}{3}$ e $b = \frac{17}{9}$ dando como resultado $y = \frac{2}{3}x + \frac{17}{9}$.

Campos vetoriais

Suponha a equação diferencial $\frac{dy}{dx} = \frac{-x}{y}$. Mesmo sem resolvê-la, vamos plotar diversos pontos em uma grade cartesiana, para ver o que acontece. Veja

```
x = np.arange(-3, 3, .45)
y = np.arange(-2, 2, .45)
X, Y = np.meshgrid(x, y)
dy = (-X)/Y
dx = np.ones(dy.shape)
plt.quiver(X,Y,dx, dy,scale=100)
plt.streamplot(X,Y,dx,dy)
```

com este resultado



Tipos Uma equação diferencial é ordinária se ele contém apenas 1 variável independente (" x ") e parcial se ela tem mais de 1 variável. A ordem de uma equação diferencial é o número da derivada de mais alta ordem. O grau de uma equação diferencial é o maior expoente da derivada de maior ordem que aparece na equação. Uma solução particular é uma função que resolve a equação, através da designação de valores particulares para as constantes arbitrárias. Já a solução geral apresenta todas as funções que resolvem a equação. Uma equação diferencial é linear quando a ordem da derivada mais alta é 1.

Solucionando O primeiro e mais simples caso é o de equações separáveis. A ideia é juntar antes do igual tudo o que se refere a y (incluindo o dy) e do outro lado tudo o que se refere a x e dx . Acompanhe $A(x)dx + B(y)dy = 0$ que gera $\int A(x)dx + \int B(y)dy = C$, onde C é uma constante arbitrária. Por exemplo, seja resolver a equação $\frac{dy}{dx} = \frac{x^2+2}{y}$ Reescrevendo fica $(x^2+2)dx - ydy = 0$ e cuja solução é $\int (x^2 + 2)dx - \int ydy = C$ ou $\frac{1}{3}x^3 + 2x - \frac{1}{2}y^2 = C$

Já a equação diferencial linear tem o formato $y' + p(x)y = q(x)$. Aqui admite-se um fator de integração igual a $I(x) = e^{\int p(x)dx}$ e deve-se multiplicar os dois lados por $I(x)$ e a solução geral é $y = \frac{\int I(x)q(x)dx + c}{I(x)}$. Veja-se um exemplo: Resolver $y' + (4/x)y = x^4$. Aqui $p(x) = 4/x$ e $q(x) = x^4$ e ela é linear. Daí $\int p(x)dx = \int \frac{4}{x}dx = 4 \ln|x| = \ln x^4$ e portanto $I(x) = e^{\int p(x)dx} = e^{\ln x^4} = x^4$. Multiplicando a equação original pelo fator de integração fica $x^4 y' + 4x^3 y = x^8$ ou $\frac{d}{dx}(y x^4) = x^8$ Integrando os dois lados da equação em relação a x obtem-se $y x^4 = \frac{1}{9}x^9 + C$ ou $y = \frac{c}{x^4} + \frac{1}{9}x^5$.

Equação dif. linear homogênea de 2. ordem

Seja a equação $y'' + 5y' + 6y = 0$. Supondo que $y = e^{rx}$ obtem-se as derivadas: $y = e^{rx}$, $y' = r e^{rx}$ e $y'' = r^2 e^{rx}$ e pode-se reescrever a função original como $r^2 e^{rx} + 5r e^{rx} + 6e^{rx} = 0$. Colocando e^{rx} em evidência, fica $e^{rx}(r^2 + 5r + 6) = 0$. Ora em um produto igual a zero qualquer dos fatores deve ser 0. Desconsiderando o primeiro, resolve-se $(r^2 + 5r + 6) = 0$ e por Baskara, obtem-se as duas raízes $r_1 = -2$ e $r_2 = -3$. Daqui. $y_1 = C_1 e^{-2x}$ e $y_2 = C_2 e^{-3x}$ ou pode-se reescrever a solução da equação diferencial original como $y(x) = C_1 e^{-2x} + C_2 e^{-3x}$.

Se quisermos uma solução particular (matando C_1 e C_2) precisamos condições iniciais. Suponhamos que o enunciado informe que $y(0) = 2$ e que $y'(0) = 3$. Substituindo x por 0 na equação original e igualando a 2, fica $2 = C_1 + C_2$. Agora derivamos a solução geral e fica $y'(x) = -2C_1 e^{-2x} - 3C_2 e^{-3x}$. Substituindo x por 0 e resolvendo fica $3 = -2C_1 - 3C_2$. Resolvendo este sistema linear (pelo método que se quiser) obtêm-se $C_2 = -7$ e $C_1 = 9$ e a solução particular desejada é $y(x) = 9e^{-2x} - 7e^{-3x}$.

Solução numérica - método de Euler

O método de Euler ganha importância associado a computadores, já que ele não resolve simbolicamente a equação, mas descobre simulações que podem se aproximar tanto quanto se queira da função buscada. Por exemplo, ao resolver $\frac{dy}{dx} = y$, com $y(0) = 1$ constrói-se uma tabela com $\delta x = 1$ onde

0	1	1
1	2	2
2	4	4
3	8	8

Construindo a mesma tabela, com $\delta x = 0.5$ a mesma tabela fica

0	1	1
1	2	2
0.5	1.5	1.5
1	2.25	2.25

e assim por diante. Este é o método por excelência dos pacotes computacionais.

Usando sympy

Há um capítulo enorme no sympy para equações diferenciais. Por falta de espaço, vamos nos limitar a um exemplo. Seja resolver $y'' - 2y' + y = \sin(x)$

```
import sympy as s
s.var('x')
f=s.symbols('f',cls=s.Function)
ed=s.Eq(f(x).diff(x,x)-2*f(x).diff(x)+f(x),s.sin(x))
print(ed)
print(s.dsolve(ed,f(x)))
```

```
Eq(f(x) - 2*Derivative(f(x), x) + \
    Derivative(f(x), (x, 2)), sin(x))
Eq(f(x), (C1 + C2*x)*exp(x) + cos(x)/2)
```

Para você fazer

Resolva as seguintes séries conforme pedido. Para cada uma, calcule 4 parcelas (portanto até $f'''(x)$) e depois calcule $f(x)$ no ponto pedido.

- $xy' + 2y = \sin(x)$, $y(\pi/2) = 1$
- $y' + 2y = xe^{-2x}$, $y(1) = 0$
- $y' = y^2$, $y(0) = 1$

Responda aqui:

1	2	3



- 1 -

$$1= \$y=x^{-2}(\pi^2/4-1-x\cos x+\operatorname{se} \$y=(1/2)(x^2-1)e^{-2x} \$$$

$$\$y=(1-x)^{-1} \$$$

Listagem de todos os problemas de 351

$$1. y' - y = 2xe^{2x}, y(0) = 1 \quad 3e^x + 2(x-1)e^{2x}$$

$$2. y' + 2y = xe^{-2x}, y(1) = 0 \quad y = (1/2)(x^2 - 1)e^{-2x}$$

$$3. x^2y' + 2xy = \cos(x), y(\pi) = 0 \quad y = x^{-2}\operatorname{sen}(x)$$

$$4. xy' + 2y = \operatorname{sen}(x), y(\pi/2) = 1 \quad y = x^{-2}(\pi^2/4 - 1 - x\cos x + \operatorname{sen} x)$$

$$5. y' = x + y, y(0) = 1 \quad y = 2e^x - x - 1$$

$$6. \cos(x)y' - \operatorname{sen}(x)y = 1, y(2\pi) = \pi \quad y = \frac{x-\pi}{\cos x}$$

$$7. y' = y^2, y(0) = 1 \quad y = (1-x)^{-1}$$

$$8. y' = \frac{x^2}{y(1+x^3)}, y(0) = 1 \quad y = (1 + 2/3)\ln(1 + x^3))^{1/2}$$

$$9. y' = \frac{3x^2+4x+2}{2y-2}, y(0) = -1 \quad y = 1 - \sqrt{x^3 + 2x^2 + 2x + 4}$$

$$10. y' = \frac{y \cos(x)}{1+2y^2}, y(0) = 1 \quad \ln|x| + y^y = 1 + \operatorname{sen} x$$

$$11. y' = \frac{2x}{y+x^2y}, y(0) = -2 \quad y = -(2\ln(1+x^2) + 4)^{1/2}$$

$$12. y' = \frac{y^3}{1-2xy^2}, y(0) = 1 \quad xy^2 - \ln|y| = 0$$

$$1= \$4u(t)\$$$

$$\$3e^{-5t}\$$$

$$\$s/(s^2+(1/4))\$$$

Listagem de todos os problemas de 352

1. Ache a Transformada de Laplace de

$$5 \operatorname{sen}(3t)$$

$$15/(s^2 + 9)$$

2. Ache a Transformada de Laplace de

$$\cos\left(\frac{t}{2}\right)$$

$$s/(s^2 + (1/4))$$

3. Ache a Transformada de Laplace de

$$\frac{t^4}{3}$$

$$8/s^5$$

4. Ache a transformada de Laplace de

$$\frac{1}{2}e^{-5t}$$

$$1/(2(s + 5))$$

5. Ache a transformada de Laplace inversa de

$$\frac{1}{s - 4}$$

$$e^{4t}$$

6. Ache a transformada de Laplace inversa de

$$\frac{3}{s + 1}$$

$$3e^{-t}$$

7. Ache a transformada de Laplace inversa de

$$\frac{4}{s}$$

$$4u(t)$$

8. Ache a T.L. usando as propriedades da transformada de

$$te^{-3t}$$

$$1/(s + 3)^2$$

9. Resolva a equação diferencial

$$y' + 5y = 0 \text{ com } y(0) = 3$$

usando a Transformada de Laplace $3e^{-5t}$

10. Resolva a equação diferencial

$$y' + y = t \text{ com } y(0) = 0$$

usando a Transformada de Laplace $-1 + t + e^{-t}$

11. Resolva a equação diferencial

$$y'' + 4y = 9t$$

com $y(0) = 0$ e $y'(0) = 7$ usando a Transformada de Laplace $19\operatorname{sen}(2t)/8 + (9t/4)$

Taxas de variação relacionadas

Regra da cadeia Precisamos relembrar uma regra importante do cálculo que nos habilita a obter a derivada de uma função composta. Um exemplo de função composta é $\cos(\sin(x))$. Primeiro note que isto NÃO é a mesma coisa que $\cos(\sin(x))$. Este segundo caso é um produto de funções e não sua composição.

Para entender a diferença, vamos imaginar um caso simples $f(x) = x + 1$ e $g(x) = 2x$. $f(g(3)) = 7$ enquanto $f(3)g(3) = 24$. Ainda, $g(f(3)) = 8$.

Voltando ao exemplo, primeiro calculamos o $\sin(x)$ (chamada função interna) e depois calculamos o $\cos$ deste valor (e não de x). Esta segunda função é chamada externa.

A regra da cadeia nos ensina a calcular a derivada de uma função composta e é

$$\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x)$$

Vamos a um exemplo: Para calcular a derivada de $y = \ln(\sin(x))$, precisamos determinar quem é quem: $f(x) = \ln(x)$ e $g(x) = \sin(x)$. Daqui $f'(x) = \frac{1}{x}$ e $g'(x) = \cos(x)$. Agora aplica-se a regra da cadeia e fica $y' = \frac{1}{\sin(x)} \cdot \cos(x)$ e finalmente $y' = \frac{\cos(x)}{\sin(x)}$

Regra do produto Voltando ao exemplo anterior, quando se tem $y = f(x) \cdot g(x)$ e se precisa obter y' , usa-se a regra do produto que informa

$$y = f(x) \cdot g(x) \quad e \quad y' = f'(x) \cdot g(x) + f(x) \cdot g'(x)$$

Ou usando um exemplo, seja $y = \ln(x) \cdot \sin(x)$ e precisa-se obter a sua derivada. Fazendo:

$$y = \underbrace{\ln(x)}_{f(x)} \cdot \underbrace{\sin(x)}_{g(x)}$$

tem-se

$$y' = \underbrace{\frac{1}{x}}_{f'(x)} \cdot \underbrace{\sin(x)}_{g(x)} + \underbrace{\ln(x)}_{f(x)} \cdot \underbrace{\cos(x)}_{g'(x)}$$

$$\text{Fica } y' = \frac{1}{x} \cdot \sin(x) + \ln(x) \cdot \cos(x).$$

Taxas relacionadas Vamos introduzir o tema usando um exemplo tirado da Khan Academy (pt.khanacademy.org). Seja um plácido lago num dia sem vento. Você joga uma pedra no meio e percebe a formação de uma onda circular que aumenta com o passar do tempo. Começa-se definindo que o raio do círculo aumenta a 1cm/seg . A pergunta é qual a taxa de variação da área do círculo, QUANDO o raio vale 3cm ? Dando nomes às coisas: $\frac{dr}{dt} = 1\text{cm/seg}$ e o que se quer é $\frac{dA}{dt}$. Área e raio estão relacionados pela velha equação $A = 2\pi r^2$. Só que o que se necessita é $\frac{dA}{dt}$ e não $\frac{dA}{dr}$. Para resolver aqui usa-se a regra da cadeia acima relembrada:

$$\frac{dA}{dt} = \frac{d(\pi r^2)}{dt} = \frac{d(\pi r^2)}{dr} \cdot \frac{dr}{dt}$$

e agora fica fácil substituir:

$$\frac{dA}{dt} = \pi \cdot 2 \cdot r \cdot \frac{dr}{dt} = \pi \cdot 2 \cdot 3\text{cm} \cdot 1 \frac{\text{cm}}{\text{s}} = 6\pi \frac{\text{cm}^2}{\text{s}}$$

Interpretando este resultado, sabe-se que QUANDO o raio vale 3cm a área está crescendo a cerca de 18.84 cm^2 por segundo.

Um exemplo simples Um quadrado está diminuindo de tamanho. Seu lado diminui à taxa de 2km/h . No instante em que o lado vale 9km , pergunta-se qual a taxa de variação da área?

A solução começa ao relacionar lado p e área A pela equação $A = p^2$. Os dados fornecidos são $\frac{dp}{dt} = -2\text{km/h}$ e $p_0 = 9\text{km}$. A pergunta a responder é qual o valor de $\frac{dA}{dt}$. Derivando $A = p^2$ em relação ao tempo fica $\frac{dA}{dt} = 2p \frac{dp}{dt}$. O lado esquerdo da igualdade é a nossa pergunta. Do lado direito é só substituir e fica: $2 \cdot 9 \cdot -2 \text{ km}^2/\text{h}$ e a resposta é $-36\text{km}^2/\text{h}$.

Mais um Mais um exemplo (também tirado da khan academy): Imagine um triângulo que está tendo seus lados alterados. A sua base $b(t)$ diminui a uma taxa de 13m/hr e a altura do triângulo $h(t)$ aumenta a uma taxa de 6m/hr . Em um certo instante t_0 a base tem 5m e a altura tem 1m . Pergunta-se qual a taxa de variação da área $A(t)$ do triângulo nesse instante? O que se sabe da definição é que $b(t)$ está dada em m e que $b'(t)$ está em m/hr e ainda $b'(t) = \frac{db}{dt} = -13 \frac{m}{hr}$. Quanto a h sabe-se que $h(t)$ está dada em m e que $h'(t)$ está dada em m/hr . $h'(t) = 6\text{m/hr}$. O problema ainda informa que no instante t_0 , $b(t_0) = 5\text{m}$ e que $h(t_0) = 1\text{m}$.

É hora de abrir a caixa de ferramentas do matemático. Lá está a fórmula salvadora da área de um triângulo que é $A = \frac{1}{2} \cdot b \cdot h$. Generalizando esta fórmula em relação ao tempo, fica $A(t) = \frac{1}{2} b(t) \cdot h(t)$. Mas, como estamos estudando variações, precisamos derivar em relação ao tempo lembrando a regra do produto de funções vista acima, fica $A'(t) = \frac{1}{2} [b'(t) \cdot h(t) + b(t) \cdot h'(t)]$. Como nosso problema se enfoca no tempo t_0 , fica $A'(t_0) = \frac{1}{2} [b'(t_0) \cdot h(t_0) + b(t_0) \cdot h'(t_0)]$ e agora é só substituir, ficando $A'(t_0) = \frac{1}{2} [-13 \times 1 + 5 \times 6] = 8.5\text{m}^2/\text{h}$, que é a resposta procurada.

Mais um exemplo Seja uma escada de 13 metros encostada em uma parede. O pé da escada se afasta da parede à taxa de 1m/min . Quando o pé da escada está a 5m da parede a qual taxa de descida vem o topo da escada? Eis uma solução: Seja $h(t)$ a altura do topo da escada no tempo t e seja $b(t)$ a distância do pé da escada no tempo t . O Teorema de Pitágoras nos diz que $h(t)^2 + b(t)^2 = 13^2$. Quando derivamos os dois lados desta expressão em relação a t medido em minutos obtem-se

$$2 \cdot h(t) \cdot h'(t) + 2 \cdot b(t) \cdot b'(t) = 0.$$

Resolvendo para $h'(t)$ tem-se

$$h'(t) = -\frac{b(t) \cdot b'(t)}{h(t)}$$

No instante em que o problema ocorre, $b(t) = 5$, $h(t) = 12$ (Pelo Teorema de Pitágoras). Substituindo estes valores na fórmula, fica

$$h'(t) = -\frac{5 \cdot 1}{12} = -\frac{5}{12} = -0.41\text{m/min}$$

Mais um Uma bexiga redonda tem a forma de uma esfera. Ela está sendo inchada na taxa de $20\text{polegadas}^3/\text{min}$. No momento em que a esfera tem 64pol^3 qual a taxa de mudança do raio?

Sabe-se que o volume e o raio estão relacionados pela fórmula

$$V = \frac{4\pi}{3} r^3$$

A variável livre neste problema é o tempo t , assim deve-se diferenciar a equação acima em relação a t . Já que não estamos diferenciando em relação a r se necessita usar a regra da cadeia com o resultado

$$\frac{dV}{dt} = \frac{4\pi}{3} \cdot 3r^2 \cdot \frac{dr}{dt}$$

Sabe-se que $dV/dt = 20$. Nossa questão é quando o volume é $V = 64$. De acordo com a fórmula inicial sabe-se que $r = \sqrt[3]{48/\pi}$. Substituindo estes valores na segunda equação tem-se

$$20 = \frac{4\pi}{3} \cdot 3 \left[\sqrt[3]{48/\pi} \right]^2 \cdot \frac{dr}{dt}$$

Resolvendo para dr/dt fica

$$\frac{dr}{dt} = \frac{5}{48^{2/3} \cdot \pi^{1/3}} = 0.258\text{pol/min}$$

Mais um Uma esponja no formato de um cone circular. À medida em que ela recebe água, ela cresce em tamanho. Em um dado momento, ela tem 6m de altura e está escada crescendo à razão de 0.3m/s . No mesmo momento, o raio é de 4m e está crescendo à razão de 0.2m/s . A pergunta é a respeito da taxa de mudança do volume nesse mesmo instante.



Sabe-se que o volume V de um cone cilíndrico é dada pela fórmula

$$V = \frac{1}{3} \pi r^2 \cdot h$$

Derivando ambos os lados em relação a t , fica

$$\frac{dV}{dt} = \frac{1}{3} \pi \left[2r \frac{dr}{dt} h + r^2 \cdot \frac{dh}{dt} \right]$$

Substituindo os valores de r , $\frac{dr}{dt}$, h e $\frac{dh}{dt}$ no lado direito, fica

$$\frac{dV}{dt} = \frac{1}{3} \pi \left[2 \cdot 4 \cdot (0,2) \cdot 6 + 4^2 \cdot (0,3) \right] =$$

$$\frac{1}{3} \pi [9,6 + 4,8] = \frac{24\pi}{5} \text{ m}^3/\text{s}$$

Para você fazer

- Uma escada de 6m de comprimento está apoiada em uma parede vertical. Se a base da escada começa a escorregar horizontalmente à taxa constante de $0,6\text{m/s}$, com que velocidade o topo da escada percorre a parede quando ele está a 4m do solo.
- Uma pessoa que solta um papagaio segura a corda a $1,5\text{m}$ do solo. A corda é liberada à razão de $0,6\text{m/s}$ na medida em que o papagaio se move horizontalmente a uma altura de $33,5\text{m}$. Supondo que a corda fique sempre tensa, determine a taxa à qual o papagaio está se movendo no instante em que foram liberados 38m de corda.
- Suponha que uma bola de neve esteja se derretendo, com raio decrescendo à razão constante, passando de 30cm para 20cm em 45 minutos. Qual a variação do volume quando o raio está com 25cm .

Responda aqui:

1	2	3



- 1 -

Listagem de todos os problemas de 353

1. Duas variáveis x e y são funções de uma variável t e estão relacionadas pela equação:

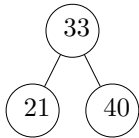
$$y^2 - 3xy + x^2 = 25$$

Se a taxa de variação de x em relação a t é igual a 1 quando $x = 0$ então determine qual a taxa de variação de y em relação a t neste mesmo instante. $3/2$

2. Uma escada de 6m de comprimento está apoiada em uma parede vertical. Se a base da escada começa a escorregar horizontalmente à taxa constante de $0,6 \text{ m/s}$, com que velocidade o topo da escada percorre a parede quando ele está a 4m do solo. $-\frac{3\sqrt{5}}{10} \text{ m/s}$
3. As 8h o navio A está a 25km ao sul do navio B. Se o navio A está navegando para o oeste à 16 km/h e o navio B está navegando para o sul a 20 km/h então determine a razão em que a distância entre os navios está variando às 8h30min. $-\frac{172}{17} \text{ km/h}$
4. Um tanque tem a forma de um cone circular reto invertido, com 4m de altura e raio da base 2m. Se a água entra no tanque à razão de $0,001 \text{ m}^3/\text{min}$, calcule a razão em que o nível de água está subindo quando a altura é 1m. $\frac{4}{1000\pi} \text{ m/min}$
5. Um farol giratório completa uma volta a cada 15 segundos. O farol está a 60m de P, o ponto mais próximo em uma praia retílinea. Determine a razão em que um raio de luz do farol está se movendo ao longo da praia em um ponto, Q, a 150m de P. $3480\pi \text{ m/min}$
6. Um painel solar de 3m de comprimento equipado com um ajustador hidráulico é colocado de forma inclinada sobre um edifício. A medida que o sol se move o painel é ajustado automaticamente de forma que os raios solares sempre incidam de maneira perpendicular a ele de modo a maximizar a captação de energia. a) determine a relação entre a taxa dy/dt à qual o painel deve ser movido e a taxa $d\theta/dt$ à qual o ângulo de inclinação dos raios aumenta. b) Se quando $\theta = 30^\circ$, $d\theta/dt = 15 \text{ graus/h}$ determine dy/dt $-3\sin\theta$ e $-\pi/8$
7. Ao ser aquecida uma chapa circular de metal, seu diâmetro varia à razão de $0,01 \text{ cm/min}$. Determine a taxa à qual a área de uma das faces varia quando o diâmetro é 30cm. $0.15\pi \text{ cm}^2/\text{min}$
8. Suponha que uma bola de neve esteja se derretendo, com raio decrescendo à razão constante, passando de 30cm para 20cm em 45 minutos. Qual a variação do volume quando o raio está com 25cm. $-\frac{5000\pi}{9} \text{ cm}^3/\text{min}$
9. Uma luz está no alto de um poste de 5m. Um menino de 1,6m se afasta do poste em linha reta à razão de $1,2 \text{ m/s}$. A que taxa se move a ponta de sua sombra quando ele está a 6m do poste? A que taxa aumenta o comprimento de sua sombra? $1,764 \text{ m/s}$ e $0,564 \text{ m/s}$
10. A areia que vaza de um depósito forma uma pilha cônica cuja altura é sempre igual ao raio da base. Se a altura da pilha aumenta à razão de 15 cm/min determine a taxa à qual a areia está se escoando quando a altura da pilha é 25cm. $9375\pi \text{ cm}^3/\text{min}$
11. Uma pessoa que solta um papagaio segura a corda a 1,5m do solo. A corda é liberada à razão de $0,6 \text{ m/s}$ na medida em que o papagaio se move horizontalmente a uma altura de 33,5m. Supondo que a corda fique sempre tensa, determine a taxa à qual o papagaio está se movendo no instante em que foram liberados 38m de corda. $1,112 \text{ m/s}$
12. Um balão de ar quente sobe verticalmente à medida que uma corda, amarrada à sua base, é liberada à razão de 1 m/min . O carretel que libera a corda está a 6,5m da plataforma de embarque dos passageiros. A que taxa o balão está subindo quando tiverem sido liberados 150m de corda? 1 m/min
13. Da beira de um rochedo 60m acima de um lago, um menino deixa cair um pedra e depois de 2s deixa cair outra pedra da mesma posição. Discuta a taxa na qual a distância entre as pedras varia durante o próximo segundo (Admita que a distância percorrida em t segundos por um objeto em queda livre é $4,9t^2 \text{ m}$). $19,6 \text{ m/s}$
14. Um míssil é lançado verticalmente para cima de um ponto que está a 8km de uma estação de rastreamento, e à mesma altura desta. Durante os primeiros 20 segundo de vôo, seu ângulo de elevação θ varia à razão constante de 2 graus por segundo. Determine a velocidade do míssil quando o ângulo de elevação for 30 graus. $\frac{32\pi}{270} \text{ km/s}$
15. Um avião está a uma velocidade constante de 580 Km/h e subindo a um ângulo de 45 graus. No momento em que ele está a uma altura de 3,2km, passa diretamente sobre um torre de controle no solo. Ache a taxa de variação da distância do avião à torre um minuto mais tarde. (Despreze a altura da torre). $569,65 \text{ km/h}$
16. Um meliante foge sobre uma muralha reta a uma velocidade de 4 m/s . Um holofote localizado a 20 metros de distância da muralha, e mesma altura que esta, focaliza o homem em fuga. A que taxa o holofote está girando quando o gatuno se encontra a 15 metros do ponto da muralha que está mais próximo do holofote? $\frac{16}{125} \text{ rad/s}$

Representação e caminhamento em árvores binárias

Árvores binárias podem ser representadas de diversas maneiras. Neste exercício você trabalhará com 3 representações distintas. As 3 árvores a seguir estão "em ordem" (ou in-order). Lembrando, neste tipo de ordenamento, a raiz sempre é um valor intermediário, maior que o filho esquerdo e menor que o filho direito. Por exemplo:



Lembre que esta propriedade se estende para todos os nodos da árvore binária. Depois que a árvore binária foi desenhada a partir dos dados fornecidos, podem ser obtidos, ao menos, 4 caminhamentos:

Pré-ordem RAIZ-FilhoEsfquerdo-FilhoDireito. O exemplo, na árvore acima é: 33,21,40.

Em-ordem FilhoEsquerdo-RAIZ-FilhoDireito. Exemplo: 21,33,40.

Pós-ordem FilhoEsquerdo-FilhoDireito-RAIZ. Exemplo: 21,40,33.

Largura Como se a árvore fosse cortada em fatias. Também conhecido como caminhamento horizontal. Exemplo: 33,21,40.

Árvore usando cursores

A primeira árvore (árvore a), vai estar representada em uma matriz de 12 linhas por 3 colunas. Cada linha da matriz representa um nodo. A raiz sempre estará na primeira posição da matriz. A primeira coluna representa o endereço do filho esquerdo, a coluna 2 o endereço do filho direito e a coluna 3 o conteúdo do nodo. O terminador é representado por -1.

1	2	5	11
2	4	3	6
3	-1	-1	7
4	10	8	4
5	6	-1	20
6	7	-1	16
7	-1	9	13
8	-1	-1	5
9	-1	-1	15
10	11	12	2
11	-1	-1	1
12	-1	-1	3

Caminhamentos obtidos

```
pré-ordem:  11  6  4  2  1  3  5  7 20 16 13 15
em-ordem:   1  2  3  4  5  6  7 11 13 15 16 20
pós-ordem:  1  3  2  5  4  7  6 15 13 16 20 11
```

Árvore usando alocação absoluta

A segunda árvore (b) está representada por uma memória M onde o apontador da raiz da árvore está em $M[0]$ e cada nodo tem o formato

&fe	&fd	conteúdo								
	0	1	2	3	4	5	6	7	8	9
0	3	12	21	24	6	35	9	13	75	19
1	60	61	42	30	16	85	-1	-1	91	27
2	34	55	18	94	37	40	22	44	-1	50
3	-1	-1	79	59	-1	-1	60	67	-1	15
4	-1	57	25	59	-1	47	37	50	-1	48
5	64	54	39	85	-1	-1	46	-1	-1	28
6	-1	-1	74	71	-1	-1	38	-1	-1	8
7	22	63	68	13	85	64	90	18	53	71
8	71	50	66	80	37	26	2	31	28	78
9	83	32	13	51	32	62	10	4	27	58

Caminhamentos obtidos

```
pré:35 22 15 8 25 28 75 61 55 50 37 48 39 38 46 60 74 85 79 91
em : 8 15 22 25 28 35 37 38 39 46 48 50 55 60 61 74 75 79 85 91
pós: 8 15 28 25 22 38 46 39 48 37 50 60 55 74 61 79 91 85 75 35
```

Árvores usando parênteses

A terceira árvore (árvore c) está representada por parênteses, na forma (raiz (filhoesquerdo) (filhodireito)). Note que (x ()) aparece como (x)

$$(4(2(1)(3))(23(14(7(())(12(())(13))))(16(())(21(18)(())))(29(25(())(28))(())))$$

Caminhamentos obtidos

```
pré-ordem: 4 2 1 3 23 14 7 12 13 16 21 18 29 25 28
em-ordem: 1 2 3 4 7 12 13 14 16 18 21 23 25 28 29
pós-ordem: 1 3 2 13 12 7 18 21 16 14 28 25 29 23 4
```

 Para você fazer

Árvore A

1	2	4	15
2	3	7	7
3	-1	5	2
4	6	-1	20
5	8	-1	5
6	-1	12	18
7	9	-1	13
8	-1	-1	3
9	10	-1	12
10	-1	11	9
11	-1	-1	11
12	-1	-1	19

Árvore B

[illegible]

Árvore C

(2(1)(25(5()(22(14(10(7)(13(12(1
1)())())(15)))(23))) (28(26)()))))

Agora resolva as questões abaixo. Cada pergunta solicita uma árvore (a, b ou c), um caminhamento (pr, in ou po) e um determinado elemento nesse caminhamento em especial.

a-pr[7]	a-in[10]	a-po[8]	b-pr[13]	b-in[14]
b-po[11]	c-pr[9]	c-in[14]	c-po[13]	

1





Para você fazer

Suponha a seguinte matriz A

1	-1	2	-1	3	-1	4	14
2	-1	-1	5	-1	-1	-1	44
3	6	7	-1	-1	-1	8	38
4	-1	-1	9	-1	-1	-1	23
5	-1	10	-1	-1	-1	-1	33
6	-1	11	-1	-1	-1	-1	91
7	12	-1	13	-1	14	-1	54
8	15	-1	16	-1	-1	17	25
9	18	19	-1	-1	-1	20	20
10	-1	21	-1	-1	-1	22	82
11	-1	-1	-1	-1	-1	23	92
12	-1	-1	-1	-1	-1	-1	5
13	-1	-1	-1	-1	-1	-1	94
14	-1	-1	-1	-1	24	-1	90
15	-1	-1	-1	-1	-1	-1	40
16	-1	-1	-1	-1	-1	-1	79
17	-1	-1	-1	-1	-1	-1	80
18	-1	-1	-1	-1	-1	-1	55
19	-1	-1	-1	-1	-1	-1	87
20	-1	25	26	27	28	-1	29
21	-1	-1	-1	-1	-1	-1	24
22	29	-1	30	-1	-1	-1	32
23	-1	-1	-1	-1	-1	-1	93
24	-1	-1	-1	-1	-1	-1	56
25	-1	-1	-1	-1	-1	-1	41
26	-1	-1	-1	-1	-1	-1	2
27	-1	-1	-1	-1	-1	-1	73
28	-1	-1	-1	-1	-1	-1	21
29	-1	-1	-1	-1	-1	-1	64
30	-1	-1	-1	-1	-1	-1	95

- Quantos nodos estão no nível 2 da árvore representada pela matriz A ?

R: _____.

- Usando a equivalência estudada acima, desenhe a árvore binária equivalente a árvore da matriz A e responda:

Quais os nodos que estão no nível 10 da árvore ?

R: _____.

- Represente a árvore obtida na etapa 2 acima através de uma matriz R de 30 linhas por 3 colunas. Note que a coluna 3 é igual à coluna 7 da matriz A. Preencha as colunas 1 e 2 de R e responda:

R[26][2] é _____ e R[22][1] é _____.

1



- 1 - /

Conversão de árvores n-árias em binárias

Suponha uma árvore qualquer com grau igual a 6, e representada por uma matriz de 7 colunas. As seis primeiras colunas representam o endereço dos filhos, enquanto a coluna 7 representa o conteúdo do nó. Lembrar que -1 é terminador.

Por exemplo, seja a seguinte árvore desse tipo

1	2	-1	-1	-1	-1	-1	2
2	3	-1	-1	-1	-1	-1	28
3	4	5	-1	-1	-1	-1	64
4	-1	-1	-1	-1	-1	-1	55
5	6	7	-1	-1	-1	-1	91
6	-1	-1	-1	-1	-1	-1	27
7	8	9	10	-1	-1	-1	97
8	-1	-1	-1	-1	-1	-1	25
9	-1	-1	-1	-1	-1	-1	84
10	11	12	-1	-1	-1	-1	69
11	-1	-1	-1	-1	-1	-1	46
12	13	-1	-1	-1	-1	-1	81
13	14	15	16	17	-1	-1	3
14	-1	-1	-1	-1	-1	-1	16
15	-1	-1	-1	-1	-1	-1	70
16	-1	-1	-1	-1	-1	-1	71
17	18	-1	-1	-1	-1	-1	44
18	19	20	-1	-1	-1	-1	58
19	-1	-1	-1	-1	-1	-1	75
20	21	-1	-1	-1	-1	-1	99
21	22	23	-1	-1	-1	-1	17
22	-1	-1	-1	-1	-1	-1	1
23	24	25	26	-1	-1	-1	50
24	-1	-1	-1	-1	-1	-1	8
25	-1	-1	-1	-1	-1	-1	90
26	27	28	29	-1	-1	-1	78
27	-1	-1	-1	-1	-1	-1	74
28	-1	-1	-1	-1	-1	-1	52
29	30	-1	-1	-1	-1	-1	7
30	-1	-1	-1	-1	-1	-1	54

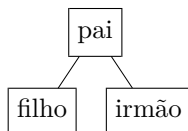
Obs: 1. A raiz está na posição 1 da tabela. 2. A árvore não tem NE-NHUMA ordenação prévia.

Ao desenhar a árvore, se obtém...(copie do quadro)

Exercício 1 Desenhe a árvore acima

Exercício 2 Responda quantos nodos estão no nível 4 ?

Exercício 3 Usando a equivalência

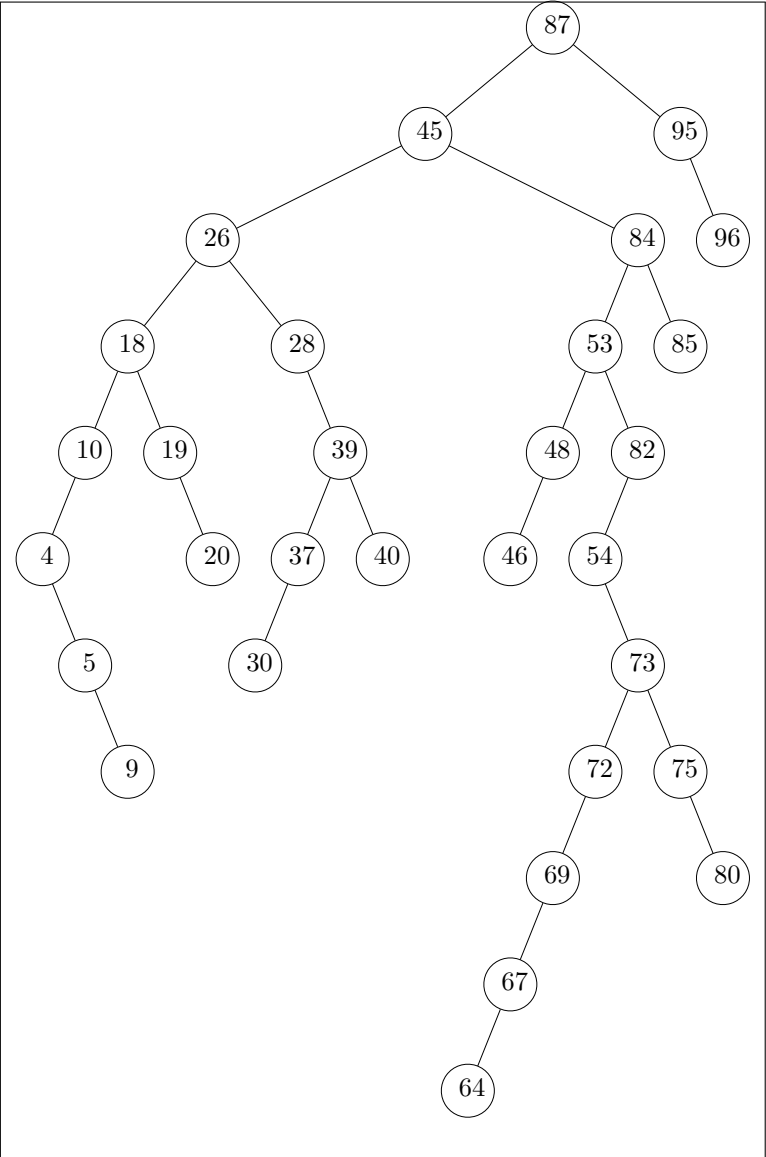


Transforme a árvore acima em uma árvore binária e desenhe-a. Depois, responda: quais os valores que estão no nível 14?

Exercício 4 Represente a árvore acima na forma de uma matriz R de 3 colunas (filho-esquerdo, filho-direito e conteúdo). Note que a coluna 3 desta matriz é exatamente igual a coluna 7 da primeira matriz. Depois, responda, quais os valores de: R[23;2] ? e R[27;1] ?

Balanceamento em árvores ABP

Considere a seguinte árvore, que está desbalanceada:



Para você fazer

- Verifique se a árvore acima é uma ABP.
- Se for, desenhe uma ABP equivalente, estritamente balanceada.
- Faça uma visitação em largura à árvore que você desenhou, obtendo o vetor VV.
- Informe as posições de VV que são pedidas:

1	9	15	17	26

1



- 1 - /

ABPs: busca, criação e altura

Altura Como visto na sala a altura de uma árvore binária de pesquisa é O MAIOR CAMINHO ENTRE A RAIZ E QUALQUER FOLHA. Se a árvore for completamente balanceada, então todos os caminhos entre raiz e folha têm o mesmo comprimento, e portanto, pode-se escolher qualquer um. Suponha uma árvore binária de pesquisa definida por uma matriz de 3 colunas (&filho esquerdo & filho direito e conteúdo) também como visto em sala de aula. Qual seria o algoritmo para determinar a altura da árvore ? Suponha que a variável ARVORE é global e já está corretamente preenchida.

```
1: funcao ACHA-ALTURA (inteiro INIC, inteiro RAIZ)
2: se INIC > A
3:   A ← INIC + 1
4: fim{se}
5: se ARVORE[RAIZ][1] ≠ -1
6:   ACHA-ALTURA (INIC+1, ARVORE[RAIZ][1])
7: fim{se}
8: se ARVORE[RAIZ][2] ≠ -1
9:   ACHA-ALTURA (INIC+1, ARVORE[RAIZ][2])
10: fim{se}
11: fimfunção
```

Note que a função é recursiva e utiliza a variável global A, que deve ser inicializada com ZERO antes da função ser chamada. Supondo uma ARVORE com a raiz na linha 1, a chamada deveria ser

```
A ← 0
ACHA-ALTURA (0,1)
Ao final, para saber a altura desejada, há que se fazer
escreva (A)
```

quantidade de folhas Seja agora o algoritmo que permite contar a quantidade de folhas em uma árvore.

```
1: funcao ACHA-FOLHA (inteiro RAIZ)
2: se ((ARVORE[RAIZ][1] = -1) ^ (ARVORE[RAIZ][2] = -1 ))
3:   QTF ← QTF + 1
4: fim{se}
5: se ARVORE[RAIZ][1] ≠ -1
6:   ACHA-FOLHA (ARVORE[RAIZ][1])
7: fim{se}
8: se ARVORE[RAIZ][2] ≠ -1
9:   ACHA-FOLHA (ARVORE[RAIZ][2])
10: fim{se}
11: fimfunção
```

Aqui também há a variável global QTF que também precisa ser inicializada com zero fora da função. A chamada seria:

```
QTF ← 0
ACHA-FOLHA (1)
Ao final, para saber a quantidade de folhas desejada, há que se fazer
escreva (QTF)
```

Altura de ABP

1. Para responder a este exercício, voce deve construir uma árvore de pesquisa binária em ordem, com os elementos fornecidos pelo exercício, NA ORDEM EM QUE ELES APARECEM.
Depois de construir a ABP, responda qual a altura da árvore criada

- a) 15 67 24 69 11 58 51 92 49 73 52 20 8 87 85 A=6
- b) 8 11 15 20 24 49 51 52 58 67 69 73 85 87 92 A=14
- c) 52 20 73 11 49 67 87 8 15 24 51 58 69 85 92 A=3
- d) 20 11 15 8 24 49 51 52 58 67 69 73 85 87 92 A=11

Para você fazer

Qual a altura de uma árvore binária de pesquisa (em ordem) formada pelos números:

- a) 7 38 5 81 62 54 28 95 87 61 35 31 64 19 66 _____
- b) 5 7 19 28 31 35 38 54 61 62 64 66 81 87 95 _____

Quantidade de folhas

2. Para este exercício, voce deve criar as árvores com os elementos fornecidos, na ordem, e contar – ao final – quantas folhas tem a árvore formada:

- a) 11 77 48 71 30 16 22 7 58 80 44 45 25 15 96 F=6
- b) 7 11 15 16 22 25 30 44 45 48 58 71 77 80 96 F=1

Para você fazer

- a) 61 46 54 44 78 72 40 56 29 7 57 12 32 43 64 _____

ABP de altura mínima

3. Para o exercício a seguir, você deve construir uma árvore binária de pesquisa em ordem, de tal maneira que ela tenha a ALTURA MINIMA POSSIVEL.

Para conseguir fazer isso, você deve modificar a ordem em que os dados a seguir (e que deverão ser incluídos na árvore) serão usados pelo algoritmo de criação da árvore, A primeira pergunta é: Como os dados devem ter sua ordem modificada para que na criação da árvore esta seja de altura mínima ?

Após modificar o vetor segundo essa ordem, responda quem são o terceiro e o décimo terceiro elemento:

Por exemplo, seja o seguinte conjunto de elementos:

- 7 68 95 18 73 30 99 66 51 69 12 33 43 85 76

Para construir com eles uma árvore de altura mínima, ele deverá ser ordenado especialmente. Depois que esta ordem for construída... o terceiro é: 76 e o décimo-terceiro é: 73

Para você fazer

Construa uma árvore ABP em ordem de altura mínima com

- 43 66 97 10 74 25 16 68 29 46 83 38 94 61 2

Para conseguir isso, os dados devem ser modificados. Após obter a ordenação, responda quem é o 3. _____ e o 13. _____

Pesquisa

4. No exercício a seguir, suponha que temos números entre 1 e 1000 em uma árvore binária de pesquisa em-ordem. Queremos procurar a chave 974. Verifique os números das seqüências que NÃO podem ser seqüência válida de nós examinados. SOME OS NUMEROS ENCONTRADOS e responda ao exercício.

Seja o exemplo:

- 1. 447 614 831 920 951 990 984 963 991 965
- 2. 877 986 909 915 939 944 952 977 957 973
- 4. 997 551 830 889 959 992 999 990 972 989
- 8. 665 754 907 946 983 957 961 980 952 975
- 16. 924 988 931 984 944 964 969 970 982 979

Examinando a linha 1, ve-se que que há um erro na seqüência de chaves pesquisadas, no item 991, logo a linha 1 está errada.

- A linha 2, está certa.
- A linha 4, tem erro no item 999, logo a 4 está errada.
- A linha 8 tem erro no item 952, e a linha 16 esta certa.
- Assim, o erro deste exemplo é: 1+4+8=13.

Para você fazer

O objetivo é localizar a chave: 106

- 1. 368 57 124 111 58 110 74 84 64 105
- 2. 383 281 128 27 50 110 52 96 102 108
- 4. 205 193 24 159 84 97 116 100 81 106
- 8. 178 137 47 51 120 82 117 111 65 104
- 16. 10 169 30 32 156 149 56 117 104 105

Os erros aparecem nas linhas _____, que somadas dão _____.

1a.	1b.	2a.	3 (3°)	3 (13°)	4.



- 1 - /

Inclusões e Exclusões em Árvores Binárias de Pesquisa

Suponha 3 ABPs compartilhando uma única tabela de 30 x 3 colunas. O formato desta tabela é aquele tradicional (&fe, &fd, conteúdo). A raiz da primeira ABP está na linha 1, a segunda em 2 e a terceira em 3. As árvores devem ser construídas e a seguir executadas as inclusões e exclusões. Para tanto, utilize os seguintes algoritmos:

```
1: inteiro função incluiabp (inteiro RAIZ, inteiro VAL)
2: inteiro onde,pai,esqdir
3: ULTIMO ← ULTIMO+1
4: TABARV[ULTIMO][ ] ← (-1, -1, VAL)
5: onde ← RAIZ
6: enquanto onde ≠ -1 faça
7:   pai ← onde
8:   se VAL > TABARV[onde][3] então
9:     esqdir ← 1
10:    onde ← TABARV[onde][2]
11:   senão
12:     esqdir ← 0
13:    onde ← TABARV[onde][1]
14:   fim se
15: fim enquanto
16: TABARV[pai][1+esqdir] ← ULTIMO
17: devolve ULTIMO
18: fimfunção
```

Agora a função de exclusão em árvores binárias de pesquisa.

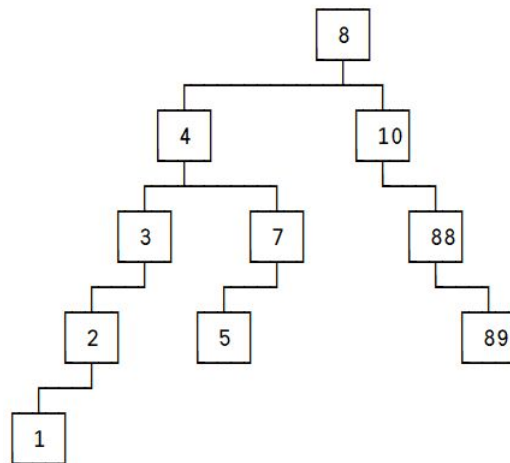
```
1: inteiro função excluiabp (inteiro RAIZ, inteiro VAL)
2: inteiro esqdir,aaux,pai,elemx,save,tempo
3: esqdir ← 0
4: aaux ← pai ← RAIZ
5: enquanto VAL ≠ TABARV[aaux][3] faça
6:   pai ← aaux
7:   se VAL > TABARV[aaux][3] então
8:     esqdir ← 1
9:     aaux ← TABARV[aaux][2]
10:   senão
11:     esqdir ← 0
12:     aaux ← TABARV[aaux][1]
13:   fim se
14: fim enquanto
15: elemx ← aaux
16: se TABARV[elemx][1] = -1 ∧ TABARV[elemx][2] = -1 então
17:   TABARV[pai][1+esqdir] ← -1
18:   devolve elemx
19: senão
20:   se TABARV[elemx][1] = -1 então
21:     TABARV[pai][1+esqdir] ← TABARV[elemx][2]
22:     devolve elemx
23:   senão
24:     se TABARV[elemx][2] = -1 então
25:       TABARV[pai][1+esqdir] ← TABARV[elemx][1]
26:       devolve elemx
27:     senão
28:       elemx ← TABARV[elemx][2]
29:       enquanto elemx ≠ -1 faça
30:         save ← elemx
31:         elemx ← TABARV[elemx][1]
32:       fim enquanto
33:       tempo ← incluiabp (raiz, TABARV[save][3])
34:       TABARV[aaux][3] ← TABARV[tempo][3]
35:       devolve aaux
36:   fim se
37:   fim se
38: fim se
39: fimfunção
```

Acompanhe o exemplo

Considere que (ao contrário do exercício, há apenas 1 árvore na tabela. Seja a ABP

```
1: 2 3 8
2: 4 5 4
3: -1 9 10
4: 7 -1 3
5: 6 -1 7
6: -1 -1 5
7: 8 -1 2
8: -1 -1 1
9: -1 10 88
10: -1 -1 89
```

cujo visual é



incluindo 12,14,50,33,19,18
 excluindo 7, 89, 2, 10 e 4
 e fazendo o caminhamento horizontal, tem-se:
 8,5,88,3,12,1,14,50,33,19,18
 (OBS: a ordem das inclusões e exclusões não importa)

Para você fazer

Dadas as árvores (formato: &fe, &fd, cont)

1	4	-1	50
2	5	8	31
3	18	6	17
4	10	7	28
5	14	20	9
6	15	9	28
7	-1	16	34
8	11	17	36
9	12	-1	49
10	13	-1	27
11	-1	-1	35
12	-1	21	33
13	25	19	8
14	-1	-1	3
15	27	-1	26

← raiz arv 1
 ← raiz arv 2
 ← raiz arv 3

16	-1	-1	43
17	23	-1	66
18	-1	-1	10
19	22	-1	22
20	-1	-1	11
21	-1	24	34
22	-1	-1	16
23	-1	26	46
24	-1	29	37
25	-1	-1	7
26	28	-1	65
27	-1	-1	24
28	-1	30	54
29	-1	-1	39
30	-1	-1	57

Faça as seguintes inclusões / exclusões. Note que cada item informa:

árvore	operação	valor

1 inclui 15	1 exclui 7
3 exclui 37	1 inclui 5
1 inclui 40	2 inclui 51
1 exclui 28	1 exclui 22
3 inclui 27	2 exclui 65
2 inclui 61	3 inclui 12
1 exclui 16	2 inclui 48
1 inclui 21	2 exclui 11
3 exclui 39	2 exclui 3
3 exclui 10	3 inclui 18
3 inclui 38	3 inclui 46
2 inclui 2	1 inclui 23
2 exclui 54	3 inclui 30
2 inclui 44	3 exclui 28

Depois de construir as árvores, incluir e excluir os elementos pedidos, faça os caminhamentos horizontais C1, C2 e C3 (em largura) e responda:

C3[5]	C3[3]	C3[1]	C3[6]	C1[2]

C2[5]	C2[6]	C2[2]	C3[11]	C1[9]

1



- 1 - /

ABP

A estrutura de árvore binária de pesquisa (conhecida como ABP) é muito importante na ciência da computação. Baseado nela existe um dos mais eficientes e simples algoritmos de ordenação: cria-se a abp com os elementos a ordenar e depois percorre-se a árvore em ordem: estão ordenados todos os elementos. Tem uma eficiência de $O(n \times \log_2 n)$.

Lembrando, uma ABP é binária (no máximo 2 filhos) e na sua construção usa-se a regra, que é válida para todos os seus nodos: tomando o nodo x como raiz, todos os nodos ligados ao filho esquerdo deverão ter valor MENOR que x , e todos os nodos ligados ao filho direito deverão ter valor MAIOR que x .

Esta folha vai tratar os algoritmos de criação, inclusão e exclusão de valores em *abps*. Em nenhum momento haverá duplicidade de chaves em qualquer árvore, isto não precisa ser testado.

Finalmente, depois pede-se um caminhamento sobre a árvore. A seguir os principais caminhamentos. Lembrar que os 3 primeiros devem ser aplicados recursivamente a toda a árvore.

pré-ordem raiz, filho-esquerdo, filho-direito.

em-ordem filho-esquerdo, raiz, filho-direito.

pós-ordem filho-esquerdo, filho-direito, raiz.

largura nível 0 (raiz), depois todos os do nível 1, 2, ...

Inclusões e Exclusões em ABPs

Suponha 3 ABPs independentes, com cerca de 20 números cada uma. Haverá uma sequência de 20 operações de exclusão em uma das árvores seguida da inclusão do mesmo elemento em outra das duas árvores.

Ao final, continuará havendo as 3 árvores, com 20 elementos tendo trocado de árvore. Você vai receber a sequência de criação das 3 árvores e depois deve efetuar as 20 operações. Depois que as 3 árvores finais estiverem prontas, você deve produzir a visitação horizontal das 3 árvores. Finalmente, deve responder a perguntas sobre essa visitação.

Lembre-se que a ordem dos elementos na entrada precisa ser rigorosamente obedecida e não pode ser alterada, sob pena do formato da árvore (e consequentemente a visitação) ser alterada.

Ao incluir, recorde-se que o local de inclusão é único e está sempre perfeitamente determinado, desde que se obedeca sempre a regra geral que é: "dado um elemento na raiz, (todos) os filhos esquerdos terão que ser MENORES do que a raiz e (todos) os filhos direitos terão que ser MAIORES do que a raiz.

Ao excluir, considere os 3 casos de exclusão, a depender do número de filhos do nodo em questão:

sem filhos A exclusão é trivial, basta eliminar o nodo e corrigir o apontamento do pai para ele.

- 1 filho O filho do nodo a excluir substitui o nodo a excluir.
- 2 filhos Este é o caso mais complicado. Há algumas tarefas

- Localizar na árvore o nodo sucessor do nodo a excluir
- Excluir o sucessor
- Trocar o nodo a excluir pelo seu sucessor

Algumas observações: 1. O sucessor nunca terá os 2 filhos, daí não existe a possibilidade do algoritmo entrar em loop. 2. Fazendo a troca do sucessor pelo nodo a excluir, por mais distante que o sucessor esteja do sucedido, a árvore sempre continua sendo uma árvore ABP.

Exemplo Sejam as seguintes 3 árvores:

35 78 16 55 33 8 29 91 56 80 15 86 87 44 43
11 47 48 67 89 39 45 28 68 41 69 3 85 46
31 7 90 1 83 26 93 32 73 66 9 60 12 2 70 14

A seguir, devem-se executar as seguintes operações: (a ordem das operações não é relevante).

retire 09 de 3 e insira-o em 1
retire 29 de 1 e insira-o em 2
retire 60 de 3 e insira-o em 2
retire 32 de 3 e insira-o em 1
retire 73 de 3 e insira-o em 2
retire 46 de 2 e insira-o em 1
retire 67 de 2 e insira-o em 1
retire 15 de 1 e insira-o em 3
retire 02 de 3 e insira-o em 1
retire 45 de 2 e insira-o em 3
retire 93 de 3 e insira-o em 1
retire 48 de 2 e insira-o em 3

Após a execução destas operações, o caminhamento em largura das 3 árvores resultantes ficou assim:

35 16 78 8 33 55 91 2 9 32 44 56 80 93 43 46 67 86 87
11 3 47 39 68 28 41 60 89 29 69 85 73
31 7 90 1 26 83 12 66 14 45 70 15 48



Para você fazer

Resolva a seguinte instância
Crie as 3 árvores com os elementos a seguir (obs: antes de começar, conte todos os elementos: eles devem totalizar 60 números nas 3 árvores).

Arvore 1:
74 50 4 36 18 53 29 80 58 77 13 34 9 89 32 93 90 33 10 56 71 14

Arvore 2:
79 87 45 15 40 17 67 43 22 70 19 57 62 54 84 69 83 61 35 31 94

Arvore 3:
51 41 20 98 85 75 91 23 11 66 76 42 64 65 55 3 8

Depois aplique as seguintes operações, em número de 20:

retire 93 de 1 e insira-o em 2
retire 66 de 3 e insira-o em 2
retire 89 de 1 e insira-o em 2
retire 34 de 1 e insira-o em 3
retire 67 de 2 e insira-o em 3
retire 87 de 2 e insira-o em 3
retire 20 de 3 e insira-o em 2
retire 62 de 2 e insira-o em 1
retire 43 de 2 e insira-o em 3
retire 14 de 1 e insira-o em 2
retire 94 de 2 e insira-o em 1
retire 54 de 2 e insira-o em 3
retire 42 de 3 e insira-o em 2
retire 36 de 1 e insira-o em 2
retire 41 de 3 e insira-o em 2
retire 11 de 3 e insira-o em 2
retire 35 de 2 e insira-o em 1
retire 57 de 2 e insira-o em 1
retire 56 de 1 e insira-o em 3
retire 32 de 1 e insira-o em 2

E agora construa o VETOR DE CAMINHAMENTO EM LARGURA das 3 árvores e depois responda:

Arv 1	Arv 2	Arv 2	Arv 3	Arv 3
V[11]	V[15]	V[22]	V[17]	V[16]

Para esta folha, a origem dos índices é 1. Logo, a contagem é 1, 2, 3... n e não 0, 1, 2... $n - 1$.

1



- 1 - /

chave	33	38	42	45	48	49	50	57	60	64
freq	11	15	8	10	7	14	6	12	5	1
continuação...	73	83	84	85	95					
	13	3	4	9	2					

```

graph TD
    38((38)) --> 33((33))
    38 --> 49((49))
    33 --> 45((45))
    33 --> 48((48))
    45 --> 42((42))
    45 --> 48((48))
    49 --> 57((57))
    49 --> 73((73))
    57 --> 50((50))
    57 --> 60((60))
    60 --> 64((64))
    73 --> 85((85))
    73 --> 83((83))
    85 --> 84((84))
    85 --> 95((95))
  
```

Construa a ABP e a ABP' para o conjunto acima.

```

graph TD
    A["38  
57"] --> B["49  
45"]
    A --> C["73  
83"]
    B --> D["33  
42"]
    B --> E["57  
48"]
    C --> F["60  
64"]
    C --> G["85  
84"]
    D --> H["42"]
    D --> I["45"]
    E --> J["48"]
    E --> K["50"]
    F --> L["64"]
    F --> M["83"]
    G --> N["84"]
    G --> O["95"]
  
```

			1



Treinamento em ABP

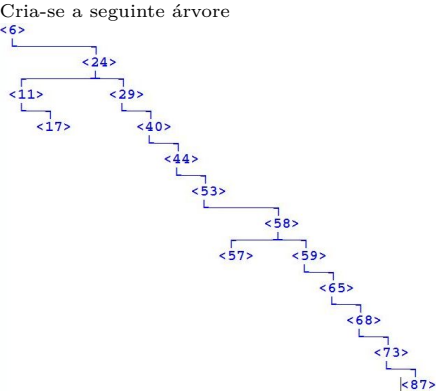
Esta folha visa ajudar os alunos a treinar, memorizar e automatizar o processo de criação e manipulação de árvores binárias de pesquisa. Lembrando a regra básica (e única) para a criação destas entidades: o primeiro nodo é a raiz e depois disso, a cada nodo seguinte usa-se a regra: **valor menor que a raiz vai para a esquerda e se o valor é maior vai para a direita.**

Exemplo 1 Usando esta regra para o seguinte conjunto de dados

6 24 29 40 44 53 58 59 65 68 73 87 11 57 17

computacionalmente, eis como ficou a árvore

0:	-1	2	6
1:	13	3	24
2:	-1	4	29
3:	-1	5	40
4:	-1	6	44
5:	-1	7	53
6:	14	8	58
7:	-1	9	59
8:	-1	10	65
9:	-1	11	68
10:	-1	12	73
11:	-1	-1	87
12:	-1	15	11
13:	-1	-1	57
14:	-1	-1	17



Para esta árvore os quantitativos de

altura 11

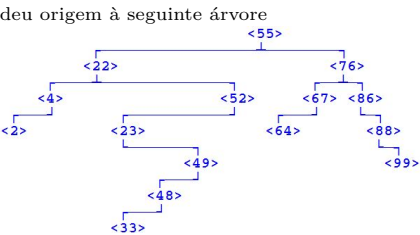
nodos folha 3 nodos (17, 547 e 87)

nodos com 1 filho 10 nodos (6, 11, 29, 40, 44, 53, 59, 65, 68, 73)

nodos com 2 filhos 2 nodos (24 e 58)

Exemplo 2 Mais um exemplo, para os dados

55 76 86 88 22 67 52 4 23 99 49 2 48 64 33



com os seguintes quantitativos de altura e nodos com 0 filhos (folhas), 1 filho e 2 filhos: **6, 4, 8 e 3.**

Programas em Python Embora este exercício possa e deva ser feito à mão, subrepticamente surge o convite para um interessante exercício de programação. Vamos ao python

```
# criação de ABP em Python
import numpy as np
global ARV
ABP=np.full((15,3),-1,int)
def crarvore(x):
    ABP[0]=[-1,-1,x[0]]
    i=1
    ult=0
    while i<len(x):
        ult=ult+1
        ABP[ult]=[-1,-1,x[i]]
        onde=0
        while onde!=-1:
            pai=onde
            if x[i]>ABP[onde,2]:
                esqdir=1
                onde=ABP[onde,1]
            else:
                esqdir=0
                onde=ABP[onde,0]
            ABP[pai,esqdir]=ult
            i=i+1
        print(ABP)
# abp=crarvore([1,2,3...])

def altura(atual):
    if ABP[atual][0]==-1 and ABP[atual][1]==-1:
        return 0
    else:
        if altura(ABP[atual][0]) >
            altura(ABP[atual][1]):
            return 1 + altura(ABP[atual][0])
        else:
            return 1 + altura(ABP[atual][1])
# print(altura(0))

def folhas(atual):
    if atual == -1:
        return 0
    if ABP[atual][0]==-1 and ABP[atual][1]==-1:
        return 1
    return folhas(ABP[atual][0]) +
        folhas(ABP[atual][1])
# print(folhas(0))

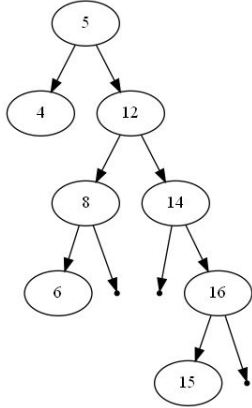
def contarNos(atual):
    if atual == -1:
        return 0
    else:
        return 1 + contarNos(ABP[atual][0]) +
            contarNos(ABP[atual][1])
# print(contarNos(0))

def des(atual):
    global ABP,nu,fi
    if ABP[atual][0]==-1 and ABP[atual][1]==-1:
        return
    if ABP[atual][0] == -1:
        fi.write('{0} -> null{1}; null{1}\
[shape=point]\n'.format(ABP[atual][2],nu))
        nu=nu+1
        fi.write('{0} -> {1} \n'.format(ABP[atual]\
[2],ABP[ABP[atual][1]][2]))
        des(ABP[atual][1])
        return
    if ABP[atual][1] == -1:
        fi.write('{0} -> {1} \n'.format(ABP[atual]\
[2],ABP[ABP[atual][0]][2]))
        des(ABP[atual][0])
        fi.write('{0} -> null{1}; null{1}\
[shape=point]\n'.format(ABP[atual][2],nu))
        nu=nu+1
        return
    fi.write('{0} -> {1} \n'.format(ABP[atual]\
[2],ABP[ABP[atual][0]][2]))
    des(ABP[atual][0])
    fi.write('{0} -> {1} \n'.format(ABP[atual]\
[2],ABP[ABP[atual][1]][2]))
    des(ABP[atual][1])
    return

def desenha(atual,arquivo):
    global ABP,nu,fi
    fi = open(arquivo,"w")
    nu=0
    fi.write("digraph BST { \n")
    des(0)
    fi.write('}\n')
    fi.close()
    return
```

#desenha(0,'c:/p/n/437/teste.dox')

Para ver o resultado do desenho, instale o graphviz (<https://graphviz.org>) garantindo que o caminho até ele seja incluído na variável PATH, e então entre no diretório onde salvou (neste exemplo 'c:/p/n/437') e execute o comando >dot teste.dox -T png -o lixo.png e o arquivo será criado neste mesmo diretório, com o nome lixo.png podendo ser visualizado, copiado, colado... por qualquer visualizador de imagens. Eis um exemplo do resultado



👉 Para você fazer

Você deve construir 10 árvores binárias de pesquisa com os seguintes 15 valores para cada árvore

- 51 69 6 80 16 40 34 53 97 27 79 62 32 39 82
- 71 1 49 9 88 54 79 60 21 3 55 63 11 89 6
- 12 30 33 40 44 53 78 90 92 93 82 17 81 65 8
- 5 30 39 58 66 71 75 91 93 94 98 46 24 32 18
- 13 15 21 38 72 77 94 12 6 95 35 42 66 36 5
- 3 27 29 39 47 70 88 89 90 97 85 87 94 53 77
- 7 8 20 35 73 92 97 58 83 31 99 11 96 40 17
- 78 60 93 54 5 95 11 63 59 19 62 58 17 87 77
- 20 32 43 44 50 51 66 72 86 90 21 94 85 38 53
- 49 70 77 87 91 93 94 32 2 59 41 5 48 98 18

Responda aqui

árv	altura	folhas	1 filho	2 filhos
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
Σ				

Aquecimento em árvores B

Suponha uma b-árvore M com $t=4$. Seus nodos têm entre 3 e 7 chaves. M é uma matriz de 17 colunas (indicativo de folha/qtde de chaves válidas/ 7 chaves/ 8filhos). M já contém 99 números.

Eis o significado das colunas:

coluna 0 um asterisco nesta posição indica que este é o nodo raiz da árvore.

coluna 1 indicativo de folha (1=sim; 0=não)

coluna 2 quantidade de chaves válidas neste nodo (nem sempre o valor inválido é zero)

cols 3 a 9 chaves, sempre em ordem crescente

cols 10 a 17 endereços dos filhos associados a cada chave. Só estão preenchidas para nodos não folha.

Veja um exemplo

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	5	3	6	7	8	10	0	0								
2	0	4	11	20	32	44	0	0	0	1	20	15	7	12			
3	1	4	128	131	132	134	0	0	0								
4	1	5	149	150	151	153	160	0	0								
5	1	6	65	73	76	79	80	83	0								
6	1	5	182	183	188	189	191	0	0								
7	1	3	34	35	37	0	0	0	0								
8	1	4	106	109	111	114	0	0	0								
9	1	6	136	137	138	140	143	145	0								
*10	0	2	64	127	0	0	0	0	0	2	17	11					
11	0	6	135	146	162	171	181	192	0	3	9	4	14	21	6	18	
12	1	7	46	49	50	55	56	57	63								
13	1	4	88	89	91	94	0	0	0								
14	1	5	164	166	167	168	169	0	0								
15	1	6	21	22	23	28	29	30	0								
16	1	7	117	118	120	122	123	125	126								
17	0	4	84	95	105	116	0	0	0	5	13	19	8	16			
18	1	5	193	195	197	199	200	0	0								
19	1	5	96	98	100	102	103	0	0								
20	1	4	14	15	17	19	0	0	0								
21	1	3	174	175	177	0	0	0	0								

Antes de continuar no exercício, você deve tentar entender o funcionamento da árvore-b.

- Desenhe a árvore
- identifique a necessidade da coluna 1. Idem para a coluna 2
- Qual a altura da árvore ?
- Quando e como a altura da árvore cresce ?

Agora suponha que o elemento 187 é incluído. Qual a linha;coluna da matriz acima que é modificada, ou seja onde o elemento recém incluído entra ? Localize-o e veja:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	5	3	6	7	8	10	0	0								
2	0	4	11	20	32	44	0	0	0	1	20	15	7	12			
3	1	4	128	131	132	134	0	0	0								
4	1	5	149	150	151	153	160	0	0								
5	1	6	65	73	76	79	80	83	0								
6	1	6	182	183	187	188	189	191	0								
7	1	3	34	35	37	0	0	0	0								
8	1	4	106	109	111	114	0	0	0								
9	1	6	136	137	138	140	143	145	0								
*10	0	2	64	127	0	0	0	0	0	2	17	11					
11	0	6	135	146	162	171	181	192	0	3	9	4	14	21	6	18	
12	1	7	46	49	50	55	56	57	63								
13	1	4	88	89	91	94	0	0	0								
14	1	5	164	166	167	168	169	0	0								
15	1	6	21	22	23	28	29	30	0								
16	1	7	117	118	120	122	123	125	126								
17	0	4	84	95	105	116	0	0	0	5	13	19	8	16			
18	1	5	193	195	197	199	200	0	0								
19	1	5	96	98	100	102	103	0	0								
20	1	4	14	15	17	19	0	0	0								
21	1	3	174	175	177	0	0	0	0								

Como voce pode ver ele está na linha ____ e coluna ____.

Mais um exemplo. Suponha a inclusão do número 26. Em que linha;coluna ele entra ?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	5	3	6	7	8	10	0	0								
... linhas iguais nao impressas ...																	
9	1	6	136	137	138	140	143	145	0								

*10	0	2	64	127	0	0	0	0	0	2	17	11					
11	0	6	135	146	162	171	181	192	0	3	9	4	14	21	6	18	
12	1	7	46	49	50	55	56	57	63								
13	1	4	88	89	91	94	0	0	0								
14	1	5	164	166	167	168	169	0	0								
15	1	7	21	22	23	26	28	29	30								
16	1	7	117	118	120	122	123	125	126								
17	0	4	84	95	105	116	0	0	0	5	13	19	8	16			
... linhas iguais nao impressas ...																	
21	1	3	174	175	177	0	0	0	0								

Como você pode ver ele entra na linha ____, coluna ____.

Algoritmo de Inclusão em árvore B

```
1: função INCLUA-BTREE (K:inteiro)
2: X ← raiz da árvore B
3: enquanto nodo X não é folha faça
4:   X ← filho correto onde K deve ser incluído
5: fim enquanto
6: se nodo X está cheio então
7:   faça split do nodo X
8:   INCLUA-BTREE (K)
9: abandone
10: senão
11:   inclua K no nodo X mantendo as chaves ordenadas
12: fim se
```



Para você fazer

Desenhe a árvore B a seguir:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	5	31	32	34	35	36	0	0								
2	0	3	37	43	60	0	0	0	0	1	9	15	16				
3	1	3	80	81	82	0	0	0	0								
4	1	4	157	158	159	160	0	0	0								
5	1	7	111	112	113	114	115	117	120								
6	1	4	75	76	77	78	0	0	0								
7	1	4	174	177	178	181	0	0	0								
8	1	7	123	125	128	132	133	135	136								
9	1	4	38	39	40	41	0	0	0								
*10	0	3	69	108	156	0	0	0	0	2	19	11	21				
11	0	3	122	139	150	0	0	0	0	5	8	12	20				
12	1	5	140	143	145	146	147	0	0								
13	1	6	185	186	188	189	191	192	0								
14	1	4	85	86	87	88	0	0	0								
15	1	6	46	53	54	55	58	59	0								
16	1	5	62	63	64	65	66	0	0								
17	1	5	195	196	197	198	200	0	0								
18	1	4	91	92	94	95	0	0	0								
19	0	4	79	83	90	97	0	0	0	6	3	14	18	23			
20	1	3	152	153	155	0	0	0	0								
21	0	4	164	170	184	193	0	0	0	4	22	7	13	17			
22	1	4	165	166	168	169	0	0	0								
23	1	3	99	104	107	0	0	0	0								

Responda agora:

Qual a altura da árvore B formada pela matriz acima ?	Se o 148 for incluído, qual a sua linha e coluna na matriz acima ?	Se o 33 for incluído, qual a sua linha e coluna na matriz acima ?

1



- 1 - /

Exercício de Árvores B com grau t=3

Em 1972, Bayer e MacCreight introduziram uma estrutura de dados bastante adequada para resolver o problema de construir índices em memórias secundárias. Denominaram-na B-árvore, mas absteriveram-se de explicar este nome. A B-árvore é uma árvore k-ária, isto é, cada nodo pode ter k filhos, sendo que em geral este valor de k é escolhido de tal maneira a otimizar a blocagem física do arquivo de índices.

O nodo original é chamado "raiz" e é o único que não tem pai, mas pode ter de 0 até $2n - 1$ chaves e $2n$ filhos. Cada um dos filhos, por sua vez, pode ter de $n - 1$ a $2n - 1$ chaves e de 0 a $2n$ filhos (que serão netos do nodo raiz) e assim sucessivamente. Este valor de n, é denominado grau da árvore, e as árvores B têm este valor na casa das dezenas ou centenas. Cria-se assim uma estrutura piramidal, com vaga semelhança com as árvores: daí o nome.

Seja o problema de construir uma matriz M representando uma árvore B (BTREE) sujeita às seguintes condições:

1. Os nodos comportam 5 chaves e 6 filhos cada

2. Os nodos são linhas de uma matriz alocada seqüencialmente

3. As colunas da matriz são:

coluna 1 1 se esta linha é folha e 0 senão

coluna 2 contador de chaves válidas neste nodo

cols 3-4-5-6-7 chaves (ou zero)

cols 8-9-10-11-12-13 se o nodo não é folha, o cursor dos filhos

4. A árvore B mantém os dados em ordem crescente

5. A raiz é representada por um "*"

Nesta estrutura, use os seguintes algoritmos

```
#include<stdio.h>
int barv[100][14];
int dr;
int dprox;
int dt;
int insbarv (int chave) {
    int r,s;
    r = dr;
    if (barv[dr][2]>=((2*dt)-1)) {
        s = dprox;
        dprox = dprox+1;
        dr=s;
        barv[s][1]=0;
        barv[s][2]=0;
        barv[s][2+(2*dt)]=r;
        quebrabarv(s,1,r);
        insncbarv(s,chave);
    }
    else {
        insncbarv(r,chave);
    }
}

int quebrabarv(int pai, int i, int filho){
    int z,j;
    z = dprox;
    dprox = dprox + 1;
    barv[z][1]=barv[filho][1]; //folha=1 ou nao=0
    barv[z][2]=dt-1; // quantidade de chaves no comeco (t-1)
    for (j =1; j<dt;j++){ //repassando as chaves para o filho
        barv[z][2+j] = barv[filho][2+dt+j];
    }
    if (barv[filho][1]==0){ //se tem pointers, tem que move-los
        for (j=1; j<dt;j++){
            barv[z][(((2*dt)+2)-1)+j]=barv[filho][(((2*dt)+2)-1)+dt+j];
        }
    }
    barv[filho][2]=dt-1; // quantidade de chaves do filho
    for (j=barv[pai][2]+1;j!=1;j--){
        barv[pai][2+(2*dt)+j]=barv[pai][(((2*dt)+2)-1)+j];
    }
    barv[pai][2+(2*dt)+1]=z;
    for (j=barv[pai][2];j>i-1;j--){
        barv[pai][3+j]=barv[pai][2+j];
    }
    barv[pai][2+i]=barv[filho][2+dt];
    barv[pai][2]=barv[pai][2]+1;
}

int insncbarv(int nodo, int chave){ // no apl=funcao nc
```

```
int i,xvelho;
i = barv[nodo][2];
if (barv[nodo][1] !=0){
    while ((i >= 1) && (chave < barv[nodo][2+i])) {
        barv[nodo][i+3]=barv[nodo][i+2];
        i--;
    }
    barv[nodo][i+3]=chave;
    barv[nodo][2] = barv[nodo][2]+1;
}
else {
    while ((i >= 1) && (chave < barv[nodo][2+i])) {
        i--;
    }
    i++;
    xvelho=nodo;
    nodo=barv[nodo][(((2*dt)+2)-1)+1];
    if (barv[nodo][2]==((2*dt)-1)) {
        quebrabarv (xvelho,1,nodo);
        if (chave > barv[xvelho][2+i]) {
            i++;
        }
    }
    insncbarv (barv[xvelho][(((2*dt)+2)-1)+i], chave);
}

int printabarv() {
    int k;
    for (k=1;k<dprox;k++){
        printf ("%2i %2i %3i %3i %3i %3i %3i %3i %3i\n",
            barv[k][1],barv[k][2],barv[k][3],barv[k][4],barv[k][5],
            barv[k][6],barv[k][7],barv[k][8],barv[k][9],barv[k][10],
            barv[k][11],barv[k][12],barv[k][13],barv[k][14]);
        printf("a raiz e = %i\n",dr);
    }
}

main() {
    int i,j,k;
    for (i=0;i<100;i++){
        for(j=0;j<14;j++){
            if (j==1) {
                barv[i][j]=1;
            }
            else {
                barv[i][j]=0;
            }
        }
    }
    dt=3; // alterar este numero se quiser arvores diferentes
    dprox=2;
    dr=1;
    insbarv(20);
    insbarv(42);
    ...
    insbarv(4);
    printabarv();
}
```

Seja um exemplo

Criar uma matriz vazia e inserir os números:

16	65	89	79	42	35	52	37	2	71	46	29	14	22	40	80	92
44	57	49	32	73	11	47	38	10	77	68	41	12	78	63	17	6
96	5	7	50	75	28											

A matriz resultado será:

	1	2	3	4	5	6	7	8	9	10	11	12	13
+	1	5	2	5	6	7	10						
2	0	4	11	16	29	37	0	1	12	5	13	4	
3	1	2	68	71	0	0	0						
4	1	4	38	40	41	42	0						
5	1	3	17	22	28	0	0						
6	1	2	46	47	0	0	0						
7	1	3	89	92	96	0	0						
* 8	0	1	44	0	0	0	0	2	9				
9	0	4	49	65	73	80	0	6	11	3	10	7	
10	1	4	75	77	78	79	0						
11	1	4	50	52	57	63	0						
12	1	2	12	14	0	0	0						
13	1	2	32	35	0	0	0						

As perguntas e as respostas:

1. Qual a primeira chave do nodo raiz ? R1:

2. Quantas são as linhas de M que tem 3 chaves válidas ? R2:_____.

3. Quanto é a soma vertical da coluna 8 de M ? R3:_____.

4. O valor da última chave válida na linha 3 de M é: R4:_____.

Para você fazer

Os 40 numeros a incluir são

66	22	69	39	20	2	24	26	30	75	96	63	45	8
97	49	50	84	54	35	6	88	32	74	92	62	37	61

Para ajudar, segue um esqueleto de árvore B

F	C	T	K	K	K	K	C	C	C	C	C	C	C
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													

Construa a árvore e responda as questões:

Qual a primeira chave do nodo raiz ?	Quantas são as linhas de M que tem 4 chaves válidas ?	Quanto é a soma vertical da coluna 10 de M ?	O valor da última chave válida na linha 6 de M é

Bibliografia para esta folha: ALGORITMOS, Cormen et alli.



Simulando os índices do SYBASE

O SYBASE usa árvores B para indexar as tabelas. Existem 2 tipos de árvores B: as granuladas e as não granuladas (clustered). As granuladas são árvores esparsas (nem todo registro de dados tem entrada nos índices) nas quais os registros (na tabela original) são mantidos em ordem dos valores de índice e apenas o primeiro registro em cada página de dados tem uma entrada de índice. rvores não granuladas são densas e cada um dos registros da tabela tem 1 entrada de índice.

Uma tabela SYBASE com índice granulado exige que seus elementos sejam mantidos em ordem dentro da tabela. Assim, inclusões podem se dar em qualquer página, a depender do conteúdo do campo de índice. Tabelas SYBASE que tem apenas índices não granulados fazem as inclusões de novos elementos sempre ao final das tabelas (na última página).

Tabelas SYBASE podem ter apenas 1 índice granulado, mas podem ter até 250 índices não granulados.

Na criação de um índice granulado, os dados da tabela devem estar colocados nas páginas em ordem.

Cada entrada de índice em sybase pode ser composta por até 16 campos e ter até 256 bytes. Estes campos não precisam estar contíguos. Só não podem ser partes de campos e resultados de cálculos.

O tamanho da página varia, mas o tamanho mais comum é 2K. Existem 5 tipos de páginas em sybase, a saber:

Páginas de Dados contém registros de dados ou registros de log. Ambos são estruturalmente diferentes, mas são construídos do mesmo jeito.

Páginas de índices contém registros de índice

Páginas de texto ou imagem contém BLOBS

Páginas de alocação contém estruturas de dados usadas para geremciar o processo de alocação de páginas

páginas de Estatísticas contém estatística de distribuição e uso para os índices.

Página de Dados Se os registros tem tamanho fixo, não haverá tabela de deslocamento na página e o endereço de cada registro é obtido mediante simples cálculo. Neste caso, o tamanho do registro (fixo) é guardado no campo TAMANHO MÍNIMO DE REGISTRO do header.

O header da página é um conjunto de 32 bytes contendo (entre outros): número lógico da página, anterior e próxima página lógica, identificação do objeto ao qual esta página pertence, primeira entrada de registro disponível nesta página, deslocamento do espaço livre, tamanho mínimo do registro

Depois vem a área de registros, que contém um número inteiro de slots para guardar registros. Um registro não atravessa limites de páginas

Tabela de deslocamento: Um conjunto de deslocamentos desde o início da página dando o início de cada um dos registros. Este conjunto cresce do fim da página para o início e só existe quando o tamanho do registro é variável.

Cada registro na página recebe um número de registro. É um campo de 1 byte e assim restringe-se o número de registros em uma página a 256. O número do registro usado nos índices não granulados (densos) é uma combinação do número da página com o número do registro na página.

O registro pode ocupar a página inteira (2048-32=2016). Não há tamanho mínimo para o registro, mas só pode haver 256 registros na página.

Os índices SYBASE são organizados na forma de arvores B + (o + significa que há uma ligação adicional entre as folhas, para permitir o processamento sequencial dos dados).

No exemplo aqui estudado, vão-se fazer algumas considerações/simplificações descritas a seguir.

Suponha um arquivo de pessoas contendo nome, cor preferida, bicho de estimação, alimento que gosta e idade.

Construindo um arquivo SYBASE contendo estes dados e que esteja sujeito às seguintes características:

- Registros de tamanho fixo igual a 40 bytes (9+9+9+9+4).
- Um índice granulado ou esparsos (clustered) por um dos campos. No exemplo que é igual para todos, o índice esparsos é por cor.
- Quatro índices não granulados (densos) pelos demais atributos.
- Páginas de dados com espaço para 4 registros e fator de preenchimento de 4:1 (usar apenas 3 espaços na carga original).
- Páginas de índices com espaço para 5 chaves (nome, cor, bicho, alimento ou idade). Lembrar que este número é ímpar, o que indica que no *split* vai subir a terceira chave (das 5 que existem).
- A identificação de cada registro deve ser página:número do registro na página. Por exemplo, a identificação 3:2, indica o segundo registro da terceira página.
- Nas páginas de índice (árvores-B) despreze as informações de endereço dos filhos. No exemplo, elas são irrelevantes. Faça como no exemplo.
- Os dados de entrada estão ordenados pela coluna do índice granulado.

Exemplo: Seja o conjunto de dados, no qual se fará o granulado por cor:

NOME	COR	ANIMAL	COMIDA	IDADE
josé	dourado	chopim	feijão	34
gabi	laranja	perú	aipim	22
sueli	lilas	gato	alho	40
antônio	musgo	vaca	batata	41
tatiana	petróleo	cavalo	arroz	37
juca	preto	águia	goiaba	36
romeu	roxo	dragão	maracujá	27
paulo	verde	urubú	rabanete	20

Com este conjunto de dados, ter-se-á a seguintes disposição nos arquivos

Páginas de Dados					
Pág1	josé	dourado	chopim	feijão	34
	gabi	laranja	perú	aipim	22
	sueli	lilas	gato	alho	40
Pág2	antônio	musgo	vaca	batata	41
	tatiana	petróleo	cavalo	arroz	37
	juca	preto	águia	goiaba	36
Pág3	romeu	roxo	dragão	maracujá	27
	paulo	verde	urubú	rabanete	20

A seguir, as páginas de índices

Páginas de índices	
Índice I1	granulado por COR
Página 1 ←raiz	dourado(1:1), musgo(2:1), roxo(3:1)
Índice I2	por NOME
Página 1	antônio(2:1), gabi(1:2)
Página 2 ←raiz	jose(1:1)
Página 3	juca(2:3), paulo(3:2), romeu(3:1), sueli(1:3), tati-ana(2:2)
Índice I3	por BICHO
P1	águia(2:3), cavalo(2:2), chopim(1:1), dragão(3:1)
P2 ←raiz	gato(1:3)
P3	perú(1:2),urubú(3:2), vaca(2:1)
Índice I4	por COMIDA
P1	aipim(1:2),alho(1:3)
P2 ←raiz	arroz(2:2)
P3	batata(2:1), feijão(1:1), goiaba(2:3), mara-cujá(3:1), rabanete(3:2)
Índice I5	por IDADE
P1	20(3:2), 22(1:2), 27(3:1), 34(1:1), 36(2:3)
P2 ←raiz	37(2:2)
P3	40(1:3), 41(2:1)

🔧 Para você fazer

Simule o funcionamento do SYBASE e construa 5 índices para o seguinte conjunto de dados:

NOME	COR	ANIMAL	COMIDA	IDADE
laura	vermelho	cachorro	abacate	20
romeu	branco	cegonha	rabanete	31
zulmira	musgo	dragao	maracuja	19
eduardo	azulao	jacare	cebola	41
jose	azul	papagaio	goiaba	44
gloria	verde	peru	figo	17
paulo	cinza	pomba	uva	27
nando	sepia	veado	coco	43

Sendo: I_1 : granulado por animal

I_2 por nome ; I_3 por cor ; I_4 por comida e I_5 por idade .

Responda:

a) Página 1 do índice 1	
b) Página 3 do índice 2	
c) Página 1 do índice 3	
d) Página 2 do índice 4	
e) Página 2 do índice 5	



- 1 - /



Exercício : 1-

_____ / _____ / _____

Identificação

Este exercício deverá ser feito pelos seguintes alunos

1

xBASE

O dbase foi o primeiro produto de banco de dados para uso em micro-computadores. Lançado em 1980 como dbase2 (nunca existiu um dbase 1), tinha limitações que hoje pareceriam inaceitáveis, mas na época foi uma revolução.

De 82 a 87 reinou soberano, e acabou criando uma genealogia de produtos e metodologias (dbase 2, 3, 3plus, 4, 5, Visualdbase 7, FoxPro, FoxBaseParadox, Clipper, harbour, só para ficar nos mais conhecidos).

Sua importância hoje reside no fato de que 100% dos produtos que iteração com algum tipo de banco de dados em microcomputadores aceitam o padrão dbase. Assim, por exemplo, o AutoCad aceita dados em formato dbase. O MSWord aceita dados para mala direta no formato dbase e assim por diante.

Também é importante estudá-lo pela simplicidade do esquema e pelo fato de que funciona e bem. Finalmente, por questões de propriedade de marca, o nome dbase foi abandonado (já que o nome – mas não os formatos – é registrado pelos proprietários, originalmente a Ashton Tate e depois a Borland) e em seu lugar usa-se o termo xbase.

Um banco de dados XBASE é um conjunto de diversos arquivos, cujos principais são

Tipo de arquivo	Conteúdo
DBF	Data Base File: o arquivo que contém os dados. Tem registros de tamanho fixo, e as inclusões sempre se dão ao final do arquivo. Exclusões mantêm o registro no seu lugar, até que ocorra um PACK. Atualizações se dão no mesmo lugar.
NDX	Index File. O arquivo organizado na forma de uma árvore B+. Os apontadores para o número do registro (no DBF equivalente) só existem nas folhas.
DBT	Data Base Text. Arquivo que contém os campos MEMORANDO que porventura sejam definidos no DBF. Lá fica apenas um apontador de 10 bytes para o endereço do memorando equivalente que está no arquivo DBT.

O dbase armazena os dados brutos em arquivos cuja extensão é DBF (data base file). Cada registro que é incluído o é no fim do arquivo e ele é numerado sequencialmente no instante em que entra no arquivo.

No começo do arquivo, vem o lay-out dos dados. Seria o header, com o seguinte formato

Posição	Significado
0	Assinatura
1-3	Data da última atualização
4-7	Número de registros no arquivo
8-9	Tamanho da estrutura de cabeçalho
10-11	Tamanho de cada registro
12-31	Reservado
...	campo-1
...	
...	Campo _n
...	X"0D"(terminador dos dados)

Cada campo, por sua vez, está assim descrito (ocupando 32 bytes cada)

Posição	Significado
0-10	Nome do campo em ASCII
11	Tipo de campo (A, N, L, D, M)
12-15	Reservado
16	Tamanho do campo
17	Quantidade de decimais
18-31	Reservado

Notas

- Os tipos de campo aceitáveis são: Alfanumérico com tamanho de até 256 bytes, numérico podendo ou não ter decimais, lógico, data e memorando.
- Algumas assinaturas importantes: x'02' é foxbase, h'03' é dbase sem arquivos memo, x'30' é VisualFoxPro, x'83' é dbase com campos memo, etc.
- O tamanho de cada registro é a soma dos tamanhos dos campos do arquivo, mais 1 (que é usado como sinalizador de exclusão).
- As páginas em dbase ocupam 512 bytes.
- Depois do header, vem os dados, gravados sequencialmente, um depois do outro.

Estrutura dos índices Os índices em dbase são guardados em arquivos separados, o que – como tudo na vida – tem vantagens e desvantagens. Vantagem: por definição, índices são descartáveis, e não precisam ser guardados. Desvantagem: eles podem ser ignorados em operações de inclusão / exclusão, o que diminui a confiabilidade deles.

Os índices são criados através de um comando dbase que recebe os parâmetros que ajudarão a montar o índice, lê o arquivo de dados e constrói o índice. Ele usa uma estrutura de árvore B+, que vem a ser uma árvore B, na qual as informações de chave estão apenas nas folhas, o que permite a leitura "sequencial" do arquivo de dados em qualquer ordem determinada por um arquivo de índices.

Lay-out do arquivo NDX

As páginas do arquivo são de 512 bytes. A primeira página contém o header, e as demais as páginas da árvore B+.

Layout do header

Posição	Significado
0-3	Qual é a página RAIZ
4-7	Total de páginas que formam este arquivo NDX
8-11	reservado
12-13	comprimento da chave
14-15	quantidade de chaves em cada página
16-17	tipo da chave (0=char, 1=num)
18-21	comprimento da entrada (chave como multiplo de 4 + 4 + 4)
22	reservado
23	flag de chave única (0=não)
24-511	string definidor da chave (comando INDEX...)

Layout dos demais registros

Cada registro de arquivo NDX é um nodo de uma árvore B+.

Posição	Significado
0-3	quantidade de entradas válidas nesta pagina
4-511	array de entradas
	0-3 (prim) apontador para o nível inferior (filho)
	4-7 (seg) número do registro no arquivo DBF
	se primeiro $\neq$ 0 e segundo = 0: este nodo é intermediário
	se primeiro = 0 e segundo $\neq$ 0: este nodo é folha
	8-n conteúdo da chave de pesquisa

(números armazenados como IEEE double = binários)

Algoritmo de pesquisa

- Sempre se começa na raiz (que fica na memória da máquina)
- A pagina é pesquisada sequencialmente até uma chave $\geq$ ser achada
- A pesquisa segue até o apontador para o nível inferior ser igual a zero. Se a chave é encontrada o número do registro DBF é devolvido

🔗 Para você fazer

Para este exercício considere as seguintes estruturas

arquivo PESSOA.DBF

NOME,C,65

CORX,C,55

BICH,C,60

COMI,C,61

IDAD,N,4

E os comandos

USE PESSOA

INDEX ON NOME TO INOME (cria o arquivo INOME.NDX)

INDEX ON CORX TO ICORX (cria o arquivo ICORX.NDX)

INDEX ON BICH TO IBICH (cria o arquivo IBICH.NDX)

INDEX ON COMI TO ICOMI (cria o arquivo ICOMI.NDX)

INDEX ON IDAD TO IIDAD (cria o arquivo IIDAD.NDX)

Estudem o DUMP do arquivo INOME.NDX , anexo e respondam as seguintes perguntas

- Este arquivo usa o campo _____ como chave
- A raiz está na página _____. Quantas páginas há ?
- O comprimento da chave é _____ e existem chaves / página
- O tipo da chave é _____ e o comprimento de cada entrada é _____.
- Desenhe no verso, a árvore B+ que este arquivo representa
- Qual é a sequência de registros (números decimas dos registros) do arquivo DBF que deve ser processada quando este arquivo de indice é usado ? Responda no verso.

Bibliografia de xbase: <http://www.e-bachmann.dk/docs/xbase.htm> ☐

B-Árvores com grau t=5

Suponha uma b-árvore M com t=5. Seus nodos têm entre 4 e 9 filhos. M é uma matriz de 21 colunas (indicativo de folha/qtde de chaves válidas/9chaves/10filhos). M já contém 200 números.

- Eis o significado das colunas:
- 1 indicativo de folha (1=sim; 0=não)
 - 2 quantidade de chaves válidas neste nodo (nem sempre é zero)
 - 3 a 11 chaves
 - 12 a 21 endereços dos filhos associados a cada chave

Veja um exemplo

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	+-----																		
1	1	7	5	11	14	22	23	26	27	0	0								
2	0	5	31	66	85	117	153	0	0	0	0	1	16	6	28	18	10		
3	1	7	517	518	525	528	555	557	565	0	0								
4	1	7	313	315	317	324	326	327	332	0	0								
5	1	5	787	789	796	797	798	0	0	0	0								
6	1	4	67	73	77	84	0	0	0	0	0								
7	1	6	912	915	923	930	931	944	0	0	0								
8	1	6	383	389	391	394	402	408	0	0	0								
9	1	9	672	677	687	688	691	697	700	703	704								
10	1	7	157	162	166	170	180	181	184	0	0								
11	1	7	461	464	466	469	472	478	479	0	0								
*12	0	3	199	459	718	0	0	0	0	0	0	2	25	13	22				
13	0	5	480	509	581	615	663	0	0	0	0	11	21	3	29	20	9		
14	1	6	851	853	864	881	890	900	0	0	0								
15	1	7	985	986	987	990	996	998	999	0	0								
16	1	8	36	37	44	45	46	53	62	63	0								
17	1	5	721	722	723	750	755	0	0	0	0								
18	1	9	118	124	128	137	140	141	143	145	152								
19	1	5	240	248	252	266	268	0	0	0	0								
20	1	8	618	628	633	634	635	641	643	647	0								
21	1	7	482	484	497	498	500	502	507	0	0								
22	0	7	758	776	799	847	907	951	983	0	0	17	30	5	32	14	7	27	15
23	1	9	203	205	206	215	220	221	222	228	236								
24	1	7	350	351	358	360	364	374	378	0	0								
25	0	6	239	273	308	347	380	421	0	0	0	23	19	31	4	24	8	26	
26	1	7	424	425	432	438	444	452	454	0	0								
27	1	6	952	961	967	974	978	981	0	0	0								
28	1	7	95	97	98	104	112	114	115	0	0								
29	1	4	595	602	603	607	0	0	0	0	0								
30	1	4	761	766	770	775	0	0	0	0	0								
31	1	6	275	278	298	299	300	307	0	0	0								
32	1	4	807	835	838	841	0	0	0	0	0								

Após a inclusão nesta árvore dos valores:

379 582 245 219 954 573 566 564 562 560 558 569 571 568 567 559 572 570 561

Veja como ficou: (LINHAS IGUAIS NÃO REPETIDAS)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	+-----																		
3	1	4	517	518	525	528	0	0	0	0	0								
13	0	7	480	509	555	564	581	615	663	0	0	11	21	3	34	35	29	20	9
19	1	6	240	245	248	252	266	268	0	0	0								
23	1	5	203	205	206	215	219	0	0	0	0								
24	1	8	350	351	358	360	364	374	378	379	0								
25	0	7	220	239	273	308	347	380	421	0	0	23	33	19	31	4	24	8	26
27	1	7	952	954	961	967	974	978	981	0	0								
29	1	5	582	595	602	603	607	0	0	0	0								
33	1	4	221	222	228	236	0	0	0	0	0								
34	1	7	557	558	559	560	561	562	563	0	0								
35	1	9	565	566	567	568	569	570	571	572	573								

👉 Para você fazer

Seja a árvore B (mesma especificação acima)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	+-----																				
1	1	8	2	7	11	14	16	17	19	24	0										
2	0	4	25	76	110	147	0	0	0	0	0	1	16	6	25	9					
3	1	6	550	555	557	560	565	567	0	0	0										
4	1	8	265	266	268	274	279	282	287	290	0										
5	1	9	762	763	771	787	788	791	794	798	801										
6	1	6	85	94	96	97	103	105	0	0	0										
7	1	6	634	653	671	678	679	687	0	0	0										
8	1	9	389	396	399	409	410	415	418	419	421										
9	1	5	148	165	167	170	172	0	0	0	0										
10	1	4	945	948	955	957	0	0	0	0	0										
11	1	8	295	311	318	321	325	329	344	346	0										
*12	0	3	180	383	689	0	0	0	0	0	0	0	2	28	13	21					
13	0	5	422	473	548	572	630	0	0	0	0	8	23	17	3	20	7				
14	1	6	694	696	702	712	722	726	0	0	0										
15	1	5	863	879	883	901	905	0	0	0	0										
16	1	7	34	35	37	41	48	63	64	0	0										
17	1	9	474	476	489	493	494	497	519	525	539										
18	1	5	351	353	354	355	363	0	0	0	0										
19	1	7	971	977	979	984	993	994	996	0	0										
20	1	8	573	578	580	588	597	613	622	624	0										
21	0	8	728	760	807	853	911	939	959	970	0	14	22	5	24	15	29	10	30	19	
22	1	8	729	730	732	734	736	745	748	759	0										
23	1	9	429	432	439	452	453	458	461	463	471										
24	1	5	823	824	828	829	833	0	0	0	0										
25	1	9	117	123	126	128	129	134	139	140	146										
26	1	5	232	237	246	251	259	0	0	0	0										
27	1	6	192	195	198	205	206	211	0	0	0										
28	0	5	226	262	291	349	366	0	0	0	0	27	26	4	11	18	31				
29	1	7	914	919	920	928	931	935	936	0	0										
30	1	6	962	963	965	966	967	968	0	0	0										
31	1	4	369	374	376	378	0	0	0	0	0										

Você deve incluir nesta árvore os valores:

865 545 228 310 302 960 961 972 950 953 958 949 973 956 952 964 954 974 969 951

e a seguir responder:

M[15][7]	M[11][10]	M[19][9]	M[13][2]	M[37][7]

Bibliografia para esta folha: ALGORITMOS, Cormen et alli.



- 1 - /

Bits e Bytes

Os computadores usam lógica binária. Vai daí que suas memórias organizam-se em dígitos binários (bits=BInary digiT). Assim, quando olha-se um conteúdo qualquer, o que se vê são apenas zeros e uns. É o chamado conteúdo binário. Ele não é muito prático, pois sempre é extenso (o número 1.300.000₍₁₀₎ é igual a 100111101011000100000₍₂₎ em binário) razão pela qual surgiu a representação hexadecimal, que é uma abreviatura do código binário. Agora cada 4 bits são representados por um único dígito hexadecimal. Com isso, o comprimento do conteúdo é 4 vezes menor. O mesmo 1.300.000₍₁₀₎ agora é 13D620₍₁₆₎. Juntando 8 bits (2 dígitos hexadecimais) tem-se um BYTE. Portanto o número 1.300.000₍₁₀₎ ocupa 3 bytes, a saber: X'13.D6.20'.

Codificação e decodificação

Humanos não têm muita habilidade para lidar com conteúdos binários, nem mesmo hexadecimais. Assim, surgiram os códigos que permitem traduzir conteúdos humanos (letras e números decimais) em conteúdos computadorizados (binários). Há muitos códigos distintos. Este exercício vai lidar com o ASCII e com dois códigos inventados: Ortelino e Trocaletra.

O código ASCII utiliza 8 bits para representar cada caracter. Este código (até pelo seu nome) é um código padrão e é reconhecido por todos os equipamentos e softwares que o manuseiam. Entretanto, nada impede, que:

- outros códigos de 8 bits/caractere existam e sejam usados (por exemplo EBCDIC ou Z-CODE).
- outros códigos com número diferente de 8 bits/caractere sejam usados. Assim, tem-se o UNICODE e o ORTELINO (com 5 bits/caractere). Note-se que quanto maior o número de bits, mais caracteres podem ser representados. O UNICODE tem espaço para mais de um milhão de caracteres, seguramente mais do que é necessário para representar todos os caracteres em todos os alfabetos em uso aqui na Terra. Da mesma maneira, ORTELINO só consegue representar 32 caracteres distintos.
- Finalmente, nada impede que se tenham códigos que representam os caracteres em um número *variável* de bits. O que se paga em aumento de complexidade no manuseio, se ganha em economia, já que nestes casos, os caracteres mais frequentes usarão menos bits para serem representados. Uma das encarnações do UNICODE (UTF-8) é assim. O código TROCALETRA é outro exemplo de código deste tipo.

Suponha o código de caracteres (ASCII convencional):

A=X'41' H=X'48' O=X'4F' V=X'56' c=X'63' j=X'6A' q=X'71' x=X'78'
B=X'42' I=X'49' P=X'50' W=X'57' d=X'64' k=X'6B' r=X'72' y=X'79'
C=X'43' J=X'4A' Q=X'51' X=X'58' e=X'65' l=X'6C' s=X'73' z=X'7A'
D=X'44' K=X'4B' R=X'52' Y=X'59' f=X'66' m=X'6D' t=X'74' =X'20'
E=X'45' L=X'4C' S=X'53' Z=X'5A' g=X'67' n=X'6E' u=X'75'
F=X'46' M=X'4D' T=X'54' a=X'61' h=X'68' o=X'6F' v=X'76'
G=X'47' N=X'4E' U=X'55' b=X'62' i=X'69' p=X'70' w=X'77'

A especificação X'zz', indica que os dígitos zz representam um número hexadecimal, correspondente a 8 bits (sendo 4 + 4).

Suponha a tabela binária:

0=0000, 1=0001, 2=0010, 3=0011, 4=0100, 5=0101, 6=0110, 7=0111
8=1000, 9=1001, A=1010, B=1011, C=1100, D=1101, E=1110, F=1111

Faça as seguintes conversões:

1. De Hexadecimal para caracteres

Exemplo:
X'4F 75 76 69 72 61 6D 20 64 6F 20 49 70 69 72 61 6E 67 61'
Ouviram do Ipiranga

4C756973616F206E616F20652061746163616E74

2. De Binário para caracter

Exemplo:
B'0100 0010 0110 0001 0110 0011 0110 0001 0110 1110 0110 0001'
Bacana

010110000110100101100011011000010111001001100001

Suponha agora o código ORTELINO

A=B'00000' F=B'00101' K=B'01010' P=B'01111' U=B'10100' Z=B'11001'
B=B'00001' G=B'00110' L=B'01011' Q=B'10000' V=B'10101' =B'11010'
C=B'00010' H=B'00111' M=B'01100' R=B'10001' W=B'10110' -B'11011'
D=B'00011' I=B'01000' N=B'01101' S=B'10010' X=B'10111' ,=B'11100'
E=B'00100' J=B'01001' O=B'01110' T=B'10011' Y=B'11000' .=B'11101'

3. Converta de caracteres para ORTELINO e depois represente em hexadecimal

Exemplo:
IVO VIU A UVA : X'45 5D AA A2 9A 06 A9 50 00', com 65 bits

VINTE E UM / tam= 50

Represente os últimos 3 bytes apenas: :_____,_____,_____.

4. Converta de ORTELINO (em hexa) para caractere

Exemplo:
X'72 34 D7 22 5A 04 28 80 com 60 bits : OI NOIS AQUI

tam= 50 / 621A8D18137480

:_____.

Suponha agora o código TROCALETRA (de exemplo, igual para todos)

B=B'0001 ' L=B'1001 ' X=B'00001 ' M=B'000001 '
A=B'0010 ' O=B'1010 ' H=B'01000 ' G=B'100000 '
P=B'0011 ' E=B'1011 ' =B'01001 ' R=B'100001 '
U=B'0101 ' D=B'1101 ' Q=B'10001 ' N=B'0000001 '
Z=B'0110 ' S=B'1110 ' V=B'11000 ' F=B'00000000 '
I=B'0111 ' T=B'1111 ' J=B'11001 ' C=B'00000001 '

O que significa: X'01.28.48.0C.15.C0' com 43 bits ? :_____.

Mais um, para você ver se entendeu: X'DA 92 86 F8', com 30 bits ? R='DOLARES'.

Suponha agora **OUTRO TROCALETRA**, que é o que você deve usar no seu exercício, e que É DIFERENTE PARA CADA FOLHA. (sugestão: risque o Trocaletra do exemplo acima, para não se confundir).

T=B'0001 ' F=B'1001 ' I=B'00001 ' P=B'000001 '
V=B'0010 ' C=B'1010 ' H=B'01000 ' J=B'100000 '
Q=B'0011 ' B=B'1011 ' D=B'01001 ' Z=B'100001 '
M=B'0101 ' G=B'1101 ' U=B'10001 ' S=B'0000001 '
R=B'0110 ' =B'1110 ' A=B'11000 ' N=B'00000000 '
L=B'0111 ' E=B'1111 ' O=B'11001 ' X=B'00000001 '

5. Converta de TROCALETRA (em hexa) para caractere

Exemplo:
A8B0443780 / tam= 36 - CURITIBA

tam= 52 / 4840D679390810

:_____.

6. Converta de caractere para TROCALETRA e depois represente em hexadecimal

Exemplo:
PARANA / tam= 33 - 070D800C00

DE GLORIAS / tam= 47

Represente os últimos 3 bytes apenas: :_____,_____,_____.

Observação

Perceba que QUALQUER computador desses que há por aí, escreve 8 bits quando quer gravar (ou ler) alguma coisa. Quando 1 caracter = 8 bits não ocorre nenhuma dificuldade. Por isso esse número (8) foi o escolhido.

Assim, quando se codifica em ASCII (ou em EBCDIC ou em qualquer código fixo com comprimento de 8 bits) **não** há necessidade de especificar a quantidade de bits utilizados na escrita. Quando a escrita acabar, acabou o código. Isto não é verdade quando se utiliza um código que tenha algo diferente de 8 bits por caracter. Por exemplo, se se utilizam 5 bits por caracter, ao escrever um único caracter usam-se 5 bits, mas não há como escrever apenas 5 bits em qualquer computador. Vai daí que se complementa o código com 3 zeros à direita (*stuffing bits*) e escrevem-se como sempre 8 bits por caracter. Mas, há necessidade de avisar que dos 8 bits escritos apenas 5 interessam.

Com muita mais razão há que se indicar os bits válidos, quando o código usar um número variável de bits por caracter.

Esta observação está aqui para alertá-lo da dificuldade com o último BYTE. Não é questão trivial. Faça força para entender a jogada, e se não conseguir, não se avexe, peça ajuda.

1



Compressão usando o algoritmo de Huffman

A codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo. Ele foi desenvolvido em 1952 por David A. Huffman que era, na época, estudante de doutorado no MIT, e foi publicado no artigo "A Method for the Construction of Minimum-Redundancy Codes".

Uma árvore binária completa, chamada de árvore de Huffman é construída recursivamente a partir da junção dos dois símbolos de menor probabilidade, que são então somados em símbolos auxiliares e estes símbolos auxiliares recolocados no conjunto de símbolos. O processo termina quando todos os símbolos foram unidos em símbolos auxiliares, formando uma árvore binária. A árvore é então percorrida, atribuindo-se valores binários de 1 ou 0 para cada aresta, e os códigos são gerados a partir desse percurso.

A codificação de Huffman tem desempenho ótimo quando as probabilidades dos símbolos são potências negativas de dois (2^{-1} , 2^{-2} , ...). A codificação gerada tem também a garantia de ser não ambígua, pois nenhum código pode ser o prefixo de outro código.

Este algoritmo começa pela análise das frequências de ocorrência de cada um dos caracteres do arquivo a comprimir.

Suponhamos, a título de exemplo, um arquivo que contenha ABCDEFABCDEFABCABAABD

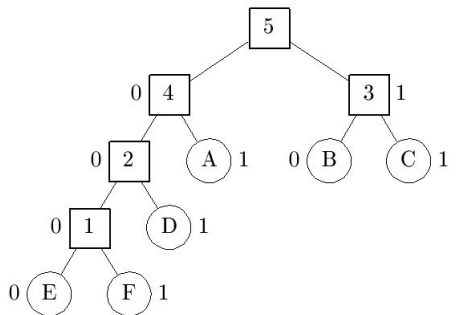
Contabilizando as ocorrências de cada caractere, ficamos com as seguintes frequências: B=25%, E=10%, A=30%, C=15%, D=15%, e F=5%.

A seguir, o algoritmo prevê a colocação desta tabela em ordem decrescente de ocorrências. No exemplo, fica: A=30%, B=25%, C=15%, D=15%, E=10% e F=5%.

Na terceira etapa, é criada uma árvore binária, onde as folhas representam os caracteres a comprimir, e o número de nós é o número de folhas diminuído de uma unidade. A regra para a montagem da árvore é: cada nó terá dois sub-nós, sendo que o caractere mais frequente ficará sempre à esquerda. Se houver frequências iguais entre nó e folha, colocamos à esquerda o nó. Vejamos no exemplo:

Comparam-se os primeiros (de baixo para cima, ou seja os menores), dois elementos, E (10%) e F (5%). Então o E fica à esquerda e o F à direita. Agrupamos o E e o F sob um nó, a quem chamaremos 1, que passará a ter a frequência dos dois somados (15%). Continuamos a buscar os 2 menores valores, que agora são o nó 1 e o caractere D. (Note que aqui há um empate, assim escolhemos quaisquer dois menores). Agrupamos os 2, formando o nó 2, ao qual estão subordinados o nó 1 (a esquerda) e a letra D (a direita). O nó 2 tem frequência 30%.

Os próximos 2 menores são o B e o C. Cria-se o nó 3 que os agrega, com frequência 40%. Os próximos são o nó 2 e a letra A, que criam o nó 4, com frequência 60%. Finalmente encerra a árvore o nó 5, que agrega os nós 4 e 3, com frequência 100%.



O que resultaria nos seguintes códigos de bit

A	01
B	10
C	11
D	001
E	0000
F	0001

A partir desta árvore, cada caractere a transmitir será enviado pelos bits que permitem o caminharmento pela árvore, entendendo-se o bit zero como "esquerda" e o bit 1 como "direita".

No nosso exemplo, ao transmitir a mensagem ABAEC, enviaríamos 01.10.01.0000.11, com os pontos colocados apenas por clareza. Note-se que os caracteres de maior frequência terão comprimento menor, o que caracteriza a compressão e o conseqüente ganho em termos de transmissão. Por outro lado, olhando a mensagem, sem conhecer a árvore original é muito difícil determinar onde começa e termina cada caractere, o que torna a sua quebra bem difícil.

Para determinarmos o comprimento do arquivo a mandar, e que em última análise determinará a economia pela compressão, usamos a regra do comprimento ponderado da árvore gerada. Isto significa que cada nó deverá gerar uma multiplicação entre o comprimento em bits do caractere representado no nó pela sua frequência de ocorrência.

No nosso exemplo, isto ficaria:

$$\begin{aligned} A &= 2 \times 0.30 = 0.60 \\ B &= 2 \times 0.25 = 0.50 \\ C &= 2 \times 0.15 = 0.30 \\ D &= 3 \times 0.15 = 0.45 \\ E &= 4 \times 0.10 = 0.40 \\ F &= 4 \times 0.05 = 0.20 \end{aligned}$$

A somatória destes valores é, em bits, o comprimento médio de cada caractere neste arquivo. No nosso exemplo, a soma deu 2.45, e como são 20 caracteres no total, o comprimento do arquivo comprimido (2.45×20) é 49 bits. Existe um teorema, que não será demonstrado, que informa ser a árvore construída segundo o critério visto acima, a árvore de menores caminhos ponderados, ou seja a árvore de Huffman é a solução ótima para este problema.

Etapas do Algoritmo

1. Contagem dos caracteres e obtenção da tabela de frequências
2. ordenação da tabela de frequências
3. Criação da árvore, escolhendo sempre as 2 menores frequências (maior à esquerda). Se houver empate entre nodo pai [] e nodo folha (), [] à esquerda. Após excluir os 2 nodos usados, incluir o pai com a soma.
4. Marcar a árvore: 0 a esquerda e 1 a direita.
5. Para descomprimir: Converter a mensagem de hexa para binário. Começar na raiz da árvore e seguir o string até encontrar uma folha. Anotar o caractere e retornar a raiz.
6. Para comprimir: Escrever um dicionário de caracteres a partir da árvore. Converter a mensagem a comprimir em string binário, transformando-a em hexadecimal.

Para você fazer

O universo do qual serão extraídas as mensagens é:

AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
EEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEE
SSSSSSSSPPPPPPPPPPPP
SSSSSSSSPPPPPPPPPPPP
MMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMM
IIIIIIIIIIUUUUUUUUUU
IIIIIIIIIIUUUUUUUUUU
TTTTTTTTTTTBBBBBB
BBBBBBBBBBBBBBBBNN
NNNNNNNNNNFFFFFQQ
QQQQQQGGGGGGHHHHJJ
JJVVVXXZ

Para ajudar na contagem, saiba que cada caractere da lista acima tem 1 unidade a mais que o caractere que vem depois.

Lembre que todas as mensagens contém o nome de uma peça de Shakespeare, seguido por 3 espaços em branco e seguido por uma fala de um dos personagens.

1. Descomprima a frase abaixo e escreva-a em português:

89 F0 54 75 45 97 9C 44 69 59 6D 68
69 62 6E 36 4E 1E 04 DC 74 C7 3E A2
DC E6 E2 4E 2E B9 F5 8D F1 38

Tamanhos antes= 34 depois= 65 caracteres. Bits significativos= 269

FRASE: _____

2. Comprima a frase a seguir e diga qual o byte pedido:

ROMEU/E/JULIETA//REVESTI/VOS/DE/PAC
IENCIA/SENHOR/VOS/PECO

Tamanhos antes= 58 depois= 32 . Para sua conferência, os 2 primeiros bytes comprimidos são: 63 A5 . O byte imediatamente anterior ao pedido é 65

Responda qual o 31 º byte:

_____ (responda em hexadecimal, usando dois dígitos de 0 a F).





Huffman Dinâmico

Acompanhe o seguinte exemplo real. Seja a co-
dificação da mensagem FRIFRRR, contendo 7
bytes. Eis o desenvolvimento dos algoritmos
conforme estudados na aula teórica.

```
////////// F  \\\\\\\\\\\\\\\\\\\
incluindo F
Atualizacao de Nodo
  0:-1 1 2
  1: 0-1 1
  2: 0-1 1
=====
////////// R  \\\\\\\\\\\\\\\\\\\
incluindo R
troca 1 2
ANTES
  0:-1 1 2
  1: 0-1 1
  2: 0 3 2
  3: 2-1 1
  4: 2-1 1
FR*
  1 3 4
DEPOIS
  0:-1 1 2
  1: 0 3 2
  2: 0-1 1
  3: 1-1 1
  4: 1-1 1
FR*
  2 3 4
-----
Atualizacao de Nodo
  0:-1 1 3
  1: 0 3 2
  2: 0-1 1
  3: 1-1 1
  4: 1-1 1
=====
////////// I  \\\\\\\\\\\\\\\\\\\
incluindo I
troca 2 4
ANTES
  0:-1 1 3
  1: 0 3 2
  2: 0-1 1
  3: 1-1 1
  4: 1 5 2
  5: 4-1 1
  6: 4-1 1
FIR*
  2 5 3 6
```

```
DEPOIS
  0:-1 1 3
  1: 0 3 2
  2: 0 5 2
  3: 1-1 1
  4: 1-1 1
  5: 2-1 1
  6: 2-1 1
FIR*
  4 5 3 6
-----
Atualizacao de Nodo
  0:-1 1 4
  1: 0 3 2
  2: 0 5 2
  3: 1-1 1
  4: 1-1 1
  5: 2-1 1
  6: 2-1 1
=====
////////// F  \\\\\\\\\\\\\\\\\\\
troca 3 4
ANTES
  0:-1 1 4
  1: 0 3 2
  2: 0 5 2
  3: 1-1 1
  4: 1-1 2
  5: 2-1 1
  6: 2-1 1
FIR*
  4 5 3 6
DEPOIS
  0:-1 1 4
  1: 0 3 2
  2: 0 5 2
  3: 1-1 2
  4: 1-1 1
  5: 2-1 1
  6: 2-1 1
FIR*
  3 5 4 6
-----
Atualizacao de Nodo
  0:-1 1 5
  1: 0 3 3
  2: 0 5 2
  3: 1-1 2
  4: 1-1 1
  5: 2-1 1
  6: 2-1 1
=====
////////// R  \\\\\\\\\\\\\\\\\\\
```

```
Atualizacao de Nodo
  0:-1 1 6
  1: 0 3 4
  2: 0 5 2
  3: 1-1 2
  4: 1-1 2
  5: 2-1 1
  6: 2-1 1
=====
////////// R  \\\\\\\\\\\\\\\\\\\
troca 2 4
ANTES
  0:-1 1 6
  1: 0 3 4
  2: 0 5 2
  3: 1-1 2
  4: 1-1 3
  5: 2-1 1
  6: 2-1 1
FIR*
  3 5 4 6
DEPOIS
  0:-1 1 6
  1: 0 3 4
  2: 0-1 3
  3: 1-1 2
  4: 1 5 2
  5: 4-1 1
  6: 4-1 1
FIR*
  3 5 2 6
-----
Atualizacao de Nodo
  0:-1 1 7
  1: 0 3 4
  2: 0-1 3
  3: 1-1 2
  4: 1 5 2
  5: 4-1 1
  6: 4-1 1
=====
////////// R  \\\\\\\\\\\\\\\\\\\
Atualizacao de Nodo
  0:-1 1 8
  1: 0 3 4
  2: 0-1 4
  3: 1-1 2
  4: 1 5 2
  5: 4-1 1
  6: 4-1 1
=====
```

 Para você fazer

- Suponha a mensagem a comprimir:
ZPSPZPP
Comprima-a usando o algoritmo de Huffman Dinâmico e informe qual o valor do QUARTO byte da saída: _____.
- Suponha a mensagem já comprimida:
39.4E.31.3A.83.80.00.
Descomprima-a usando os mesmos algoritmos e responda qual o QUARTO caractere original: _____.

Transformadas BWT e MTF

Antes de entrar no assunto, vai-se imaginar um arquivo ASCII com texto em português. Um bom exemplo é o arquivo fonte do livro de Estrutura de Dados de P. Kantek que está disponível no lugar de sempre. Em 23/12/2008, este arquivo tinha 816.820 bytes.

Comprimindo-o com PKZIP, usando a compressão normal (mais rápida) o arquivo ficou com 228.516 bytes. Uma razoável compressão para 27.97% do arquivo original.

Usando o mesmo utilitário com compressão extra forte e que demora mais obteve-se 225.343 bytes. Bem melhor.

Mas, o que dizer do resultado da execução do BZIP2 ? Este é um utilitário mais novo e ele conseguiu deixar o mesmo arquivo com o tamanho de 181.728 bytes. Ou, 22.24% do tamanho original.

Claro está, que nos 3 casos acima, o arquivo pôde ser descomprimido gerando sempre o mesmo arquivo original.

O BZIP2 é o um campeão da compressão, e ele usa 3 algoritmos em cadeia: A Transformada Burrows-Wheeler, depois a transformada Move To Front (MTF) e finalmente, o algoritmo de compressão de Huffman, nosso velho conhecido.

O método de compressão de Burrows-Wheeler foi descrito em 1994. Bastante recente, portanto. Ele é baseado em uma pesquisa (não publicada) de Wheeler em 1983. O método é composto pela combinação de três algoritmos:

Inicialmente, submete-se os dados desejados a uma função de transformação, a “Burrows-Wheeler Transform”, ou simplesmente BWT, que reordena os bytes originais, tornando-os bastante propícios para compressão.

A seguir, aplica-se a heurística “Move To Front”, (MTF). Aplicá-la logo após o BWT faz com que os dados de saída tenham muitos zeros e grande tendência para números positivos pequenos.

Finalmente submete-se os dados resultantes a algum método de compressão que atue sobre estatísticas dos dados (por exemplo, código de Huffman).

Descrição do algoritmo “Burrows-Wheeler Transform” O BWT opera em blocos de dados, e não em streams que lêem poucos dados de cada vez. Quanto maior o tamanho dos blocos, maior a taxa de compressão atingida. O algoritmo recebe um vetor A de tamanho N com os dados a serem transformados e devolve um vetor B (permutação dos elementos de A) e um índice k

Ele opera gerando todas as N rotações à direita possíveis para o vetor A, formando uma matriz N x N. A seguir, ordena estas N linhas e copia o último caractere de cada uma para o vetor B. Atribui a k o número da linha da matriz ordenada que corresponde à A.

A saída da BWT é o vetor B e a variável k.

Para facilitar a explicação, veja-se tal transformação na palavra: MISSISSIPI.

Criando a matriz BWT: que, ordenada, fica:
MISSISSIPI IMISSISSIP
ISSISSIPIM IPIMISSISS
SSISSIPIMI ISSIPIMISS
SISSIPIMIS ISSISSIPIM
ISSIPIMISS MISSISSIPI
SSIPIMISSI PIMISSISSI
SIPIMISSIS SIPIMISSIS
IPIMISSISS SISSIPIMIS
PIMISSISSI SSIPIMISSI
IMISSISSIP SSISSIPIMI

Note-se que k = 5 ou seja é a quinta linha da segunda matriz que corresponde à mensagem original.

A resposta ao algoritmo corresponde à última coluna da segunda matriz ou PSSMIISSII.

Montando o algoritmo:

- 1: BWT (A, N)
- 2: monta matriz M, rotacionando as linhas
- 3: ordena as N linhas de M
- 4: para i de 1 até N
- 5: B[i] ← M[i,N]
- 6: se M[i] = A

- 7: k ← i
- 8: fim{se}
- 9: fim{para}
- 10: devolva (B, k)

Mas como obter a sequência original A a partir de B e k? Note que a partir de B pode-se obter a primeira coluna da matriz N x N. Para isto, basta ordenar os dados de B.

Para reconstituir a mensagem original, recupera-se a saída da transformada BWT, que no caso foi PSSMIISSII. com k = 5.

Ordena-se esta saída, obtendo-se: IIIIMPSSSS. Agora, monta-se o vetor que leva a resposta → resposta ordenada, no caso, 6 7 8 5 1 2 9 10 3 4. Para recuperar o string original:

- * recupera-se k (no caso k = 5)
- * obtém-se a k-ésima letra do vetor resposta (não ordenado): (no caso ‘I’)
- * faz-se k ← INDICES[k] onde INDICES é o vetor que leva da resposta para a resposta ordenada.
- * Sucessivamente para todas as respostas.
- * Finalmente inverte-se a sequência obtida e eis a mensagem original.

Descrição da heurística “Move-To-Front”

Esta heurística consiste em montar uma fila F de caracteres, onde o mais recentemente utilizado é movido para o topo. A fila é inicializada com os elementos ordenados por seu próprio valor. Para cada caractere, armazenamos sua posição na fila naquele instante e, em seguida, movemos tal caractere para o início da mesma.

- 1: MTF (A, N)
- 2: para i de 0 até 255
- 3: INSERE-FILA (F, i)
- 4: fim{para}
- 5: para i de 1 até N
- 6: j ← 0
- 7: enquanto F[j] ≠ A[i]
- 8: j ← j + 1
- 9: fim{enquanto}
- 10: B[i] ← j
- 11: REMOVE-FILA (F, j)
- 12: INSERE-TOPO (F, i)
- 13: fim{para}
- 14: devolva (B)

A transformada MTF é um codificador de dados, tipicamente um conjunto de bytes desenhado para melhorar a performance da entropia em técnicas de compressão. Quando eficientemente implementada, ele é rápido o suficiente, de modo que seus benefícios justificam incluí-lo como etapa extra em algoritmos de compressão.

A transformada MTF Cada byte é codificado pelo seu índice em uma lista. Este índice é alterado no curso do algoritmo. A lista é inicializada no começo com a ordem usual dos bytes (0,1,2,...,255). Assim, o primeiro byte sempre é codificado com seu próprio valor. Depois, que um byte é codificado seu valor é movido para a frente da lista, antes de continuar com o próximo.

Um exemplo pode mostrar como a transformada trabalha: Imagine, ao invés de bytes, deseja-se codificar os valores 0-7. Pretende-se codificar a seguinte sequência: 1 1 0 4 3 4 5.

No que segue, considere ORIGEM DOS INDICES = 0

A lista é inicialmente (0,1,2,3,4,5,6,7). O primeiro número a codificar é o 1, que aparece no índice 1. Joga-se o 1 no stream de saída.

O 1 é movido para o início da lista, produzindo (1 0 2 3 4 5 6 7). O novo número é 1, que agora aparece na primeira posição (ou no índice 0)

A saída agora é 10.

E a lista agora é (1 0 2 3 4 5 6 7). Continuando, a

Número	Sequência	Lista
início		(0 1 2 3 4 5 6 7)
1 1 0 4 3 4 5	1	(1 0 2 3 4 5 6 7)
1 1 0 4 3 4 5	10	(1 0 2 3 4 5 6 7)
1 1 0 4 3 4 5	101	(0 1 2 3 4 5 6 7)
1 1 0 4 3 4 5	1014	(4 0 1 2 3 5 6 7)
1 1 0 4 3 4 5	10144	(3 4 0 1 2 5 6 7)
1 1 0 4 3 4 5	101441	(4 3 0 1 2 5 6 7)
1 1 0 4 3 4 5	1014415	(5 4 3 0 1 2 6 7)
Final	1014415	(5 4 3 0 1 2 6 7)

Pode-se ver como a transformada é reversível. Simplesmente, deve-se manter a mesma lista e decodificar trocando cada índice no stream codificado pelo valor deste índice. Note a diferença entre isto e o método de codificação: O índice na lista é usado diretamente ao invés de olhar na lista cada valor para índice.

Ou seja, se se começa com (0,1,2,3,4,5,6,7). Obtém-se o 1 no bloco codificado e olhando na lista quem é o segundo elemento (o de índice 1). É o 1 que vai para a saída. Move-se o 1 para a frente da lista que fica (1 0 2 3 4 5 6 7). Daí recebe-se o 0, olha-se na lista, tira-se o 1 que vai para a saída e para a frente da lista e assim por diante.

Como um exemplo, imagine-se a compressão do solilóquio de Hamlet (Ser ou não ser, eis a questão..., naturalmente em inglês: *to be or no to be that is the question...*). A entropia foi calculada em 7033 bits. Aplicando MTF diretamente a entropia vai a 7807 bits. Pior, portanto. A razão é que o inglês não tem alto nível de frequência de correlação local. Entretanto, se se aplicar antes a transformada de Burrows Wheeler e depois a transformada MTF a mensagem terá 6187 bits de entropia. Note que a BWT não modifica a entropia da mensagem, ela apenas reordena os bytes de maneira que a MTF seja mais efetiva.

Etapa final: compressão dos dados Para compressão dos dados resultantes, podemos usar código de Huffman ou algum codificador aritmético, sobre sub-blocos da sequência de dados original.

BZIP2 O bzip2 comprime arquivos utilizando os métodos descritos por Burrows-Wheeler e código de Huffman. O grau de compressão em geral é consideravelmente melhor do que a atingida pelos compressores da família “LZ”, aproximando-se da performance dos compressores estatísticos da família “PPM”, com a vantagem de ser mais rápido que estes últimos.

Seu autor, Julian Seward, declara que o bzip2 não se constitui de uma pesquisa propriamente dita, pois não apresenta idéias novas.

Este utilitário recebe um arquivo qualquer e gera um arquivo comprimido, acrescentando o sufixo .BZ2 ao nome do arquivo original. Para descomprimir chamar o utilitário com a chave “-d” (de descomprimir) quando ele gerará o arquivo original. Na dúvida, chame o utilitário com a chave “-help”.

Para você fazer

No exercício a seguir, Aplicou-se a transformada BWT a uma frase de 10 caracteres (pode eventualmente ser uma única palavra). Logo depois, aplicou-se a transformada MTF ao resultado. O universo de caracteres para efeito da MTF é

.ABCDEFGHIJKLMNOPQRSTUVWXYZ (. é espaço)
Note que não há Ç, nem acentos, nem til.

e obteve-se como resultado o vetor resultado de MTF:

1 0 5 15 3 0 22 0 4 2

com k original da saída de BWT = 5 Uma observação importante: para manter a coerência com os métodos originais, a origem dos índices no algoritmo MTF é ZERO e no BWT é UM.

Pergunta-se, qual a frase original ?



Caso prático de Huffman Suponha que você, como analista de sistemas, enfrenta o seguinte problema: *É necessário transmitir diariamente um volume de 1.100.000 registros de 76 bytes entre Curitiba e Mendoza, na Argentina.* Note-se que o volume total de dados é $1.100.000 \times 76 \approx 80\text{MB}$ bytes. Antes de prosseguir no exercício é de se notar que tais registros apresentam característica interessante: dos 76 bytes, nada menos que 56 são dígitos numéricos e os 20 restantes apresentam um texto com alta ocorrência das seguintes letras: A, E, O, P, M, S, L, R e C, com probabilidades decrescentes exatamente nessa ordem. O registro não tem nenhum tipo de compressão ou configuração, sendo portanto, texto plano.

As alternativas atuais para a tarefa incluem:

Adquirir uma ligação privativa entre Curitiba e Mendoza. Esta alternativa oferece qualidade e garantia, mas tem custos significativos bem como apresenta desperdício do recurso.

Fazer a transferência via Internet. A qualidade deixa a desejar, mas não há custos significativos. A desvantagem é que por ser pública a comunicação distribui o arquivo por inúmeros servidores ao longo do caminho, o que pode comprometer a confidencialidade.

Comprimir e criptografar o arquivo antes da remessa via Internet. Por exemplo, usando-se o utilitário PKZIP com a opção “-s”. Nesta alternativa pode-se usar uma senha padrão (aqui e lá), por exemplo a palavra **mandrake**. O arquivo tende a ser muito diminuído em tamanho e os dados, são – em princípio – ilegíveis. A crítica é que por usar sempre a mesma senha a comunicação pode ser tornar comprometida.

Comprimir e criptografar o arquivo, mas usando uma senha modificada diariamente. A dificuldade aqui é compartilhar a senha. É preciso ser criativo: talvez usar a primeira palavra com 8 ou mais caracteres, depois da décima palavra do principal editorial do jornal LA NACION diariamente publicado em Buenos Aires. Tanto o emissor quanto o receptor precisam ter acesso ao mesmo jornal, o que em princípio é fácil. O mesmo pode ser buscado em algum site da Internet que permaneça estático ao longo do tempo. A crítica aqui é a dificuldade operacional de recuperar e usar a chave de criptografia.

Queimar um CD diariamente e enviá-lo por portador de confiança. A latência é insuportável, e continua havendo riscos importantes de quebra de confidencialidade.

Usar um esquema de criptografia e compressão *ad hoc*. É o processo que vai ser descrito e usado a seguir neste exercício.

Esquema ad hoc de compressão e criptografia

Vamos usar o algoritmo de Huffman, com uma modificação importante: Como se conhece a distribuição de caracteres a enviar (grosso modo), vai-se usar uma árvore de Huffman fixa. Assim evitar-se-á a tarefa demorada de analisar o arquivo a enviar cada dia. Usando uma árvore fixa, também há a vantagem de não ter que enviá-la pela comunicação, evitando que ela possa ser devidamente capturada. Naturalmente, com certa periodicidade (1 mês ?) a árvore pode ser recriada ligeiramente modificada e enviada por meio seguro.

Entretanto, há um problema aqui: embora os caracteres 0123456789AEOPMSLRC sejam os mais frequentes, ficou claro que, em algum momento, virão outros caracteres não listados. Há várias maneiras de tratar este fato. Vejamos duas:

1. Construir a árvore privilegiando os caracteres 0123456789AEOPMSLRC mas incluindo todos os demais caracteres ASCII (já que o texto é plano). A árvore vai ficar enorme, mas ainda assim vai haver compressão, já que os caracteres 0123456789AEOPMSLRC estarão na parte alta da árvore e usarão poucos bits para serem transmitidos.

2. Construir a árvore usando os caracteres 0123456789AEOPMSLRC e mais um caracter especial, a quem chamaremos • e que, quando presente, sinalizará que os 8 bits a seguir devem ser interpretados como ASCII plano. Este caracter virá na frente de qualquer caracter que não seja 0123456789AEOPMSLRC. Não há que se temer muita indiscreção, pois tais caracteres planos (uma óbvia fraqueza do método) só estarão alinhados em fronteira de byte em 1/8 das vezes, algo como 12,5% das vezes. A vantagem é que a árvore ficará pequena, e o arquivo devidamente comprimido.

O método: a árvore Vai-se construir a árvore usando os caracteres •0123456789AEOPMSLRC nesta ordem. A exata proporção entre cada um dos caracteres, pode e vai variar. Assim, o exercício já vai dar a árvore construída, a fim de que todos obtenham o mesmo dicionário. Veja no pé da página como fica a árvore.

O caracter • está representado na árvore pela letra “a” minúscula.

Auxílio com a tabela ASCII Para quem está sem a cartela ASCII eis os códigos das letras não citadas na árvore de Huffman:

B 01000010
D 01000100
F 01000110
G 01000111
H 01001000
I 01001001
J 01001010
K 01001011
N 01001110
Q 01010001
T 01010100
U 01010101
V 01010110
W 01010111
X 01011000
Y 01011001
Z 01011010

Alguns exemplos Seja comprimir a mensagem '00000723MAIS'. O resultado é 00052FBE526C38924F0. Eis a análise: 00052 é a quantidade de bits a usar. O resultado da compressão é FBE526C38924F0.

F	B	E	5
1111	1011	1110	0101
2	6	C	3
0010	0110	1100	0011
8	9	2	4
1000	1001	0010	0100
F	0		
1111	...0000		

comprime456 'SALD00000000000200'
000523DC392235FF970
comprime456 'SALD0000000200X000'
000623DC392235F974B1C
comprime456 'ROLETE900000000000'
0005735A39B4A8DBBFF80

Para você fazer

1. Comprima a frase a seguir e responda qual o conteúdo dos 4 bytes solicitados. Note que para esta resposta, deve-se DESPREZAR as 5 posições numéricas iniciais que indicam a quantidade de bits significativos na resposta.

CREDIT000000450X110

2	9	12	12

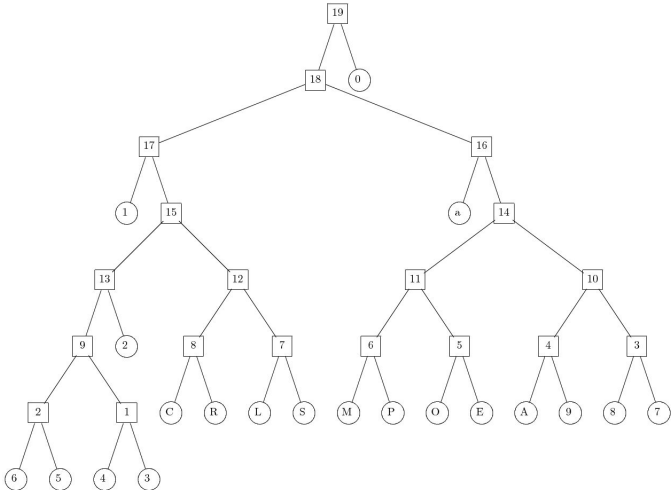
2. Descomprima a mensagem seguinte e ache os 4 caracteres da mensagem descomprimida cujos números são pedidos

0007730D6D222494A9FC9
11F8

2	9	14	20



Para padronizar a mensagem a enviar, vai-se colocar no início do string comprimido um número decimal de 5 casas, preenchido com zeros à esquerda contendo a quantidade de bits significativos que existem no string comprimido, uma vez que este vai ser apresentado em hexadecimal, com junção de 4 em 4 bits e alinhado em múltiplo de 8 bits.



Compressão de Huffman

Resultado do trabalho de doutorado de David Huffman, este método permite construir uma árvore de codificação que ao substituir caracteres do arquivo original, faz com que ele seja comprimido. O truque empregado é reservar configurações binárias menores em comprimento para os caracteres que são mais frequentes.

No exercício de hoje, o objeto da compressão vai ser uma imagem BMP mapeada. O objetivo deste exercício é mostrar como construir uma árvore de Huffman de grande porte (já que deverá haver cerca de 250 níveis distintos de cor na imagem) e também mostrar como se obtém grandes níveis de compressão ao fazê-la sobre imagens digitais.

Tal redução de tamanho deve-se ao fenômeno denominado "localidade de referência espacial" que sugere que a probabilidade de pixels vizinhos terem o mesmo nível de cor é bastante alta. Por exemplo, numa paisagem, o céu tem grande probabilidade de compartilhar níveis de cor bastante próximos (ou até iguais). Numa foto de pessoa em close-up, a cor de sua pele tende a se concentrar em poucos níveis de cor.

As etapas a desenvolver neste exercício, são as seguintes

ler a imagem Deve-se criar um programa capaz de ler a imagem, importando a matriz de pixels para dentro da memória. Lembre-se que o arquivo a ler seguirá o padrão BMP de imagens mapeadas. Neste tipo de imagem, após os 54 bytes iniciais (que caracterizam o bloco de controle do arquivo), seguem-se 1024 bytes que são a tabela de cores. (embora fossem necessários apenas $256 \times 3 = 768$ bytes, o padrão reserva 1024 por questões de eficiência). Finalmente, depois deste ponto começam os pontos da imagem, à razão de 1 byte por pixel.

A imagem lida sempre terá 512 x 512 pixels. O número de colunas em particular é múltiplo de 4, o que determina que não haverá *stuffing columns*, e simplificará o processamento.

A compressão vai se dar apenas sobre os dados da imagem. Assim o bloco de controle (54 bytes) e a tabela de cores (1024 bytes) não serão manuseados nesta folha.

O resultado final deste procedimento é uma matriz numérica de 512 linhas por 512 colunas. Cada item da matriz é um único caracter, mas aqui tratado como um número inteiro entre 0 e 255.

montar o histograma O próximo passo é contar quantas ocorrências existem de uma determinada cor.

desconsiderar zeros Devem ser ignorados eventuais níveis de cor que tenham 0 ocorrências. Esta etapa não é obrigatória, mas simplifica os cálculos a seguir.

construir a árvore Antes de estudar o algoritmo, vai-se estudar o significado das principais variáveis envolvidos.

A variável *mt* é uma matriz de *n* linhas (depende da variedade de cores da imagem) por 2 colunas. Na primeira coluna está a cor (um número entre 0 e 255) e na segunda coluna está a quantidade de pixels que têm esta cor. A variável *mt* tem limites verticais fluídos (começando em *ini* e terminando em *fim*).

A variável *arvore* é a própria. Contém 3 colunas, sendo a primeira o endereço do filho esquerdo, a segunda o endereço do filho direito e a terceira o conteúdo do nodo. Neste último, valores negativos correspondem a um nível de cor (em valor absoluto) e valores positivos correspondem a nodos intermediários (pelo seu endereço).

```

10: se mt[j;2] < men
11:   men ← mt[j;2]
12:   imen ← j
13:   fim{se}
14:   j ← j + 1
15: fim{enquanto}
16: swap(mt[imen;],mt[ini;])
17: ini ← ini + 1
18: j ← ini
19: men ← +99999
20: enquanto j ≤ fim
21:   se mt[j;2] < men
22:     men ← mt[j;2]
23:     imen ← j
24:   fim{se}
25:   j ← j + 1
26: fim{enquanto}
27: swap(mt[imen;],mt[ini;])
28: ini ← ini + 1 {ini-2 é o menor, ini-1 é o 2.menor}
29: arvore[iar;1] ← i
30: arvore[iar;2] ← mt[ini-2;1]
31: arvore[iar;3] ← mt[ini-1;1]
32: iar ← iar + 1
33: fim ← fim + 1
34: mt[fim;1] ← i
35: mt[fim;2] ← mt[ini-2;2]+mt[ini-1;2]
36: i ← i + 1
37: fim{enquanto}
    
```

📖 Para você fazer

Leia o arquivo de nome CUR 24 .BMP

Neste arquivo, monte a árvore de Huffman, conforme descrito nesta folha.

Montada a árvore, imprima-a e responda as 4 perguntas a seguir:

1. Quantos pixels tem a cor mais frequente na imagem ?

2. Quais os 3 nodos na árvore se esta for considerada uma matriz e tiver suas linhas numeradas (lembre-se que a raiz é o último nodo desta matriz). Considere as linhas de números. O valor inicial é 1.

136		
147		
237		



```

1: mt[;1] ← -mt[;1] // negativa coluna 1
2: iar ← 1
3: i ← 1
4: ini ← 1
5: fim ← número de linhas de mt
6: enquanto fim-ini > 2
7:   j ← ini
8:   men ← +99999
9:   enquanto j ≤ fim
    
```

Caso prático de Huffman: Os Lusíadas

Os Lusíadas É uma obra poética do escritor Luís Vaz de Camões, considerada a epopéia portuguesa por excelência. Provavelmente concluída em 1556, foi publicada pela primeira vez em 1572 no período literário do classicismo, três anos após o regresso do autor do Oriente.

A obra é composta de dez cantos, 1.102 estrofes e 8.816 versos que são oitavas decassílabas. A ação central é a descoberta do caminho marítimo para a Índia por Vasco da Gama, à volta da qual se vão descrevendo outros episódios da história de Portugal, glorificando o povo português.

Veja o começo

AS/ARMAS/E/OS/BAROE/ASSINALADOS
QUE/DA/OCIDENTAL/PRAIA/LUSITANA
POR/MARES/NUNCA/DE/ANTES/NAVEGADOS
PASSARAM/AINDA/ALEM/DA/TAPROBANA
EM/PERIGOS/E/GUERRAS/ESFORCADOS
MAIS/DO/QUE/PROMETIA/A/FORCA/HUMANA
E/ENTRE/GENTE/REMOTA/EDIFICARAM
NOVO/REINO/QUE/TANTO/SUBLIMARAM
E/TAMBEM/AS/MEMORIAS/GLORIOSAS
DAQUELES/REIS/QUE/FORAM/DILATANDO
A/FE/O/IMPERIO/E/AS/TERRAS/VICIOSAS
DE/AFRICA/E/DE/ASIA/ANDARAM/DEVASTANDO
E/AQUELES/QUE/POR/OBRAS/VALEROSAS
SE/VAO/DA/LEI/DA/MORTE/LIBERTANDO
CANTANDO/ESPALHAREI/POR/TODA/PARTE
SE/A/TANTO/ME/AJUDAR/O/ENGENHO/E/ARTE
...

Neste exercício, você vai receber um arquivo contendo duas coisas

árvore A árvore de Huffman usada para comprimir um texto e

texto comprimido um texto formado por 3 versos de Os Lusíadas

Você deve primeiro construir a árvore, depois decodificar em binário o texto e finalmente recuperar o texto plano que foi originalmente comprimido.

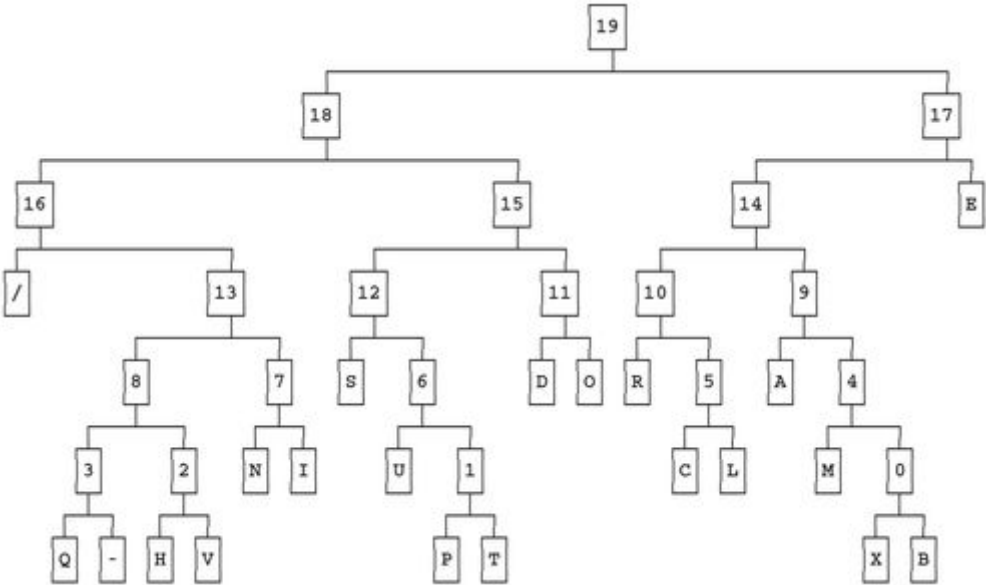
Para isso, deverá usar o programa DUMP.PY que imprime o conteúdo hexadecimal de arquivos. Ele só funciona no Python 3.x. O programa está disponível no lugar usual. O arquivo que vai ser lido contém a árvore no formato de lista de python nos primeiros 512 bytes do arquivo. No byte 513 (em hexadecimal x'200') existem 4 bytes contendo em binário o comprimento em bits do string comprimido. Este começa no byte 517 (hexadecimal x'204') e prossegue até o final do arquivo sendo seguido de eventuais **stuffing bits** necessários para completar o arquivo em limite de byte. O arquivo termina por x'0D' x'0A' que devem ser ignorados. Outra possibilidade é olhando o arquivo através de um utilitário binário (por exemplo wxHexEditor ou então o site <http://www.onlinehexeditor.com/>) e investigando os bytes em questão.

Exemplo

Seja o seguinte exemplo:

[[0,-88,-66], [1,-80,-84], [2,-72,-86], [3,-81,-45],
[4,-77,0], [5,-67,-76], [6,-85,1], [7,-78,-73],
[8,3,2], [9,-65,4], [10,-82,5], [11,-68,-79],
[12,-83,6], [13,8,7], [14,10,9], [15,12,11],
[16,-47,13], [17,14,-69], [18,16,15], [19,18,17]]

cujo desenho é



Dicionário Que gera o seguinte dicionário

/=000	D=0110
Q=001000	O=0111
=001001	R=1000
H=001010	C=10010
V=001011	L=10011
N=00110	A=1010
I=00111	M=10110
S=0100	X=101110
U=01010	B=101111
P=010110	E=11
T=010111	

O string comprimido é

010011000010011101100101101000110001110110000
0010111111000010001110000010100101010110001111
001101101100010010111001111101111100010100110
011100100111000001000010101010001100110011100
001101101001001011000011100001101100111101110
101000001101110001000101001101100111100001110
010010010000101011000010011000110100010110101
010011001010110001100001001100010010101000110
010111110000011001110000101000110001110010111
1100001000111

Que devidamente descomprimido deu origem aos seguintes versos

SE/SEMPRE/EM/VERSO/HUMILDE/CELEBRADO
E/QUANDO/DESCE/O/DEIXA/DERRADEIRO
QUE/SE/ESPALHE/E/SE/CANTE/NO/UNIVERSO

Na árvore, os valores **negativos** indicam a ocorrência de uma letra. O valor foi retirado de qualquer tabela ASCII. Assim, -65 equivale a “A” , -66 a “B” e assim por diante até o “Z” que vale -90. Note que, no arquivo, o caracter hífen (-) separa os 3 versos.

Para você fazer

Procure no lugar de sempre seu arquivo que é o

F458N001.myd

Responda aqui os 3 versos comprimidos no seu arquivo

verso 1
verso 2
verso 3

HEAP

Primeiro, esta estrutura não tem nada a ver com o gerenciador de memória heap do compilador Pascal (e outros compiladores). Só o nome é o mesmo, já que heap em inglês é monte, pilha, montanha...

Segundo, a estrutura heap tem como funcionalidade principal apontar muito rapidamente, qual é o maior elemento de um vetor. É óbvio, que se se ordenar um vetor, será fácil ver quem é o maior elemento. Só que a ordenação mais "barata" que existe (o quicksort) é muito mais cara (demorada) que construir um heap, que é muito mais barato.

Este fato é utilizado na ordenação HEAPSORT, que é um dos mais eficientes algoritmos de ordenação. Durante a ordenação um heap é sucessivamente construído para escolher – a cada turno – quem é o campeão (o maior valor) do vetor.

Apenas para comparação, suponha um vetor de 2500 números aleatórios. Ordenando-o usando os diversos algoritmos obtiveram-se as seguintes demoras: Bolha: 429 segundos. Inserção: 198 segundos. Shellsort: 8 segundos. Heapsort: 6 segundos e o grande campeão, o Quicksort com 3 segundos.

Um heap é uma lista linear composta de elementos com chaves $s_1, s_2, \dots, s_n$, satisfazendo a seguinte propriedade:

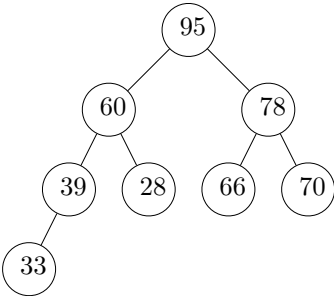
$$s_i \leq s_{\lfloor i/2 \rfloor} \forall i \leq n$$

Em outras palavras: se imaginarmos uma árvore completa com os elementos do vetor, então o pai tem que ser maior do que os seus filhos...

Esta propriedade pode ser mais facilmente entendida se escrevermos seqüencialmente os elementos da lista em uma árvore binária na qual todos os nodos (com possível exceção do último) estão completos. Exemplificando e considerando uma lista de prioridade onde o identificador do elemento já é a sua prioridade, (quanto maior o número, maior a prioridade), tem-se:

95, 60, 78, 39, 28, 66, 70, 33

Escrevendo-se estes números em uma árvore ordenada, tem-se:



Agora fica mais fácil perceber a propriedade acima, que pode ser traduzida como "a prioridade do pai é sempre maior do que de seus filhos", utilizada recursivamente por toda a árvore.

Entretanto, não se esqueça: um heap é um vetor e não uma árvore.

O heap é usado num ótimo algoritmo de ordenação (o heap sort), em listas de prioridade e sempre que uma busca rápida pelos maiores (ou menores) de um conjunto.

A propriedade básica do heap garante que se um vetor é heap, o maior (ou menor) elemento ocupa a primeira posição.

Finalmente, duas outras regras precisam ser conhecidas: a inclusão em um heap sempre se dá AO FINAL do vetor. A exclusão sempre é do primeiro elemento (que, lembrando, é o único elemento do qual se pode afirmar algo).

Algoritmos

```
1: funcao CRIAHEAP (vetor V) : vetor
2: I ← ⌊ tamanho(V) ÷ 2
```

```
3: enquanto I ≥ 1
4:   retorne DESCE (V,I)
5:   I ← I - 1
6: fim{enquanto}
7: fimfuncao
```

Note que este algoritmo toma um vetor V desordenado e com ele constrói um heap. A primeira metade de V é conduzida – elemento a elemento – até o seu local definitivo.

A construção poderia usar a função SOBE (abaixo descrita), mas o esforço seria dobrado, já que neste caso TODOS os elementos precisam ser erguidos até seu local definitivo.

```
1: funcao DESCE (vetor V, indice I) : vetor
2: J ← 2 × I
3: se J ≤ tamanho(V)
4:   se J ≠ tamanho(V)
5:     se V[J+1] > V[J]
6:       J ← J + 1
7:   fim{se}
8:   fim{se}
9:   se V[I] < V[J]
10:    X ← V[I]
11:    V[I] ← V[J]
12:    V[J] ← X
13:   retorne DESCE(V,J)
14:   fim{se}
15: fim{se}
16: fimfuncao
```

Esta função toma um elemento e vai descendo-o até seu local definitivo. Note que a comparação que determina a movimentação do elemento é feita sempre com o maior "filho". Veja também que a função é recursiva, e que esta recursão só acaba quando o elemento acha seu lugar definitivo.

```
1: funcao SOBE (vetor V, indice I) : vetor
2: J ← ⌊ I ÷ 2
3: se J ≥ 1
4:   se V[I] > V[J]
5:     X ← V[I]
6:     V[I] ← V[J]
7:     V[J] ← X
8:   retorne SOBE(V,J)
9:   fim{se}
10: fim{se}
11: fimfuncao
```

Exercício 1

Considere a seguinte seqüência de 10 números

51 19 26 63 86 8 65 34 2 53

Construindo o heap "R"seqüencialmente a partir do primeiro, qual seria o valor de R ?

1(51)
2-51(19)
3-51 19(26)
4(63)51 26 19
5(86)63 26 19 51
6-86 63 26 19 51 (8)
7-86 63(65)19 51 8 26
8-86 63 65(34)51 8 26 19
9-86 63 65 34 51 8 26 19 (2)
10-86 63 65 34(53) 8 26 19 2 51

Com isto, quanto e?

R[7]	R[1]	R[5]	R[8]
26	86	53	19

Exercício 2

Um vetor H de 30 números é dado. Pergunta-se: ele é heap ? Se for, responda zero e se não for, indique qual o número de H que impede que ele seja. Escreva o valor do primeiro pai que tem filhos maiores.

97 94 73 92 90 68 42 81 30 53
89 45 57 39 24 27 56 19 15 7
1 76 46 23 36 28 55 18 6 22
A resposta aqui é 0 (significando que é HEAP)

Exercício 3

Dado o heap de 30 números, através do vetor V

99 82 93 65 80 75 63 39 64 73
77 69 33 49 52 5 3 42 14 72
44 46 48 28 62 32 9 40 47 26

Deve-se incluir o novo elemento 21. Após qual valor de V ele entra? DEPOIS DO 26

Exercício 4

Dado um vetor K, representando um heap de 30 números, exclui-se o primeiro elemento de K. Trazendo o último para a frente e reposicionando-o depois, este elemento ficará depois de qual elemento de K ? (pergunta-se o conteúdo e não a posição).

99 89 91 81 79 82 85 50 69 49
55 45 36 77 76 40 22 46 26 31
41 35 19 43 6 12 14 33 56 17

DEPOIS DO 33.

Para você fazer

Para os "heaps" aqui construídos, considere:

- Todas as numerações começam em 1 e
- o pai sempre tem mais prioridade (é maior) do que os filhos

I Considere a seguinte seqüência de 10 números

43 84 34 9 50 60 39 49 88 69

Construindo o heap "R"seqüencialmente a partir do primeiro, qual seria o valor de

1.R[8], 2.R[9]

II Um vetor H de 30 números é dado. Pergunta-se: ele é heap ? Se for, responda zero e se não for, indique qual o número de H que impede que ele seja. Escreva o valor do primeiro pai que tem filhos maiores

99 93 91 90 79 87 69 71 57 42
74 21 77 63 24 61 68 51 4 7
18 39 43 6 19 67 73 52 9 12

III Dado o heap de 30 números, através do vetor V:

100 93 98 91 86 94 97 74 77 14
63 62 75 49 54 64 8 72 7 9
2 61 28 25 20 15 60 31 45 53

Deve-se incluir o novo elemento 40 . Após qual valor de V ele entra?

IV Dado um vetor K, representando um heap de 30 números, exclui-se o primeiro elemento de K. Trazendo o último para a frente (vide algo- ritmo no livro texto) e reposicionando-o depois, este elemento ficará depois de qual elemento de K ? (pergunta-se o conteúdo e não a posição)

99 82 96 69 79 94 93 39 66 71
30 74 83 67 92 7 8 58 18 47
20 28 14 59 50 65 13 16 48 68

I ₁	I ₂	II	III	IV



Como trocar informações ?

A informática convive – desde seu início – com uma dificuldade e tanto: como padronizar a utilização de códigos binários. O objetivo é nobre: permitir a troca de objetos binários e a sua correta interpretação por parte de todos os parceiros em uma comunicação qualquer. A necessidade é nova, a informática só apareceu na face da terra na década de 50 do século passado. No início, cada fabricante usava seus códigos. A IBM, por exemplo, construiu toda a sua família de equipamentos usando o código EBCDIC. Isto evoluiu para o código ASCII (início de 1960, comite ANSI-X3.2) que acabou sendo adotado como padrão. Ele privilegiou o idioma inglês, e padronizou seus caracteres nas primeiras 128 posições do código ASCII. Aos demais idiomas do planeta ficou disponível a segunda parte do código ASCII (mais 128 posições). É aqui que o código BRASCII (NBR-9614:1986 e NBR-9611:1991 da ABNT) colocou os caracteres acentuados do português. Assim como o Brasil, muitos outros países fizeram o mesmo. Isto apresentou pelo menos 2 problemas:

- Um arquivo composto no Brasil, quando impresso numa impressora fabricada na França, aparecia como uma série de caracteres malucos.
- Idiomas com muitos caracteres (katakana, hiragana, hangul, tagalog,...) simplesmente não tinham como ser representados no código ASCII.

Unicode

Em meados da década de 80, a academia e a indústria começaram a discutir propostas de resolver os 2 problemas acima. Diversos padrões foram sugeridos e eles começaram a ser usados, na base do “cada um por si”. Uma primeira proposta sugeria usar 16 bits (2 bytes) o que permitiria $2^{16} = 65536$ caracteres. Parecia muito, mas vale citar Peter Norton, que disse há mais de 30 anos: *na informática, não importa quanto você tem. Não é o suficiente.*

Em 1991, criou-se o Consórcio Unicode e em outubro de 1991 o primeiro volume do padrão foi publicado. Em 1996, introduziu-se um mecanismo de codificação e com ele, a barreira dos 2 bytes foi ultrapassada. Agora, o espaço de endereçamento do Unicode ultrapassou o milhão de caracteres, o que permitiu incluir idiomas históricos (hieróglifos egípcios) ou mesmo idiomas muito pouco usados (Linear B por exemplo). O padrão atual consagra 1.114.112 códigos de ponto (*codepoints*), correspondendo ao intervalo hexadecimal de 00.00.00 a 10.FF.FF. Note que nem todo codepoint indica um caracter, já que alguns deles são reservados a outras finalidades.

Há alguns codepoints privados, que não têm significação no Unicode. Eles ficam para uma negociação entre remetente e destinatário e podem ser usados para qualquer coisa (são eles: Área privada: U+E000..U+F8FF com 6,400 caracteres; Área suplementar A: U+F0000..U+FFFFD com 65,534 caracteres e Área suplementar B: U+100000..U+10FFFF com 65,534 caracteres).

O Consórcio Unicode trabalha junto com a ISO (International Organization for Standardization) já que o padrão ISO/IEC 10646 é o próprio código Unicode. Um conceito importante é o de script. Trata-se de um conjunto de letras, sinais e regras de escrita que agregam partes do Unicode e que são usados em conjunto de *writing systems*. Na versão 7 do Unicode, são 123 os scripts, e há uma lista de candidatos a serem incluídos.

UTF=Unicode Transformation Formats

Diversos mecanismos foram sugeridos para implementar o código Unicode, com a preocupação sempre presente de compatibilidade reversa. Foram propostos os UTF-1 (nunca chegou a ser muito usado, pois tinha problemas de performance), UTF-16 (usava 1 ou 2 palavras de 16 bits) , UTF-32 (cada caracter usava exatos 32 bits), mas todas elas acabaram migrando para a UTF-8 que se firmou como padrão de fato, principalmente porque muito cedo foi adotado como padrão pelo consórcio IMC (Internet Mail Consortium) e pelo W3C. O Google reporta que desde 2008, a maioria dos conteúdos HTML trocados na web usa a codificação UTF-8. O esquema todo foi escrito durante um jantar em 2 de setembro de 1992 por Ken Thompson e Rob Pike. Suas grandes qualidades são:

- Compatibilidade reversa, já que caracteres da primeira parte do ASCII não sofrem modificação (1º bit=0). Desta maneira qualquer arquivo ASCII que não use a segunda parte já é um arquivo UTF-8.
- Clara distinção entre caracteres de byte único (1º bit=0) e caracteres multi-bytes. Estes contêm um primeiro byte heading e um ou mais bytes de continuação. O heading tem 2 ou mais bits 1 seguidos por um zero enquanto os bytes de continuação sempre começam por 10.
- Auto-sincronização. Bytes únicos, header e continuação não compartilham valores, indicando que o início do caracteres está a no máximo 3 bytes para trás.
- Indicação do comprimento: O número de uns na alta ordem do byte heading indica a quantidade de bytes do caractere, sem ser necessário examinar os seus bytes.

Veja-se um resumo do exposto

bits	início	fim	tamanho	byte1	byte2	byte3	byte4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Veja-se agora alguns exemplos de caracteres unicode de 1, 2, 3 e 4 bytes

Character	Binary code point		
\$ U+0024		0100100	
¢ U+00A2	000	10100010	
€ U+20AC	00100000	10101100	
報 U+24B62	00010	01001011	01100010

Binary UTF-8		Hexadecimal UTF-8
	00100100	24
	11000010 10100010	C2 A2
	11100010 10000010 10101100	E2 82 AC
	11110000 10100100 10101101 10100010	F0 A4 AD A2

Uma tabela de conversão binário hexadecimal para ajudar:

hex	bin	hex	bin	hex	bin	hex	bin
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Exemplos

1. Seja o seguinte conjunto de 5 caracteres UNICODE codificados em UTF-8.

F2B1B88DF4BAA8A7E192A9EF92A5F4A584BF

Seus codepoints em hexadecimal

0B1E0D 13AA27 14A9 F4A5 12513F

E em decimal

728589 1288743 5289 62629 1200447

2. Seja o seguinte conjunto de 5 caracteres UNICODE codificados em UTF-8.

39F6B2B6A503E68FAFF59A84B1

Seus codepoints em hexadecimal

39 1B2DA5 03 63EF 15A131

E em decimal

57 1781157 3 25583 1417521

☞ Para você fazer

A seguir um conjunto de 5 caracteres UNICODE codificados em UTF-8. Você deve examiná-los e descobrir quais os 5 codepoints primeiro escritos em hexadecimal e depois em decimal (é só uma simples conversão).

4BF2A192BF3CF5B4BE956E

1 em hexa	2 em hexa	3 em hexa	4 em hexa	5 em hexa
1 em dec	2 em dec	3 em dec	4 em dec	5 em dec



Algoritmos de ordenação - 1.parte

Os algoritmos de ordenação, grosso modo, podem ser divididos em 2 categorias: aqueles de complexidade $O(n^2)$ e os de complexidade $O(n \times \log_2 n)$, ambas as expressões – como sempre – a menos das constantes multiplicativas.

O estudo da primeira família se justifica pelas seguintes razões:

- Constantes multiplicativas eventualmente menores
- Abundância de recursos de hardware
- Irrelevância do desempenho para pequenas instâncias
- Simplicidade (inclusive conceitual) dos algoritmos envolvidos

Este exercício tratará de 3 algoritmos, além de uma pequena derivação de um deles, todos com desempenho quadrático. São eles:

Bolha Trata-se do algoritmo mais simples que existe. Pares de vizinhos são comparados e se eles estiverem desordenados, são invertidos. A cada passada, o elemento mais leve é levado para a ponta. Daí o nome.

Bolha* O mesmo algoritmo acima, mas com um pequeno truque para encerrar o algoritmo mais cedo caso o vetor a ordenar seja do tipo “quasi-ordenado”.

Inserção O mesmo algoritmo usado pelo jogador de buraco ao ordenar as cartas que recebe. Há um monte de cartas desordenadas. As cartas, uma a uma, vão sendo retiradas desse monte e colocadas em ordem na mão do freguês. Quando o monte terminar, a mão está ordenada.

Seleção O menor elemento do vetor a ordenar é intercambiado com o primeiro elemento do vetor. Depois, o segundo menor com o segundo elemento, e assim por diante.

Algoritmo Bolha

```
1: inteiro I,J,AUX
2:  $I \leftarrow$  tamanho do vetor a ordenar
3: enquanto ( $I \geq 2$ ) faça
4:    $J \leftarrow 2$ 
5:   enquanto ( $J \leq I$ ) faça
6:     se  $V[J-1] > V[J]$  então
7:        $AUX \leftarrow V[J-1]$ 
8:        $V[J-1] \leftarrow V[J]$ 
9:        $V[J] \leftarrow AUX$ 
10:    fim se
11:     $J++$ 
12:  fim enquanto
13:   $I--$ 
14: fim enquanto(***)
```

Note as duas características que este algoritmo tem: uma ineficiência grande aliada à simplicidade do código. A grande maioria dos programadores profissionais, só conhece “de cabeça”este algoritmo.

Algoritmo Bolha-Estrela

```
1: inteiro I,J,AUX
2: lógico MUDOU  $\leftarrow$  .V.
3:  $I \leftarrow$  tamanho do vetor a ordenar
4: enquanto ( $I \geq 2$ )  $\wedge$  MUDOU faça
5:    $J \leftarrow 2$ 
6:   MUDOU  $\leftarrow$  .F.
7:   enquanto ( $J \leq I$ ) faça
8:     se  $V[J-1] > V[J]$  então
9:        $AUX \leftarrow V[J-1]$ 
10:       $V[J-1] \leftarrow V[J]$ 
11:       $V[J] \leftarrow AUX$ 
12:      MUDOU  $\leftarrow$  .V.
13:    fim se
14:     $J++$ 
15:  fim enquanto
16:   $I--$ 
17: fim enquanto(***)
```

A única diferença em relação ao original é a presença da variável MUDOU que determina o fim do loop externo, quando nenhuma mudança mais for feita.

Algoritmo de Inserção

```
1: inteiro I, J, X
2:  $I \leftarrow 2$ 
3:  $TAM \leftarrow$  Tamanho do vetor a ordenar
4: enquanto ( $I \leq TAM$ ) faça
5:    $X \leftarrow V[I]$ 
6:    $J \leftarrow I-1$ 
7:   enquanto ( $J \geq 1$ )  $\wedge$  ( $V[J] > X$ ) faça
8:      $V[J+1] \leftarrow V[J]$ 
9:      $J--$ 
10:  fim enquanto
11:   $V[J+1] \leftarrow X$ 
```

```
12:    $I++$ 
13: fim enquanto(***)
```

Na linha 5, o elemento que vai ser colocado em seu lugar definitivo é sacado do vetor. Com isso ele abre um espaço para remanejamento dentro do vetor. Isso é feito enquanto $V[J] > X$ nas linhas 8 e 9. Se o sujeito que foi sacado é o menor do vetor, a condição do enquanto é encerrada pela outra condição ($J \geq 1$). Depois que o remanejamento foi feito, o elemento sacado volta para o vetor, agora no seu lugar certo (linha 11).

Algoritmo Seleção

```
1: inteiro I,J,COR,INUI,AUX
2:  $TAM \leftarrow$  Tamanho do vetor a ordenar
3:  $I \leftarrow 1$ 
4: enquanto ( $I \leq TAM$ ) faça
5:    $COR \leftarrow V[I]$ 
6:    $INUI \leftarrow I$ 
7:    $J \leftarrow I+1$ 
8:   enquanto ( $J \leq TAM$ ) faça
9:     se  $V[J] < COR$  então
10:       $COR \leftarrow V[J]$ 
11:       $INUI \leftarrow J$ 
12:   fim se
13:    $J++$ 
14: fim enquanto
15:   $AUX \leftarrow V[I]$ 
16:   $V[I] \leftarrow V[INUI]$ 
17:   $V[INUI] \leftarrow AUX$ 
18:   $I++$ 
19: fim enquanto(***)
```

O menor elemento é procurado (linhas 8 a 14). Encontrado (agora ele é apontando por INUI, ele é trocado de lugar com o elemento apontado por I (linhas 15 a 17).

Desempenho

Compare os desempenhos dos 4 algoritmos estudados: ¹

Tamanho e tipo	Bolha	Bolha*	Inserção	Seleção
100 aleatórios	0.6 seg	0.7 seg	0.3 seg	0.4 seg
100 quasi-ordenados	0.4 seg	0.06 seg	0.01 seg	0.4 seg
500 aleatórios	16 seg	17 seg	7 seg	10 seg
500 quasi-ordenados	11 seg	0.1 seg	0.1 seg	10 seg
1000 aleatórios	68 seg	70 seg	31 seg	41 seg
1000 quasi-ordenados	47 seg	0.3 seg	0.2 seg	40 seg
2500 aleatórios	429 seg	436 seg	198 seg	255 seg
2500 quasi-ordenados	294 seg	0.6 seg	0.5 seg	255 seg

Exemplo

Para fazer este exercício você deve fazer um mini-chinês em cada um dos 4 algoritmos. Cada um deles receberá um vetor diferente de 10 inteiros. Cada vez que o seu chinês passar pelo lugar marcado com (***) no algoritmo, você deve imprimir o conteúdo de V. Na segunda vez em que passar lá, responda o elemento de V que é pedido. Acompanhe nos exemplos.

Se o algoritmo BOLHA for executado com o vetor

37 63 30 45 6 9 50 58 40 80

Ao passar a segunda vez pelo lugar marcado, V[6]=50.

Se o algoritmo BOLHA* for executado com o vetor

2 60 87 28 55 53 33 48 62 73

Ao passar a segunda vez por (***), V[8]=62

Se o algoritmo INSERÇÃO for executado com o vetor

30 10 72 29 42 78 53 47 86 92

Ao passar a segunda vez por (***), V[2]=30

Se o algoritmo SELEÇÃO for executado com

22 53 77 20 37 64 6 31 44 88

Ao passar a segunda vez por (***), V[4]=53

Para você fazer

Algoritmo BOLHA

V= 85 31 18 65 55 43 20 46 38 26 . Qual será V[5]=_____ ?

Algoritmo BOLHA*

V= 38 18 40 56 69 68 32 29 11 92 . Qual será V[9]=_____ ?

Algoritmo INSERÇÃO

V= 10 25 41 66 60 23 40 85 45 19 . Qual será V[1]=_____ ?

Algoritmo SELEÇÃO

V= 30 72 37 77 27 83 74 69 12 29 . Qual será V[1]=_____ ?

1

¹Medidas feitas com um computador Pentium 500MHz, com Windows Millenium usando a linguagem APL plus, versão 6.0

Algoritmos de ordenação

- Na década de 70, James Martin sugeriu que 40% do tempo de TODOS os computadores é gasto neste algoritmo.
- O conceito de ordem é fundamental em programação
- Excelente tópico para estudar algoritmos e estudar comportamentos e complexidades

Antes de começarmos a estudar os diversos algoritmos de classificação, vamos definir o problema:

Seja X um conjunto composto por i elementos entre os quais se pode estabelecer uma relação de ordem. Dados X_i e X_j com $i \neq j$, sempre pode-se estabelecer $X_i > X_j$ ou $X_i = X_j$ ou $X_i < X_j$.

- X estará em ordem crescente se e somente se $X_i \leq X_j, \forall i < j$.
- X estará em ordem estritamente crescente se e somente se $X_i < X_j, \forall i < j$.
- X estará em ordem decrescente se e somente se $X_i \geq X_j, \forall i < j$.
- X estará em ordem estritamente decrescente se e somente se $X_i > X_j, \forall i < j$.

Um algoritmo de ORDENAÇÃO é aquele que recebe um X qualquer (possivelmente desordenado), permuta seus elementos e devolve X em ordem.

Tipicamente fazem parte de X_i um conjunto de informações. Neste caso, haverá uma parte de X_i denominada CHAVE é identificada por k ($k=key$) pela qual se fará a ordenação. Embora devamos ter em mente a existência dos outros campos, apenas a chave será tratada nos algoritmos.

Os algoritmos de ordenação, grosso modo, podem ser divididos em 2 categorias: aqueles de complexidade $O(n^2)$ e os de complexidade $O(n \log_2 n)$, ambas as expressões – como sempre – a menos das constantes multiplicativas.

O estudo da primeira família se justifica pelas seguintes razões:

- Constantes multiplicativas eventualmente menores
- Abundância de recursos de hardware
- Irrelevância do desempenho para pequenas instâncias
- Simplicidade (inclusive conceitual) dos algoritmos envolvidos

Principais algoritmos

Bolha Trata-se do algoritmo mais simples que existe. Pares de vizinhos são comparados e se eles estiverem desordenados, são invertidos. A cada passada, o elemento mais leve é levado para a ponta. Daí o nome.

Bolha* O mesmo algoritmo acima,

mas com um pequeno truque para encerrar o algoritmo mais cedo caso o vetor a ordenar seja do tipo “quasi-ordenado”.

Inserção O mesmo algoritmo usado pelo jogador de buraco ao ordenar as cartas que recebe. Há um monte de cartas desordenadas. As cartas, uma a uma, vão sendo retiradas desse monte e colocadas em ordem na mão do freguês. Quando o monte terminar, a mão está ordenada.

Seleção O menor elemento do vetor a ordenar é intercambiado com o primeiro elemento do vetor. Depois, o segundo menor com o segundo elemento, e assim por diante.

Shell Tem este nome devido ao seu descobridor, o dr Shell em 1959. Parecido com o BOLHA no sentido de que ele troca vizinhos, mas não próximos e sim distantes. Adicionalmente aos demais algoritmos, este precisa uma sequência de controle, como operador adicional. Pode ter quantos números se quiser, desde que o último seja sempre 1.

HeapSort Usa como estrutura auxiliar um heap. Tem desempenho excelente, e se o programador souber bem o que é um heap, sua implementação é fácil e o seu conceito mais ainda.

Quick Devido a C.R.Hoare em 1962, este algoritmo é o campeão. Ninguém até hoje conseguiu ser mais rápido que ele. Sua estratégia passa pela escolha de pivots, que dividem o conjunto sendo classificado em 2 partes. Uma contém os elementos menores que o pivot e a outra os maiores. Levando esse procedimento ao limite, o conjunto estará classificado. Este é um excelente exemplo da estratégia “dividir e conquistar”.

Algoritmo Bolha

```
1: inteiro I,J,AUX
2: I ← tamanho do vetor a ordenar
3: enquanto (I ≥ 2)
4:   J ← 2
5:   enquanto (J ≤ I)
6:     se V[J - 1] > V[J]
7:       AUX ← V[J - 1]
8:       V[J - 1] ← V[J]
9:       V[J] ← AUX
10:    fim{se}
11:    J ++
12:    fim{enquanto}
13:    I --
14: fim{enquanto}
```

Note as duas características que este algoritmo tem: uma ineficiência grande aliada à simplicidade do código. A grande maioria dos programadores profissionais, só conhece “de cabeça” este algoritmo.

Algoritmo Bolha-Estrela

```
1: inteiro I,J,AUX
2: lógico MUDOU ← .V.
3: I ← tamanho do vetor a ordenar
4: enquanto (I ≥ 2) ∧ MUDOU
5:   J ← 2
6:   MUDOU ← .F.
7:   enquanto (J ≤ I)
8:     se V[J - 1] > V[J]
9:       AUX ← V[J - 1]
10:      V[J - 1] ← V[J]
11:      V[J] ← AUX
12:      MUDOU ← .V.
13:    fim{se}
14:    J ++
15:    fim{enquanto}
16:    I --
17: fim{enquanto}
```

A única diferença em relação ao original é a presença da variável MUDOU que determina o fim do loop externo, quando nenhuma mudança mais for

feita.

Algoritmo de Inserção

```
1: inteiro I, J, X
2: I ← 2
3: TAM ← Tamanho do vetor a ordenar
4: enquanto (I ≤ TAM)
5:   X ← V[I]
6:   J ← I - 1
7:   enquanto (J ≥ 1) ∧ (V[J] > X)
8:     V[J + 1] ← V[J]
9:     J --
10:    fim{enquanto}
11:    V[J + 1] ← X
12:    I ++
13: fim{enquanto}
```

Na linha 5, o elemento que vai ser colocado em seu lugar definitivo é sacado do vetor. Com isso ele abre um espaço para remanejamento dentro do vetor. Isso é feito enquanto $V[J] > X$ nas linhas 8 e 9. Se o sujeito que foi sacado é o menor do vetor, a condição do enquanto é encerrada pela outra condição ($J \geq 1$). Depois que o remanejamento foi feito, o elemento sacado volta para o vetor, agora no seu lugar certo (linha 11).

Algoritmo Seleção

```
1: inteiro I,J,COR,INUI,AUX
2: TAM ← Tamanho do vetor a ordenar
3: I ← 1
4: enquanto (I ≤ TAM)
5:   COR ← V[I]
6:   INUI ← I
7:   J ← I + 1
8:   enquanto (J ≤ TAM)
9:     se V[J] < COR
10:      COR ← V[J]
11:      INUI ← J
12:   fim{se}
13:   J ++
14:   fim{enquanto}
15:   AUX ← V[I]
16:   V[I] ← V[INUI]
17:   V[INUI] ← AUX
18:   I ++
19: fim{enquanto}
```

O menor elemento é procurado (linhas 8 a 14). Encontrado (agora ele é apontando por INUI, ele é trocado de lugar com o elemento apontado por I (linhas 15 a 17).

Algoritmo Shell

```
1: inteiro k, h, i, x, j
2: TAM ← Tamanho do vetor a ordenar
3: DECR ← {sequência de controle desejada}
4: para K = 1 até tamanho(DECR)
5:   H ← DECR[K]
6:   I ← H
7:   enquanto (I < TAM)
8:     X ← V[I + 1]
9:     J ← I
10:    enquanto (J ≥ H) ∧ (V[1 + J - H] > X)
11:      V[1 + J] ← V[1 + J - H]
12:      J ← J - H
13:    fim{enquanto}
14:    V[1 + J] ← X
15:    I ++
16:    fim{enquanto}
17: fim{para}
```

Algoritmo Heap Sort

```
1: inteiro I,J
2: TAM ← Tamanho do vetor a ordenar
3: I ← ⌊(TAM ÷ 2)⌋
4: para J = I até 1, passo = -1
5:   desce J,TAM
```

```
6: fim{para}
7: para J = TAM até 1, passo = -1
8:   V[J] ← exclui J
9: fim{para}
10: devolva V

1: inteiro função exclui (int J)
2: devolva V[1]
3: V[1] ← V[J]
4: desce (1, J)
1: função desce (int K, TAM)
2: inteiro J
3: J ← K × 2
4: se (J ≤ TAM)
5:   se (J < TAM)
6:     se (V[J + 1] > V[J])
7:       J ++
8:     fim{se}
9:     fim{se}
10:    se (V[K] < V[J])
11:      AUX ← V[K]
12:      V[K] ← V[J]
13:      V[J] ← AUX
14:    desce (J, TAM)
15:    fim{se}
16: fim{se}
```

Algoritmo QUICK

```
1: função RAP(inteiro I, F)
2: inteiro ANT,DEP,MEI,AUX
3: ANT ← I
4: DEP ← F
5: MEI ← V[⌊(I + F) ÷ 2⌋]
6: repita
7:   enquanto (V[ANT] < MEI)
8:     ANT ++
9:   fim{enquanto}
10:  enquanto (V[DEP] > MEI)
11:    DEP --
12:  fim{enquanto}
13:  se (ANT ≤ DEP)
14:    AUX ← V[ANT]
15:    V[ANT] ← V[DEP]
16:    V[DEP] ← AUX
17:    ANT ++
18:    DEP --
19:  fim{se}
20: até (ANT > DEP)
21: se (I < DEP)
22:   RAP (I, DEP)
23: fim{se}
24: se (ANT < F)
25:   RAP (ANT, F)
26: fim{se}
27: fim função
28: TAM ← Tamanho do vetor a ordenar
29: RAP (1,TAM)
```

Desempenho Compare os desempenhos dos algoritmos estudados:

Vetor aleatório				
alg	100	500	1K	2.5K
bol	0.6	16	68	429
bol*	0.7	17	70	436
ins	0.3	7	31	198
sel	0.4	10	41	255
shell	0.1	1	2	8
heap	0.1	1	2	6
quick	0.06	0.5	1	3

Vetor quasi-ordenado				
alg	100	500	1K	2.5K
bol	0.4	11	47	294
bol*	0.06	0.1	0.3	0.6
ins	0.01	0.1	0.2	0.5
sel	0.4	10	40	255
shell	0.1	0.5	1	3
heap	0.1	1	2	6
quick	0.06	0.3	0.8	2

Para você fazer

Estude o que acontece quando os dados a ordenar não cabem todos ao mesmo tempo na memória do computador.

¹Medidas feitas com um computador Pentium 500MHz, com Windows Millenium usando a linguagem APL plus, versão 6.0

Algoritmos de ordenação - 2.parte

Nesta segunda parte, veremos os algoritmos de ordenação que tem desempenho sub-quadrático. Quando têm seu desempenho comparado com os algoritmos da aula anterior, estes sempre dão lavada. A contrapartida é que são algoritmos mais complicados de entender e de implementar. Mas, nada que morda.

Este exercício tratará de 3 algoritmos, São eles:

Shell Tem este nome devido ao seu descobridor, o dr Shell em 1959. Parecido com o BOLHA no sentido de que ele troca vizinhos, mas não próximos e sim distantes. Adicionalmente aos demais algoritmos, este precisa uma seqüência de controle, como operador adicional. Pode ter quantos números se quiser, desde que o último seja sempre 1.

HeapSort Usa como estrutura auxiliar um heap. Tem desempenho excelente, e se o programador souber bem o que é um heap, sua implementação é fácil e o seu conceito mais ainda.

Quick Devido a C.R.Hoare em 1962, este algoritmo é o campeão. Ninguém até hoje conseguiu ser mais rápido que ele. Sua estratégia passa pela escolha de pivots, que dividem o conjunto sendo classificado em 2 partes. Uma contém os elementos menores que o pivot e a outra os maiores. Levando esse procedimento ao limite, o conjunto estará classificado. Este é um excelente exemplo da estratégia “dividir e conquistar”.

Algoritmo Shell

```
1: inteiro k, h, i, x, j
2: TAM ← Tamanho do vetor a ordenar
3: DECR ← {seqüência de controle desejada}
4: para K = 1 até tamanho(DECR) faça
5:   H ← DECR[K]
6:   I ← H
7:   enquanto (I < TAM) faça
8:     X ← V[I + 1]
9:     J ← I
10:    enquanto (J ≥ H) ∧ (V[1 + J - H] > X) faça
11:      V[1 + J] ← V[1 + J - H]
12:      J ← J - H
13:    fim enquanto
14:    V[1 + J] ← X
15:    I ++
16:  fim enquanto(*4*)
17: fim para
```

Algoritmo Heap Sort

```
1: inteiro I,J
2: TAM ← Tamanho do vetor a ordenar
3: I ← ⌊(TAM ÷ 2)⌋
4: para J = I até 1, passo = -1 faça
5:   desce J,TAM
6: fim para
7: para J = TAM até 1, passo = -1 faça
8:   V[J] ← exclui J
9: fim para(*3*)
10: devolva V
```

Entrada: inteiro **função** exclui (int J)

```
1: devolva V[1]
2: V[1] ← V[J]
3: desce (1, J)
```

Entrada: **função** desce (int K, TAM)

```
1: inteiro J
2: J ← K × 2
3: se (J ≤ TAM) então
4:   se (J < TAM) então
5:     se (V[J + 1] > V[J]) então
6:       J ++
7:     fim se
8:   fim se
9:   se (V[K] < V[J]) então
10:    AUX ← V[K]
11:    V[K] ← V[J]
12:    V[J] ← AUX
13:    desce (J, TAM)
14:  fim se
15: fim se
```

Algoritmo QUICK

Entrada: **função** RAP(inteiro I, F)

```
1: inteiro ANT,DEP,MEI,AUX
2: ANT ← I
3: DEP ← F
4: MEI ← V[(I + F) ÷ 2]
5: repita
6:   enquanto (V[ANT] < MEI) faça
7:     ANT ++
8:   fim enquanto
9:   enquanto (V[DEP] > MEI) faça
10:     DEP --
11:   fim enquanto
12:   se (ANT ≤ DEP) então
13:     AUX ← V[ANT]
14:     V[ANT] ← V[DEP]
15:     V[DEP] ← AUX
16:     ANT ++
17:     DEP --
18:   fim se
19: até (ANT > DEP) {(1*)}
20: se (I < DEP) então
21:   RAP (I, DEP)
22: fim se
23: se (ANT < F) então
24:   RAP (ANT, F)
25: fim se
Entrada: fim função
26: TAM ← Tamanho do vetor a ordenar
27: RAP (1,TAM)
```

Desempenho

Compare os desempenhos dos 3 algoritmos estudados e reveja os valores do exercício passado.¹

Tamanho e tipo	Shell	HeapSort	Quick
100 aleatórios	0.1 seg	0.1 seg	0.06 seg
100 quasi-ordenados	0.1 seg	0.1 seg	0.06 seg
500 aleatórios	1 seg	1 seg	0.5 seg
500 quasi-ordenados	0.5 seg	1 seg	0.3 seg
1000 aleatórios	2 seg	2 seg	1 seg
1000 quasi-ordenados	1 seg	2 seg	0.8 seg
2500 aleatórios	8 seg	6 seg	3 seg
2500 quasi-ordenados	3 seg	6 seg	2 seg

Exemplo

Para fazer este exercício você deve fazer um mini-chinês em cada um dos 3 algoritmos. Cada um deles receberá um vetor diferente de 10 inteiros. Cada vez que o seu chinês passar pelo lugar marcado com (*n*) no algoritmo, você deve imprimir o conteúdo de V. Na n-ésima vez em que passar lá, responda o elemento de V que é pedido. Acompanhe nos exemplos.

Se o algoritmo SHELL for executado a seqüência de controle {4, 1} e com o vetor

61 15 79 72 45 9 58 29 44 94

Ao passar a quarta vez pelo lugar marcado, V[1]=45.

Se o algoritmo HEAPSORT for executado com o vetor

16 26 86 53 40 8 83 13 67 74

Ao passar a terceira vez por (*3*), V[2]=67

Se o algoritmo QUICK for executado com o vetor

16 29 24 99 62 83 40 47 98 14

Ao passar a primeira vez por (*1*), V[7]=83

Para você fazer

Algoritmo SHELL, com seqüência = {4,1}

V = 17 85 80 34 87 43 56 45 51 30 . Qual será V[6] = _____ ?

Algoritmo HEAPSORT

V = 2 46 28 54 29 90 13 14 94 71 . Qual será V[8] = _____ ?

Algoritmo QUICK

V = 66 38 84 19 6 34 67 53 15 60 . Qual será V[1] = _____ ?

1

¹Medidas feitas com um computador Pentium 500MHz, com Windows Millenium usando a linguagem APL plus, versão 6.0

Redes de comparação e de ordenação

Como já se viu, o melhor algoritmo de ordenação tem custo $O(n \times \log_2 n)$. O que não foi dito naquela ocasião, mas que é verdade, é que este custo é para ambientes mono-processados, ou seja naqueles nos quais as operações são seriais, só é possível fazer uma operação de cada vez.

Suponha agora que você dispõe de um sistema que permita processamento paralelo (diversas operações a cada ciclo). Qual seria um bom algoritmo para obter uma ordenação mais rápida ?

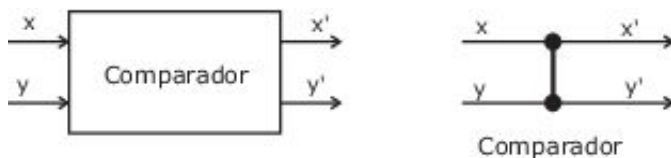
Trata-se do conceito de redes de ordenação. Usando-as, pode-se criar um algoritmo de ordenação de n elementos com custo $O(\log_2^2 n)$.

Para relembrar o significado dessas fórmulas, preencha:

se a entrada tiver	$O(n \times \log_2 n)$	$O(\log_2^2 n)$
10		
100		
1000		
1000000		

Uma rede de comparações é um conjunto de fios e de comparadores. Um comparador é um dispositivo com duas entradas (x, y) e duas saídas (x', y') que executa as seguintes funções: $x' = \min(x, y)$ e $y' = \max(x, y)$

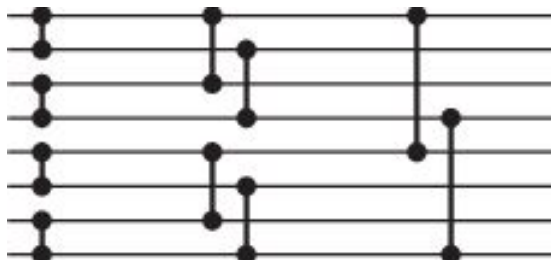
Um comparador pode ser apresentado como uma caixa (à moda LISP), ou mais praticamente como uma única linha vertical.



No desenho acima, as entradas aparecem à esquerda do comparador e as saídas à direita. Cada comparador opera com tempo $O(1)$, ou seja o tempo gasto entre a chegada das entradas e a obtenção das saídas é uma constante.

Uma rede de ordenação é um conjunto de comparadores interconectados por fios, de tal maneira que a saída de um comparador pode ser a entrada de outro.

Acompanhe no exemplo:



A rede de comparações acima, consegue responder qual é o maior e o menor elementos de uma entrada contendo 8 valores, usando apenas 3 comparações. Lembrando, 3 é o $\log_2 8$. Faça um chinês com os valores 2, 7, 1, 9, 3, 6, 4 e 8.

Representação de redes

Embora o desenho seja adequado e de fácil entendimento, precisa-se estabelecer uma representação matricial para as redes de comparação. Uma rede de n entradas e profundidade k , será representada por uma matriz binária de n linhas e k blocos. Cada bloco será separado dos seus vizinhos por uma coluna integralmente composta de zeros. A rigor, este separador é desnecessário, mas ele ajuda a identificar as etapas que podem ser executadas em paralelo.

O exemplo do desenho acima, seria assim representado:

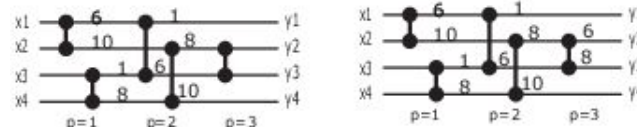
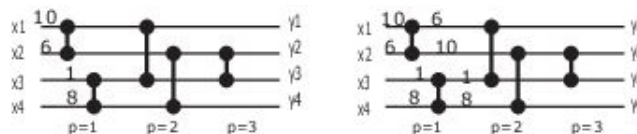
```

1 0 0 0 0 1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0 0 0 0 1 0
0 0 1 0 0 0 0 1 0 0 1 0 0
0 0 1 0 0 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0 0 1 0

```

Para entender o funcionamento de uma ordenação de $n=4$ números, acompanhe o exemplo a seguir, com as entradas = 10, 6, 1 e 8.

Execute os seguintes "chineses" na mesma rede de ordenação acima:



1,2,3,4
1,9,2,7
9,6,4,2
5,6,4,3

Represente a rede de ordenação acima, na forma de matriz:



Para você fazer

Seja a seguinte rede de comparações:

```

0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0

```

Suponha que a rede acima, tem como entrada os seguintes valores:

4 15 14 13 2 10 1 9 20 12

- Desenhe a rede na forma de fios e comparadores
- Forneça as entradas acima à rede
- Transporte os valores até obter a saída
- Responda os elementos pedidos.

Na resposta, Considere a saída como um vetor X , e responda:

$X[2] = \underline{\hspace{2cm}}$; $X[5] = \underline{\hspace{2cm}}$; $X[8] = \underline{\hspace{2cm}}$

Bibliografia: CORMEN, T. et alli, Algoritmos, pág 555 Parte "Tópicos Seleccionados", capítulo 27, Redes de Ordenação. (edição brasileira).

Busca em cadeias: algoritmo de Boyer-Moore

Seja a questão de localizar um determinado padrão P em um texto T. Por exemplo P="bola" em T="O jogador comeu a bola e depois a bolacha". Neste caso ocorre um "casamento"(match) na posição T_{19} e na T_{53} . Atente-se que qualquer editor de textos chinfrim tem esta função... O primeiro algoritmo que surge é o chamado "Força Bruta" e que pode ser assim descrito

Força Bruta

```
1: Algoritmo Força Bruta para busca em cadeias (padrão P, texto T)
2: i ← 1
3: CONTADOR ← 0
4: enquanto (i ≤ 1+(tamanho(T)-tamanho(P))) faça
5:   aux ← 0; j ← 1
6:   enquanto (j ≤ tamanho(P)) ∧ (T[aux + i] = P[j]) faça
7:     CONTADOR++
8:     aux++
9:     j++
10:  fim enquanto
11:  se j = tamanho(P) + 1 então
12:    imprima "ocorrência em ",i
13:    i++ {Em casos especiais, somar o tam(P), ao invés de 1}
14:  senão
15:    i++
16:  CONTADOR++
17: fim se
18: fim enquanto
19: fim algoritmo
```

Este algoritmo é claramente ruim, tem complexidade no pior caso de $O(\text{tamanho}(P) \times \text{tamanho}(T))$. Acompanhe no exemplo, a quantidade de testes que é feita:

T='o galo o gato e a gata gaguejaram' e P='gato'
QUANTIDADE DE TESTES = 38

Para você fazer

T=QUE SOMENTE POR POU SEU NA VOZ CONHECE
P=POR

Quantidade de testes: _____.

Algoritmo de Boyer-Moore

Este algoritmo, descrito na década de 70, usa 3 artifícios aceleradores que permitem aumentar o tamanho do salto

- a busca é feita da esquerda para a direita
- Pré-compila uma tabela de ocorrências de caracteres
- Pré-compila uma tabela de ocorrências de prefixos no padrão.

Os algoritmos são: (referência para todos CLR, pag 870)

```
1: Algoritmo BOYER-MOORE (texto T, padrao P)
2: tabela1 ← CHARACTERRUIM (padrão P, alfabeto S)
3: tabela2 ← SUFIXOBOM (padrão P)
4: i ← 0
5: enquanto (i ≤ tamanho(T) - tamanho(P)) faça
6:   j ← tamanho(P)
7:   enquanto (j > 0) ∧ P[j] = T[i + j] faça
8:     CONTADOR++
9:     j-
10:  fim enquanto
11:  CONTADOR++
12:  se (j = 0) então
13:    imprima "ocorrência em ",i
14:    i ← i + tamanho(P)
15:  senão
16:    i ← i + MAX ((j-tabela1[T[i+j]]), tabela2[j])
17:  fim se
18: fim enquanto
19: fim algoritmo
```

```
1: Algoritmo CHARACTERRUIM (padrão P, alfabeto S) {alfabeto S contém
  todos os caracteres de T (não repetidos)}
2: para cada caracter do alfabeto faça
3:   tabela1[índice do caracter] ← 0
4: fim para
5: para j de 1 até tamanho(P) faça
6:   tabela1 [índice do caracter P[j]] ← j
7: fim para
8: devolva tabela1
9: fim algoritmo
```

Exemplo deste algoritmo: CHARACTERRUIM ('gato', 'o/galteujrm') 4 0
1 2 0 3 0 0 0 0 0, significando salto de 4 para /leujrm, salto de 3 (4-1) para 'g', 2 (4-2) para 'a' e 1 para 't'. Nenhum salto para 'o'.

A segunda tabela é construída pela função SUFIXOBOM, cujo algoritmo é:

```
1: Algoritmo SUFIXOBOM (padrão P)
2: tabelaPI ← PREFIXO (P)
3: tabelaPILINHA ← PREFIXO (reverso P)
4: tabela2 ← tamanho(P) - tabelaPI [tamanho (P)] {repetir tam(P) vezes}
5: para LM de 1 até tamanho(P) faça
6:   j ← tamanho(P) - tabelaPILINHA [LM]
7:   se (tabela2 [j] > LM - tabelaPILINHA [LM]) então
8:     tabela2[j] ← LM - tabelaPILINHA [LM]
9:   fim se
10: fim para
11: devolva tabela2
12: fim algoritmo
```

```
1: Função PREFIXO (padrão P)
2: tabelaPREFIXOS ← 0..0 (o mesmo tamanho de P)
3: k ← 0
4: para q de 2 até tamanho(P) faça
5:   enquanto (k > 0) ∧ (P[k+1] ≠ P[q]) faça
6:     k ← tabelaPREFIXOS [k]
7:   fim enquanto
8:   se P[k+1] = P[q] então
9:     k++
10:   fim se
11:   tabelaPREFIXOS [q] ← k
12: fim para
13: devolva tabelaPREFIXOS
14: fim algoritmo
```

Acompanhe o exemplo do teste 'o galo o gato e a gata gaguejaram' para o padrão P='gato' no quadro negro. Note que neste exemplo o número de testes é de apenas 15. Compare com os 38 da FORÇA BRUTA.

Para você fazer

Você está recebendo um texto T e um padrão P. As tabelas 1 e 2 que são mais complexas de serem calculadas, também já estão sendo fornecidas. Calcule a quantidade de testes que são necessários nesta pesquisa.

T=QUE SOMENTE POR POU SEU NA VOZ CONHECE
P=POR

Q	U	E		S	O	M	N	T	P	R	A	V	Z	C	H				P	O	R
0	0	0	0	0	2	0	0	0	1	3	0	0	0	0	0			3	3	1	

Quantidade de testes: _____.

1

Um exemplo

```
'oba' MOSTRA 'oba a gororoba e boa'
o b a   g r e   o b a
T1= 1 2 3 0 0 0 0 T2= 3 3 1
oba a gororoba e boa
=
oba
S=3 *** ACHOU ***
-----
oba a gororoba e boa
=
oba
S=s1 max s2=3 T1:( ), j=3, s1=3; T2:(a), s2=1
-----
oba a gororoba e boa
=
oba
S=s1 max s2=3 T1:(r), j=3, s1=3; T2:(a), s2=1
-----
oba a gororoba e boa
=
oba
S=s1 max s2=2 T1:(o), j=3, s1=2; T2:(a), s2=1
-----
oba a gororoba e boa
=
oba
S=3 *** ACHOU ***
-----
oba a gororoba e boa
=
oba
S=s1 max s2=3 T1:( ), j=3, s1=3; T2:(a), s2=1
-----
oba a gororoba e boa
=
oba
S=s1 max s2=3 T1:(o), j=2, s1=1; T2:(b), s2=3
-----
Numero de saltos 14
```

==== 01/01/2026 11:51:38.9 =====E=PL531
1 38 17

Um exemplo:
p=EXAMPLE
t=HERE IS A SAMPLE EXAMPLE

Comecando o ciclo tem-se
EXAMPLE
HERE IS A SAMPLE EXAMPLE
-

Olhando o S sublinhado na posição correspondente
à última posição da palavra P, nós descobrimos várias coisas
- Não achamos o padrao P aqui, já que S <> E
- Nao acharemos nada, mesmo se deslocarmos o padrao P de 1
posição, (pois S <> L).
- Idem para 2 pois S <> P
- ...

Já que S não faz parte do padrao P, podemos deslocar o padrão
P do seu tamanho, sem medo de errar

Para fazer isso alinharemos o "S" do texto T, com um "S" virtual
que colocaremos no começo do padrão P. Este deslocamento
pode ser precalculado para todos os elementos do alfabeto e
armazenados em uma tabela.

SEXAMPLE
HERE IS A SIMPLE EXAMPLE

Note que alinhamos o "S" de antes com um "S" virtual no começo de P.
Olhemos agora o final. Temos E<>P.

O padrao P pode ser deslocado até a linhar a letra P embaixo com a
letra P encima.

SEXAMPLE
HERE IS A SIMPLE EXAMPLE

Olhando para o final. Ve-se que E=E. Podemos ter achado nosso
padrão.

VOLta-se uma posição e L=L, jóia.

Volta-se mais uma e ei-nos de sobre o P

Note que é a segunda vez que se analisa este mesmo "P".
Este pode ser um calcanhar de aquiles do método, que no
caso pessimista pode ficar ruim.

Olhando para o caracter anterior, temos M=M

Voltando mais uma fica A<>I.

Nós poderíamos deslocar o padrão até alinhar esse I com um I virtual
precedendo o padrão P.

MAs, podemos fazer melhor. Descobrimos que MPLE ocorre no texto.
Agora, podemos deslocar o padrão para alinhar esta recém descoberta
com o seu sufixo no padrão. (o qual é parcialmente imaginário).

MPLEXAMPLE
HERE IS A SIMPLE EXAMPLE

Neste caso, há apenas 7 substrings terminais do padrao P, assim
podemos précomputá-los e armazena-los em uma tabela.

O algoritmo olha esses dois deslocamentos e escolhe o maior.
fica

MPLEXAMPLE
HERE IS A SIMPLE EXAMPLE
-

Olha-se o "P" e deslocamos o padrão a contento e fica

EXAMPLE
HERE IS A SIMPLE EXAMPLE

beleza. Achou uma ocorrência.

COMO SE CONSTROEM AS DUAS TABELAS:Tabela 1: (Caracter ruim)
Ao se encontrar um T[x] diferente de um
P[y], em T1 se indica de quantas
posicoes pode-se saltar em x (no T), ou
seja, cada letra em T tem o
indice da ultima ocorrencia desse
caracter de T em P.

Exemplo:
Se T='o gato caiu' e P='gato', temos a primeira
comparacao entre T[4]=a e P[4]=o, como sao
diferentes, a T1 manda pular
o g a t c i u
T1= 4 0 1 2 3 0 0 0
2 posicoes, (ja que j=4 e T[a]=2 e 4-2=2)

Tabela 2: (Sufixo bom)

Realiza um processamento apenas em P, e busca
sufixos do padrao que tenham
sido repetidos anteriormente no padrao.

Por exemplo, se P='abeb', a T2 e: 4,4,2,1, ja
que o sufixo b (na p.4) ja ocorreu anteriormente
em P (na p.2). Note que isto so e verdade ja que
P[1]<> P[3]

Este exercicio complementa a folha e
deve ser feito no quadro

t=o galo o gato e a gata gaguejaram
p=gato

o g a l t e u j r m g a t o
T1= 4 0 1 2 0 3 0 0 0 0 0 T2= 4 4 4 1

O exercicio comeca:

o galo o gato e a gata gaguejaram
=
gato

Note que neste ponto, t[4]=a
e p[4]=o. Como houve diferenca
A T1[a] manda pular j-2=2 e a
T2[o] manda pular 1 so. PULO=2

S=s1 max s2=2 T1:(a), j=4, s1=2; T2:(o), s2=1

o galo o gato e a gata gaguejaram
=
gato

Apos a coincidencia de "o", volta-se 1
posicao e fica:

S=s1 max s2=4 T1:(l), j=3, s1=3; T2:(t), s2=4

o galo o gato e a gata gaguejaram
=
gato

S=s1 max s2=3 T1:(g), j=4, s1=3; T2:(o), s2=1

o galo o gato e a gata gaguejaram
=
gato
S=4 *** ACHOU ***

o galo o gato e a gata gaguejaram
=
gato

S=s1 max s2=2 T1:(a), j=4, s1=2; T2:(o), s2=1

o galo o gato e a gata gaguejaram
=
gato

S=s1 max s2=3 T1:(g), j=4, s1=3; T2:(o), s2=1

o galo o gato e a gata gaguejaram
=
gato

S=s1 max s2=2 T1:(a), j=4, s1=2; T2:(o), s2=1

o galo o gato e a gata gaguejaram
=
gato

S=s1 max s2=3 T1:(g), j=4, s1=3; T2:(o), s2=1

o galo o gato e a gata gaguejaram
=
gato

S=s1 max s2=4 T1:(u), j=4, s1=4; T2:(o), s2=1

o galo o gato e a gata gaguejaram
=
gato

S=s1 max s2=4 T1:(r), j=4, s1=4; T2:(o), s2=1

Numero de saltos 15

Bibliografia: CLR, pag 870 (versao em ingles)
Szwarcfiter e Markenzon, LTC
www.consiste.dimap.ufrn.br/~david/Disciplinas
/2001.2/DIM013/Recursos/sequencias.PDF
(um so endereco)

Em www-igm.univ-mlv.fr/~lecroq/string/node14.html
o visualizador da o mesmo resultado que esta folha

Em www.cs.utexas.edu/~moore/best-ideas/string-searching/
tem uma boa explicacao, que foi copiada parcialmente
aqui.

www-sr.informatik.uni-tuebingen.de/~buehler/BM/BM.html
boa animacao.

QUE SOMENTE POR POU SEU NA VOZ CONHECE
Q U E S O M N T P R A V Z C H P O R
T1= 0 0 0 0 0 2 0 0 0 1 3 0 0 0 0 T2= 3 3 1
QUE SOMENTE POR POU SEU NA VOZ CONHECE

=
POR
S=s1 max s2=3 T1:(E), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=1 T1:(O), j=3, s1=1; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(M), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(T), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=2 T1:(P), j=3, s1=2; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=3 *** ACHOU ***

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=1 T1:(O), j=3, s1=1; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(U), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(E), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(N), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(V), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(N), j=3, s1=3; T2:(R), s2=1

QUE SOMENTE POR POU SEU NA VOZ CONHECE
=
POR
S=s1 max s2=3 T1:(C), j=3, s1=3; T2:(R), s2=1

Numero de saltos 17

Busca em tabelas

Este exercício visa comparar diversas técnicas de acesso a uma tabela. Este problema é o escolhido por ser um dos problemas fundamentais em informática. Para cada uma das técnicas abaixo, você deve verificar quantas vezes o algoritmo mostrado será executado.

Busca Linear Para este algoritmo o conjunto de chaves não é mantido ordenado. O algoritmo básico é:

```
1: função BUSCALIN(inteiro CHAVE, inteiro TABELA[num])
2: inteiro i ← 1; qtd ← 0
3: enquanto i ≤ tamanho(TABELA) faça
4:   qtd ← qtd + 1
5:   se CHAVE = TABELA[i] então
6:     ...ACHOU...
7:     i ← tamanho(TABELA)
8:   fim se
9:   qtd ← qtd + 1
10:  i++
11: fim enquanto
12: devolva qtd
```

37	BUSCALIN	62	45	61	78	1	94	30	28	4	31	33	92	37	2	23	e	26
98	BUSCALIN	70	98	81	46	29	12	65	40	72	14	20	4	82	2	30	e	4
3	BUSCALIN	29	66	71	42	48	44	69	35	30	37	94	68	36	19	21	e	30

Busca linear ordenada Neste algoritmo, o conjunto de chaves deverá estar ordenado. Eis o algoritmo:

```
1: função BUSCALINORD (inteiro CHAVE, inteiro TABELA[num])
2: inteiro i ← 1; qtd ← 1
3: enquanto (i ≤ tamanho(TABELA)) ∧ (CHAVE > TABELA[i]) faça
4:   qtd ← qtd + 2
5:   i++
6: fim enquanto
7: se CHAVE = TABELA[i] então
8:   ...ACHOU...
9: fim se
10: devolva qtd
```

96	BUSCALINORD	5	10	26	34	39	48	56	60	68	70	73	89	92	93	96	e	29
58	BUSCALINORD	1	2	8	10	12	41	42	47	49	53	58	59	60	62	85	e	21
96	BUSCALINORD	2	12	21	26	27	37	43	46	52	54	65	79	83	84	89	e	31

Busca linear com sentinela A tabela não precisa estar ordenada, e a busca gasta apenas a metade dos testes da busca linear. A sentinela é a chave que se busca que é forçosamente incluída ao final da tabela. Assim, a saída do laço sempre se dará pela igualdade de chaves. Se a chave procurada for encontrada no meio da tabela, ...ACHOU... Se for encontrada no final da tabela, (onde foi explicitamente colocada) é porque ela não se encontrava lá antes do início do algoritmo.

```
1: função BUSCALINSEN (inteiro CHAVE, inteiro TABELA[num])
2: inteiro i ← 1; qtd ← 1
3: TABELA[ULTIMO+1] ← CHAVE {note que ULTIMO não é alterado}
4: enquanto CHAVE ≠ TABELA[i] faça
5:   qtd++
6:   i++
7: fim enquanto
8: se i ≠ (ULTIMO + 1) então
9:   ...ACHOU...
10: fim se
11: devolva qtd
```

38	BUSCALINSEN	24	35	91	18	8	72	10	88	27	54	28	34	43	3	38	e	15
54	BUSCALINSEN	63	17	35	68	27	86	54	53	5	78	41	58	44	51	25	e	7
61	BUSCALINSEN	75	67	51	60	85	38	49	74	73	78	66	35	24	4	2	e	16

Busca através de tabela HASH O gasto em espaço é recompensado com ótimo desempenho. O objetivo de qualquer pesquisa em tabelas hash é garantir um custo de $O(1)$. Para que isto ocorra a probabilidade de colisões deve ser próxima a zero. Uma desvantagem importante da tabela HASH é que a mesma não pode ser "visitada" em ordem, já que os dados não são armazenados em ordem. Para que a busca possa ocorrer, a tabela precisa ser criada previamente usando uma função de criação como a mostrada no quadro, por exemplo: Com esta função de inclusão, a função de busca seria:

```
1: função BUSCAHASH(inteiro CHAVE, inteiro TABELA[23])
2: inteiro i ← 1 + CHAVE mod 23
3: qtd ← 1
4: enquanto (CHAVE ≠ TABELA[i]) ∧ (TABELA[i] ≠ 0) faça
5:   qtd ← qtd + 2
6:   i++
7:   se i > 23 então
8:     i ← 1
```

```
9:   fim se
10: fim enquanto
11: se TABELA[i] ≠ 0 então
12:   ...ACHOU...
13: fim se
14: devolva qtd
```

Note que os números que vão ser fornecidos à função de busca ainda não são uma tabela HASH. Esta precisa ser criada antes da primeira consulta.

4	BUSCAHASH	18	78	73	90	44	93	81	4	57	65	34	82	37	29	20	e	3
97	BUSCAHASH	55	64	30	8	15	97	53	21	11	82	50	87	45	12	23	e	1
8	BUSCAHASH	40	18	32	94	75	9	65	22	54	25	16	33	80	86	27	e	11

Busca binária Conjunto ordenado, ótimo desempenho

```
1: função BUSCABIN( inteiro TABELA[15], inteiro CHAVE)
2: inteiro INIC,METADE,FIM
3: INIC ← 1 {campo delimita o limite inferior da pesquisa}
4: qtd ← 1
5: FIM ← 15 {delimita o limite superior da pesquisa}
6: repita
7:   METADE ← ⌊ ((INIC + FIM)/2)
8:   qtd++
9:   se CHAVE ≤ TABELA[METADE] então
10:     FIM ← METADE - 1
11:   senão
12:     INIC ← METADE + 1
13:   fim se
14:   qtd ← qtd+2
15: até TABELA[METADE] = CHAVE ∨ INIC > FIM
16: se TABELA[METADE] = CHAVE então
17:   devolva METADE
18: senão
19:   devolva -1
20: fim se
21: devolva qtd
```

63	BUSCABIN	1	2	7	22	29	35	40	41	43	63	77	86	89	90	95	e	10
39	BUSCABIN	18	20	21	32	35	39	40	50	51	53	65	79	91	97	99	e	10
21	BUSCABIN	27	28	31	42	50	51	53	64	65	73	74	83	88	93	98	e	13

Busca em ABP (árvore binária de pesquisa) Ótimo desempenho, fácil programação usando recursividade Para esta busca, antes a árvore deve ser criada, por exemplo, com a seguinte função, mostrada no quadro. Eis agora a busca

```
1: função BUSCAABP (inteiro CHAVE, inteiro ARVORE[num][3])
2: inteiro ONDE, qtd
3: qtd ← 0
4: ONDE ← RAIZ
5: enquanto ONDE ≠ -1 faça
6:   qtd++
7:   se CHAVE < TABELA[ONDE][3] então
8:     ONDE ← TABELA[ONDE][1]
9:   qtd++
10:  senão
11:    se CHAVE > TABELA[ONDE][3] então
12:      ONDE ← TABELA[ONDE][2]
13:    qtd++
14:    senão
15:      ...ACHOU...
16:    fim se
17:  fim se
18: fim enquanto
19: devolva qtd
```

28	BUSCAABP	35	42	64	85	28	24	68	23	6	7	59	71	94	46	37	e	3
34	BUSCAABP	98	53	67	93	77	75	66	69	41	70	51	76	65	59	34	e	7
55	BUSCAABP	32	96	46	21	95	72	28	39	24	34	84	79	53	68	73	e	14

Alguns dados experimentais

Para o teste a seguir, fiz 6 passagens de 1000 chaves. Para cada método, em cada passagem foram 3 tentativas: as primeiras 2 com sucesso e a última para uma chave inexistente. Eis os dados:

passagem	Lin	Lin Ord	Lin Sen	Hash	Bin	ABP
1	1265.3	1063.3	817.3	5.7	24.0	18.7
2	1777.3	749.3	621.7	5.0	30.0	18.7
3	1832.0	1314.0	674.0	5.7	29.0	20.0
4	1244.7	613.3	634.7	1.0	26.0	28.0
5	1200.7	1186.7	607.3	18.3	28.0	19.3
6	948.0	832.7	770.3	3.7	28.0	27.3
média	1378.0	959.9	687.6	6.6	27.5	22.0



Para você fazer

Calcule:

70	BUSCALIN	83	81	5	26	85	3	35	92	86	98	70	15	1	52	84	e	-----
25	BUSCALINORD	2	10	19	43	44	45	47	55	56	58	73	75	77	83	94	e	-----
64	BUSCALINSEN	51	12	98	65	20	19	5	64	55	34	61	79	9	97	2	e	-----
74	BUSCAHASH	94	16	60	38	30	74	17	26	86	4	49	40	20	8	63	e	-----
24	BUSCABIN	4	11	15	16	20	23	24	26	36	37	39	47	80	81	84	e	-----
60	BUSCAABP	89	5	61	60	77	3	58	78	86	57	97	69	19	38	26	e	-----

Casamento aproximado de cadeias

Uma importante tarefa em processamento de texto é o batimento de cadeias – achar todas as ocorrências de uma palavra em um texto. Lastimavelmente muitas palavras foram escritas de maneira errada. A questão é como podemos percorrer o texto localizando as palavras que foram escritas errado.

A pesquisa em biologia molecular precisa de ferramentas que suportem buscas em sequências de DNA por homologia (similaridade). Suponha que é necessário sequenciar um novo gen no homem e descobriu-se que ele é similar ao gene da hemoglobina em ratos. Sai daqui que este novo gene também produz hemoglobina e que qualquer diferença encontrada é o resultado de mutações genéticas no processo evolutivo.

Para ser mais preciso, seja P um padrão a pesquisar e T um texto sobre o mesmo alfabeto. A distância de edição entre P e T é o menor número de trocas suficientes para transformar um subconjunto de T em P , através das seguintes operações:

substituição Dois caracteres correspondentes são diferentes: $GATO \rightarrow GALO$ ($T \rightarrow P$)

inserção Adicionou-se um caracter a T , tal como está em P : $GTO \rightarrow GATO$ ($T \rightarrow P$)

exclusão Eliminou-se um caracter de T , que não está em P : $GAATO \rightarrow GATO$ ($T \rightarrow P$)

Por exemplo, $P = abcdefghijkl$ pode ser batido com $T = bcdefghixkl$ usando exatamente 3 operações, cada uma de um tipo dos acima mostrados. (substituição: $j \rightarrow x$, inserção: $f \rightarrow ff$ e exclusão: $a \rightarrow$).

Parece ser um problema muito difícil porque há que se decidir onde incluir e excluir caracteres no padrão e no texto. Entretanto, uma boa estratégia é pensar no problema de modo reverso. Que informação gostaríamos de ter para tomar a decisão final, isto é, o que deveria acontecer com o último caracter de cada cadeia? Os últimos caracteres podem coincidir (se eles forem iguais ou se um for substituído pelo outro) ou não.

Mais precisamente, Seja $D[i, j]$ o número mínimo de diferenças entre $P_1, P_2, \dots, P_i$ e o segmento de T terminando em j . $D[i, j]$ é o mínimo de três modos de estender cadeias pequenas.

1. Se ($P_i = T_j$) então $D[i-1, j-1]$, senão $D[i-1, j-1] + 1$. Isto significa que os caracteres batem ou deve-se substituir os caracteres i -ésimo e j -ésimo dependendo de onde eles concordam ou não.
2. $D[i-1, j] + 1$. Isto significa que há um caractere extra no P que deve ser contado. Então, não se avança o apontador de T e se paga o custo de uma inserção.
3. $D[i, j-1] + 1$. Isto significa que há um caracter extra em T que deve ser removido. Não se avança no apontador de P e paga-se o custo da exclusão.

O leitor deve ter percebido que não se especificou as condições limite da relação de recorrência. Isto é crítico para que o programa funcione bem e se obtenha corretamente a distância de edição. O valor de $D[0, i]$ deve corresponder ao custo de parear os primeiros i caracteres do texto com nenhum caractere do padrão. O que este valor deve ser depende do que você quer computar. Se você quer olhar o padrão inteiro contra o texto inteiro, isto significa que nós devemos excluir os primeiros i caracteres do texto, então $D[0, i] = i$ para pagar o custo das exclusões. Mas, e se o que queremos achar quando os padrões ocorrem em um texto bastante longo? Ele não deveria custar mais do que se o padrão casado começasse mais longe dentro do texto do que se ele estiver próximo do começo. Assim sendo, o custo inicial deve ser o mesmo para todas as posições. Neste caso, $D[0, i] = 0$, já que não há custo para excluir as primeiras i caracteres do texto. Em ambos os casos $D[i, 0] = i$ já que não podemos deletar os primeiros i caracteres do padrão sem penalidade.

Aceita a relação de recorrência, é direto transformá-la em um algoritmo de programação dinâmica que cria uma matriz D $n \times m$, onde n é o tamanho do padrão ($n = |P|$) e m é o tamanho do texto ($m = |T|$).

```
1: função Distância-Edição (P,T : cadeias)
2: para i=0 até n faça
3:     D[i,0]=i
4: fim para
5: para i=0 até m faça
6:     D[0,i]=i
7: fim para
8: para i=1 até n faça
9:     para j=1 até m faça
10:         C1 = D[i-1,j-1] + custo substituição( $s_i, t_j$ )
11:         C2 = D[i-1,j] + 1 {custo da exclusão em  $P_j$ }
12:         C3 = D[i,j-1] + 1 {custo da exclusão em  $T_i$ }
13:         D[i,j] = min(C1,C2,C3)
14:     fim para
15: fim para
```

Seja o exemplo onde $P=abcdefghijkl$ e $T=bcdefghixkl$

		b	c	d	e	f	f	g	h	i	x	k	l
	0	1	2	3	4	5	6	7	8	9	10	11	12
a	1	1	2	3	4	5	6	7	8	9	10	11	12
b	2	1	2	3	4	5	6	7	8	9	10	11	12
c	3	2	1	2	3	4	5	6	7	8	9	10	11
d	4	3	2	1	2	3	4	5	6	7	8	9	10
e	5	4	3	2	1	2	3	4	5	6	7	8	9
f	6	5	4	3	2	1	2	3	4	5	6	7	8
g	7	6	5	4	3	2	2	3	4	5	6	7	8
h	8	7	6	5	4	3	3	2	3	4	5	6	7
i	9	8	7	6	5	4	4	4	3	2	3	4	5
j	10	9	8	7	6	5	5	5	4	3	3	4	5
k	11	10	9	8	7	6	6	6	5	4	4	3	4
l	12	11	10	9	8	7	7	7	6	5	5	4	3

Para estudar como t virou p , basta examinar a sequência de respostas (em negrito) de 3 até 0. Quando 3 vira 2 tem-se uma substituição ($x \rightarrow j$), Quando 2 vira 1 é uma exclusão ($ff \rightarrow f$) e quando 1 vira 0 é uma inclusão (nada $\rightarrow a$). Os valores de retorno para as respostas ao problema de casamento de padrões depende do que se está interessado em buscar. Se for apenas o custo de comparação de todo o padrão P com todo o texto T , tal como na comparação de soletração de duas palavras, então necessita-se achar o custo de $D[n, m]$, como mostrado na tabela acima. Mas, se o que se quer é identificar o melhor casamento de uma subcadeia dentro de uma outra cadeia? Neste caso, precisa-se conhecer o menor custo do casamento do padrão P completo terminando em algum ponto qualquer do texto T . Portanto quer-se $\min_{1 \leq i \leq m} D[n, i]$ ou seja o menor custo da última linha de D . Este número apenas dá o custo do casamento ótimo, enquanto estamos frequentemente interessados em reconstituir o processo que foi feito: quais caracteres foram casados, substituídos, incluídos e excluídos. Isto pode ser reconstruído a partir de T e P , além da tabela D , sem necessidade de memória adicional, simplesmente a partir da célula de custo menor. A partir desta célula, vai-se caminhar para trás e para cima através da matriz. Dado o custo dos 3 vizinhos e os correspondentes caracteres pode-se reconstituir qual escolha foi feita para buscar a célula objetivo anterior. A direção de cada um dos passos para trás (para esquerda, acima ou em diagonal para esquerda-acima, identifica quando ocorreu uma exclusão, inclusão ou casamento/substituição.

Em resumo, na volta, tem-se

esquerda exclusão

acima inclusão

diagonal casamento ou substituição

Cadeias correspondentes a transições legais podem ser quebradas arbitrariamente, já que qualquer caminho custa a mesma coisa. Caminha-se para trás até chegar ao início da matriz. Esta fase custa $O(n + m)$. Já que são ocupadas apenas as células envolvidas no alinhamento. Note que conta-se uma operação cada vez que o número em análise diminui uma unidade.

Para saber mais, olhe o livro The Algorithm Design Manual de Steven Skiena, disponível na biblioteca da UP sob o número 005.1 S628a.

Um exemplo

Seja $P=GBQALIMEGGQSK$ e $T=GBQHLMAGGDSK$. A aplicação deste algoritmo resultará

	G	B	Q	H	L	K	M	A	G	G	D	S	K
0	1	2	3	4	5	6	7	8	9	10	11	12	13
G	1	0	1	2	3	4	5	6	7	8	9	10	11
B	2	1	0	1	2	3	4	5	6	7	8	9	10
Q	3	2	1	0	1	2	3	4	5	6	7	8	9
A	4	3	2	1	1	2	3	4	4	5	6	7	8
L	5	4	3	2	2	1	2	3	4	5	6	7	8
I	6	5	4	3	3	2	2	3	4	5	6	7	8
M	7	6	5	4	4	3	3	2	3	4	5	6	7
E	8	7	6	5	5	4	4	3	3	4	5	6	7
G	9	8	7	6	6	5	5	4	4	4	5	6	7
G	10	9	8	7	7	6	6	5	5	4	4	5	6
Q	11	10	9	8	8	7	7	6	6	5	4	4	5
S	12	11	10	9	9	8	8	7	7	6	5	5	4
K	13	12	11	10	10	9	8	8	7	6	6	5	4

Gerando uma distância de edição de 4.

Outros exemplos:

$P=IDDKCNBBFICJR$, $T=JDGHCNBBFICJI$, $d=4$
 $P=AMSGHKQHEMJQB$, $T=AMSGHKQJKHJQB$, $d=3$
 $P=SNLDQFAMANBLI$, $T=SFKIQFACAEBLI$, $d=5$

Para você fazer

Aplique o algoritmo acima à seguinte instância

$P=FSPLPFJIRNCMR$ e $T=LSPLIFJIRACMC$

e informe a seguir qual a distância de edição: _____.



Bioinformática - parte 2

Custo para transformar X em Y

Dada a cadeia X cujo comprimento é $t(x)$ e a cadeia Y cujo comprimento é $t(y)$ e criando uma cadeia Z que, ao final será igual a Y e dados os custos c_c de cópia, c_i de inclusão, c_e de exclusão e c_s de substituição, crie uma matriz M de $t(x) + 1$ linhas e $t(y) + 1$ colunas. O elemento $M[1, 1]$ dessa matriz supõe o cálculo da transformação de uma cadeia vazia nela mesma, logo $M[1, 1] = 0$. A partir daqui, faça $M[i, 1] \leftarrow (i - 1) \times c_e$. E $m[1, j] \leftarrow (j - 1) \times c_i$. Crie outra matriz contendo C, R, I ou D para indicar as operações de cópia, substituição, inclusão ou exclusão que determinarão a transformação de $X \rightarrow Y$ e a variável c_t para indicar o custo total da transformação.

Como no exercício anterior, primeiro vai-se criar as matrizes M e VOPS usando o algoritmo que tem como entradas x, y e os quatro custos: cc, cr, cd e ci.

```
1: M ← 1 + t(x) linhas e 1 + t(y) colunas de zeros
2: OPS ← idem de espaços
3: M[1;] ← j × coluna
4: M[:,1] ← i × linha
5: para i = 2; i ≤ 1 + t(x); i ++
6:   para j = 2; j ≤ 1 + t(y); j ++
7:     se x[i - 1] = y[j - 1]
8:       M[i; j] ← M[i - 1; j - 1] + cs
9:       OPS[i; j] ← C'
10:    senão
11:      M[i; j] ← M[i - 1; j - 1] + cr
12:      OPS[i; j] ← R'
13:    fim{se}
14:    se M[i - 1; j] + cd < M[i; j]
15:      M[i; j] ← M[i - 1; j] + cd
16:      OPS[i; j] ← D'
17:    fim{se}
18:    se M[i; j - 1] + ci < M[i; j]
19:      M[i; j] ← M[i; j - 1] + ci
20:      OPS[i; j] ← I'
21:    fim{se}
22:  fim{para}
23: fim{para}
24: retorne M, OPS
```

Na posição $M[i, j]$ está o custo total da transformação de X em Y. As operações necessárias para isso estão na matriz OPS que deve ser interpretado como

- C simplesmente ultrapassar o caracter em questão
- R sai o caracter de X e entra o de Y
- D sai o caracter de X
- I entra o caracter de Y

Na volta, a primeira operação a executar deve ser buscada em OPS[i; j] Se for C ou R, deve-se substituir i e j para buscar a próxima. Se for D, deve-se diminuir apenas i (subir uma linha) e se for I deve-se andar à esquerda subtraindo 1 unidade de j. Deve-se realizar este laço até chegar à célula 1,1 da matriz OPS.

Exemplos

1. Sejam $x = BCBCCB$, $y = ACBBBCBB$ e os custos $c_c = 1$, $c_r = 3$, $c_d = 6$ e $c_i = 6$. A matriz M é

0	6	12	18	24	30	36	42	48	IIIIIIII
6	3	9	13	19	25	31	37	43	DRRCCICC
12	9	4	10	16	22	26	32	38	DCIRRCII
18	15	10	5	11	17	23	27	33	DRDCCICC
24	21	16	11	8	14	18	24	30	DRCDRRCIR
30	27	22	17	14	11	15	21	27	DRCDRRCRR
36	33	28	23	18	15	14	16	22	DRDCCCRCC

e as operações são

```
insert A
insert C
copy B
replace C by B
```

copy C ***
replace C by B
copy B

2. Seja a cadeia $x = CACBBADD$ e a $y = BADCDDADBDDA$ com os custos $c_c = 4$, $c_r = 11$, $c_d = 14$, $c_i = 14$. As matrizes M e OPS geradas pelo algoritmo são

0	14	28	42	56	70	84	98	112	126	140	154	168
14	11	25	39	46	60	74	88	102	116	130	144	158
28	25	15	29	43	57	71	78	92	106	120	134	148
42	39	29	26	33	47	61	75	89	103	117	131	145
56	46	43	40	37	44	58	72	86	93	107	121	135
70	60	57	54	51	48	55	69	83	90	104	118	132
84	74	64	68	65	62	59	59	73	87	101	115	122
98	88	78	68	79	69	66	70	63	77	91	105	119
112	102	92	82	79	83	73	77	74	74	81	95	109

Como se pode ver o menor custo é de 109 e as operações a realizar são

```
replace C by B
copy A
insert D
copy C
replace B by D
replace B by D
copy A
insert D
insert B
copy D ***
copy D
insert A
```

Note que a inserção é a do caracter $y[j]$, a exclusão é a do caracter $x[i]$, a cópia é a do caracter $x[i]$ e a troca é a de $x[i]$ por $y[j]$. Note que a antepenúltima operação está marcada com ***.

Ocorrência de padrão em texto

O problema agora é o nosso velho conhecido como CTRL-F presente em qualquer editor de textos. Os parâmetros do algoritmo são uma cadeia chamada texto e outra cadeia de tamanho menor ou igual chamada padrão. O algoritmo deve retornar as posições dentro de texto onde padrão ocorre. Por exemplo se $t = \text{ACGTGACACAAAAC}$ e $p = \text{CA}$, então p ocorre em t nas posições 7 e 9. Existem inúmeros algoritmos que resolvem este problema. Alguns são mais indicados quando o padrão é curto, outros quando o padrão é longo. Eis alguns: Knuth-Morris-Pratt, Boyer-Moore, força bruta etc. O algoritmo a ser visto aqui é o do autômato-finito. Este algoritmo é muito eficiente, mas paga um preço (proporcional ao tamanho do padrão) para construir o AF. Um autômato finito (AF) é uma máquina de estados finita. Neste caso ela terá $m + 1$ estados onde m é o tamanho do padrão. Em termos práticos é uma matriz de $m + 1$ linhas por c colunas, onde c é a cardinalidade do texto (ou seja a quantidade de caracteres distintos existentes no texto). A busca tem um algoritmo simples:

```
1: ESTADO ← 0
2: para i = 1; i < tamanho(texto); i ++
3:   ESTADO ← AF[ESTADO;caracter[i]]
4:   se ESTADO=tamanho(padrao)
5:     escreva "o padrão ocorre na posição ",i-tamanho(padrao)
6:   fim{se}
7: fim{para}
Para construir o AF, usa-se o algoritmo:
```

```
1: QUAIS ← caracteres distintos do texto
2: NS ← matriz de t(p) + 1 linhas por t(quais) colunas com 0
3: para i = 0; i < t(p) + 1; i ++
4:   para j = 1; j ≤ t(quais); j ++
5:     para z = t(padrao); z ≤ 0; z --
6:       se prefixo com z posições do padrão coincidem com z posições do sufixo de Pi concatenado com o caracter j
7:         NS[i + 1; j] ← z
8:       fim{se}
9:     fim{para}
10:   fim{para}
11: fim{para}
12: retorne NS
```

Exemplos

1. Sejam $t = DABABDBCCCBDDDDCADAB$ e $p = DAB$. A matriz next-step é

0	0	0	1
2	0	0	1
0	3	0	1
0	0	0	1

e p ocorre nas posições 2 e 19 de t.

2. Sejam as variáveis t e p como segue. $t = \text{EDAEBBDBEEAACDBEEBDBA}$ e $p = \text{BDB}$. A matriz NS é

0	1	0	0	0
0	1	0	2	0
0	3	0	0	0
0	1	0	2	0

E o padrão ocorre nas posições 6 e 18.

Para você fazer

- 1. Custo de transformação: Suponha $x = \text{CBEEDDAE}$ e $y = \text{EBDAAACDAACB}$ com custos (cc, cr, cd e ci) de 7 9 12 12 . Gere M e OPS e responda

custo	antepenúltima operação

- 2. Ocorrência de padrão em texto. Suponha $t = \text{DCABCDBBCCCABBCDDBABAA}$ e $p = \text{BB}$. Escreva a seguir a tabela next-step para este problema

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

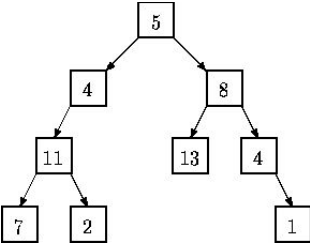
A propósito, na matemática básica os parênteses são usados para isto há séculos...

Parênteses

- Os parênteses tem diversas funções no mundo da T.I.
- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
 - Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas. Por exemplo, no LISP.
 - Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
 - Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
 - Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

(5 (4 (11 (7 () ()) (2 () ())) ()) (8 (13 () ()) (4 () (1 () ())))))

Note que nesta formulação todas as folhas da árvore tem a forma (inteiro () ()). Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar “sim” caso haja o caminho somando o valor solicitado ou “não” caso não haja.

Exemplo
22 (5(4(11(7() ()) (2() ()) ()) (8(13() ()) (4() (1() ())))))) sim
20 (5(4(11(7() ()) (2() ()) ()) (8(13() ()) (4() (1() ()))))) não
10 (3 (2 (4 () ()) (8 () () ()) (1 (6 () ()) (4 () () ())))) sim
5 () não

Para você fazer

Responda sim ou não para as 5 árvores seguintes:

- 57 (18(2() (8(3() ()) (9() (16(14(13() ()) ()))))) (19() (20() ()))))
- 31 (6(2() (3() ())) (20(11() (15(13(12() ()) (14() ())) (17() ()))))))
- 43 (1() (7(5(3(2() ()) ()) ()) (11() (14() (16() (17() (20() ()))))))))
- 6 (3(2(1() ()) ()) (12(9(8() ()) (10() (11() ()))) (16() (19() ()))))
- 25 (8(7(1() (4() (5() ())) ()) ()) (11() (15(13() ()) (16() (18() ()))))))

Responda aqui:

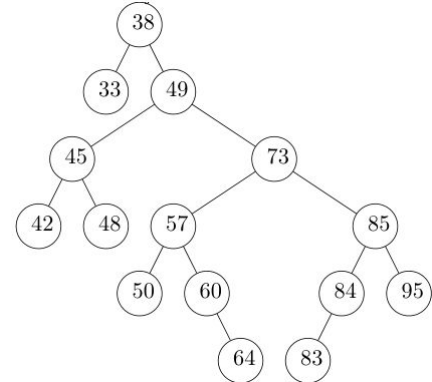
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

- lei fundamental da árvore binária de pesquisa (ABP)** filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.
- raiz** é o único nodo que não tem pai (no exemplo 38)
- folhas** nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)
- altura** é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)
- irmãos** nodos que tem o mesmo pai (50 é irmão do 60)
- netos** filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38).
- grau** número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma (raiz(filhoesquerdo)(filhodireito)). Note que (x () ()) pode aparecer como (x).

Eis como ficaria a representação do exemplo acima parentizado:

(38(33(49(45(42(48))(73(57(50(60() (64)))(85(84(83() () (95))))))

Exemplos

```
A árvore ABP
(4(3(1())(13(7() (10)) (21(17(14)()) (30(24() (27() (28))) ())))))
tem altura = 6 e a ABP
(28(20(6(3(1(5)) (7() (19(14) ()))) (24(23) (26))) ()))
tem altura = 5
```

🔗 Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(5(4(13(9(6)(11)) (20(19(17(16)(18)) ())) (30(27) ())))
(21(4(2) (14(9(5)(12)) ())) (29(24(22() (23)) ())) ()))
```

Responda aqui:

6	7

Analizador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
(a+b)	certo
((3+5)	errado
((((1*6)))	certo
)33-4(	errado
2*7	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos (), parênteses bicudos <>, colchetes [] e chaves {}.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um “abre” é encontrado ele é empilhado. Quando um “fecha” é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder **sim** se a expressão tiver parênteses corretos e **não** se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>]]-<((1/1)+(8+8))>+<2/7>-{2-4}]]>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]]>
[[<{5-2>/<8*6>]]+(1/2)+<4/1>]]/<[4*3]-{1-3}>+(4-{3+3})>>
[[{({4/8}/{2-3})/[<8+2>*[5+6]]]-<[8+2]+{1-7}>+<9-}>5*4}>]]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note +{1/2}, este parênteses redondo está errado. Na quarta veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

🔗 Para você fazer

```
<{<(5+1)-4>/[7*<9-2>]]*{[7/6]-([9*4]-{7*7}))}>
<[[((6/3)+<5+2>)]+{(5-8)*[9*3]}/{[<5/6/<7+4>]/[3+6]]}>
<<<[4-3]*[8-1]>/{[1+4]-[8*8]}>+<[8-1*6]>+{[8-2]*[9+3]}>>
{[({(4/6)-{7-2})*{4*7}]-[6*{5+5}]+<(6*7)+{5/4}>]}
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na agora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo...* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E (∧), OU (∨), NÃO (¬), SE... ENTÃO (→) e SE E SOMENTE SE... ENTÃO (↔).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se

$((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q)$, que vale 1.

b. agora $p = 1$ e $q = 0$, então

$((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p))$ vale 1.

c. p e q valem 1. resolvendo

$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge ((\sim p) \wedge (\sim q)))$, resulta 1.

d. $p = 0$ e $q = 1$, e tem-se

$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge ((\sim p) \vee (\sim p))) \vee ((\sim p) \vee q))$, cujo resultado é 0.

🔗 Para você fazer

11. Suponha que p vale 1 e q vale 1. Calcule:

$((((\sim q) \vee p) \wedge (q \wedge p)) \rightarrow ((\sim p) \wedge (\sim q)))$

12. Suponha que p vale 0 e q vale 1. Calcule:

$((q \vee (\sim p)) \wedge (p \rightarrow (\sim q))) \leftrightarrow ((\sim p) \wedge q)$

13. Suponha que p vale 0 e q vale 0. Calcule:

$(((((\sim p) \rightarrow q) \leftrightarrow (q \rightarrow q)) \vee ((\sim p) \rightarrow p)) \leftrightarrow (q \vee (\sim p)))$

14. Suponha que p vale 1 e q vale 1. Calcule:

$(((((\sim q) \vee (\sim p)) \rightarrow (q \wedge q)) \vee (p \vee q)) \wedge (q \wedge p))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b . É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+ a b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a b +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+ a b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

🔗 Para você fazer

```
(- (+ (* (+ 6 3) 3) (- (* 5 3) 2)) 4)
(+ 3 (+ 1 (* (+ (- (* 7 2) 4) (* (* 5 3) 1)) 3)))
(- (+ (+ 6 (+ 8 3)) (+ (- 7 2) 5)) 6)
(- (* (- (* 7 4) (+ 5 1)) 1) (- (* (- 5 2) 2) 5))
```

Responda aqui:

15	16	17	18



Análise Estilística de Textos

Ao usar o computador para compor e editar textos no idioma pátrio, abre-se a oportunidade de usar esse mesmo computador para fazer algumas contagens e após isso revelar métricas de produção de textos.

A principal métrica refere-se ao domínio e uso do vocabulário e ela informa qual o universo de palavras é de uso comum pelo autor. Estudos sugerem que esta característica é bastante pessoal e constante podendo chegar a ser usada como certificadora de autoria.

O processo se desenvolve em duas etapas: a preparação e a análise do texto.

A idéia – grosso modo – é obter um ou vários textos garantidamente de um autor e submetê-lo(s) a este algoritmo.

Preparação

- Depurar o texto, eliminando números e caracteres gráficos especiais (parênteses, travessões, ...). Se possível deve-se eliminar também as citações, já que estas são de outro autor.
- Padronizar o texto, transformando-o todo em maiúsculas, a fim de evitar que a caixa do texto interfira na contagem.
- Especializar o texto, criando uma lista de StopWords (SW). As SWs são palavras auxiliares no idioma e que pode convir não tratar. Em português poderiam ser – por exemplo – artigos e pronomes. Obviamente esta etapa é opcional, ou dito de outra maneira, a lista de SW pode ter qualquer comprimento, desde zero (SW=Φ) até o tamanho total do texto.
- Construir o vocabulário que vem a ser uma espécie de histograma de palavras. Ou seja cada vez que uma nova palavra aparecer no texto, ela será incluída no vocabulário. Alguns autores sugerem incluir nas SW todos os nomes próprios do texto.
- Construir as ocorrências, que são os apontadores de cada palavra para as posições em que elas ocorrem no texto.

Até aqui, nosso algoritmo recebeu uma entrada (o texto original) e construiu 2 saídas: o vocabulário e as ocorrências. Acompanhe no exemplo:

Texto original
Ouviram do Ipiranga as margens plácidas De um povo heróico o brado retumbante, E o sol da liberdade, em raios fúlgidos, Brilhou no céu da pátria nesse instante. E o povo cantou.

Ouviram do Ipiranga as margens plácidas De um povo heróico o brado retumbante E o sol da liberdade em raios fúlgidos Brilhou no céu da pátria nesse instante E o povo cantou

OUIRAM DO IPIRANGA AS MARGENS PL CIDAS DE UM POVO HER ICO O BRADO RETUMBANTE E O SOL DA LIBERDADE EM RAIOS FLGIDOS BRILHOU NO CÉU DA P TRIA NESSE INSTANTE E O POVO CANTOU

SW=E, O, A, DE, DA, OS, AS, EM, NO, NA

OUIRAM IPIRANGA MARGENS PL CIDAS UM
1 9 18 26 35
POVO HER ICO BRADO RETUMBANTE SOL
38 43 51 57 68
LIBERDADE RAIOS FLGIDOS BRILHOU CÉU
72 82 88 97 105
P TRIA NESSE INSTANTE POVO CANTOU
109 116 122 131 136

Vocabulário	Ocorrências
BRADO	51
BRILHOU	97
CANTOU	136
CÉU	105

F LGIDOS	88
HER ICO	43
INSTANTE	122
IPIRANGA	9
LIBERDADE	72
MARGENS	18
NESSE	116
OUIRAM	1
PL CIDAS	26
POVO	38, 131
P TRIA	109
RAIOS	82
RETUMBANTE	57
SOL	68
UM	35

A propósito, uma métrica importante de correção no uso do idioma é a quantidade de palavras do vocabulário que não existem em um bom repositório do idioma e que não são nomes próprios. Dito de outro modo, são as palavras *erradas* do texto.

Análise

Para a análise do texto, a primeira coisa a fazer é ordenar o vocabulário em ordem decrescente de quantidade de ocorrências por palavra. Na sequência, podem-se obter as seguintes métricas:

amplitude vocabular é a cardinalidade do conjunto vocabulário dividido pelo número de palavras do texto. Quanto mais próximo a 1 maior é a amplitude vocabular

quartis vocabulares Quantos histogramas (palavras no vocabulário) correspondem a 25%, 50% e 75% do total de palavras usadas.

Lei de Heaps Esta lei informa que o vocabulário de um texto em linguagem natural obedece à

V = Kn^β

onde V é a quantidade de palavras do texto, n é a quantidade de histogramas e K e β são mais ou menos constantes para um determinado autor. (K varia entre 10 e 100. β varia entre 0 e 1, usualmente entre 0,4 e 0,6). K tem a ver com o tipo de texto, e β com a riqueza vocabular do autor. [Retirado de Ziviani, Projeto de Algoritmos, pág. 305].

Erros vocabulares A quantidade de ocorrências referentes a histogramas que não estão em um repositório do idioma e não são nomes próprios.

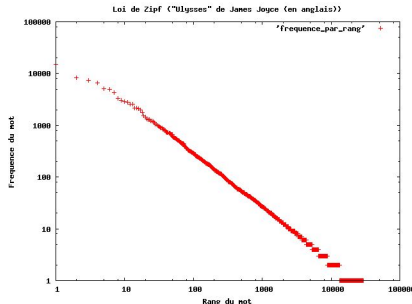
Outras formulações

George Zipf, linguista de Harvard, nos anos 40 elaborou uma lei empírica que diz que a frequência de uma palavra nas ocorrências ordenadas é ≈ 1/n^a onde n é a posição da palavra e a é próximo da unidade. Em palavras, a segunda palavra ocorre a metade das vezes da primeira, a terceira um terço da primeira e assim por diante.

Ele conduziu uma pesquisa sobre a monumental obra *Ulisses* de James Joyce e verificou que:

- ★ a palavra mais frequente ocorre 8000 vezes
- ★ a décima 800 vezes
- ★ a centésima 80 vezes
- ★ a milésima 8 vezes

Veja-se na figura a seguir o resultado obtido



Outra abordagem

Bate Byte, n. 35, 07/1994, diz, (por P.Kantek): ...Diante disto, comecei a elaborar um estudo particular para saber quais as palavras que eu mais freqüentemente uso, com o objetivo de elaborar um dicionário pessoal que aumente a minha produtividade durante a tarefa de digitação.

Comecei o levantamento na minha dissertação de mestrado, e os resultados a que cheguei estão a seguir descritos. Antes de analisá-los, entretanto, valem duas considerações: A primeira é de que trata-se de um texto específico, tratando sobre um aspecto único da computação. O segundo é que todo texto reflete um pouco do seu autor. Diante das duas questões, deve-se tomar muito cuidado ao generalizar os resultados.

A dissertação trata sobre o tema "downsizing", tem 163 páginas A4, com letra tamanho 10. É formada por 68.363 palavras. Dessas, apenas 42.704 têm três ou mais bytes e apenas 7.746 são diferentes.

As mais freqüentes:

que = 1234 ocorrências, sistemas = 702, para = 534, uma = 532, comandar = 439, não = 322, por = 294, mais = 274, como = 259, ser = 239, dar = 191, informática = 179, tem = 173, são = 167, esta = 130, suas = 130, este 122, pelo = 112, entre = 112, software = 111 e muito = 108.

Para você fazer

Você deve buscar o maior volume possível de texto escrito por você. Vale tudo: bilhetes, lançamentos no FB, trabalhos escolares (que não sejam copiados da wikipedia), obtendo um único arquivo garantidamente de sua autoria. Calcule V, n, amplitude vocabular, os quartis vocabulares e a lei de Heaps.

Faça a mesma coisa com um conjunto de todos os programas C (ou Java, ou Maple, ou ... – mas não misture) e calcule as mesmas variáveis.

Finalmente, compare os valores numéricos obtidos em cada caso e redija uma conclusão compatível com os achados. Devolva os valores e a conclusão para o professor.

Texto de própria lavra

V	n	AV	25%	50%	75%	Heaps

Programas Fonte

V	n	AV	25%	50%	75%	Heaps

Conclusão



Código Reed-Solomon

Este código inventado em 1960 por Irving S. Reed e Gustave Solomon no laboratório Lincoln do MIT (Massachusetts Institute of Technology) é um código de correção de erros. O artigo seminal é “Polynomial Codes over Certain Finite Fields.” (Reed & Solomon 1960). Um algoritmo eficiente de codificação só foi aparecer em 1969. Em um processo de transmissão ou armazenamento de dados, ao se usar este código é possível recuperar algum nível de erros de transmissão. Obviamente como não existe almoço grátis só é possível ter esta possibilidade ao se adicionar algum nível de redundância na transmissão. Existem diversos códigos que fazem isso, mas o código RS (Reed-Solomon) é um bom compromisso entre eficiência e complexidade. Ele é usado em

- TV digital
- no padrão QRCODE
- na missão Voyager (1977)
- na gravação de CDs (a partir de 1982)

Ainda são muito usados, se bem que em alguns casos, estão sendo lentamente substituídos por um código mais moderno (surgido em 1993): os códigos turbo.

Grupo Um **grupo** é uma tripla $(G, \times, 1)$ onde G é um conjunto, $\times$ é um operador diádico em G e $1 \in G$ é a identidade. G deve ter as seguintes propriedades:

- $\times$ é fechada. $\forall a, b \in G, a \times b \in G$.
- $\times$ é associativo. $\forall a, b, c \in G, (a \times b) \times c \equiv a \times (b \times c)$.
- $\forall a \in G, a \times 1 \equiv 1 \times a \equiv a$.
- $\forall a \in G$ existe um elemento chamado $a^{-1} \in g$ tal que $a \times a^{-1} = 1$. a é chamado *inverso* de a .

Campo Um **campo** é uma t-upla $(F, +, \times, 0, 1)$ onde F é um conjunto, $+$ é a adição, $\times$ é a multiplicação, $0 \in F$ é a identidade aditiva, e $1 \in F$ é a identidade multiplicativa. F deve ter as seguintes propriedades:

- $(F, +, 0)$ é um grupo.
- $\times$ é associativa e distributiva sobre $+$.
- $(F - \{0\}, \times, 1)$ é um grupo.

Você já conhece e trabalhou muito com campos. São campos os reais, os complexos e os números racionais. Os inteiros não formam um campo já que eles satisfazem as 2 primeiras propriedades mas falham na terceira. Conjuntos que admitem as duas primeiras, mas falham na terceira (como os inteiros) constituem um anel.

Campo de Galois Se p é um número primo, então o conjunto de inteiros módulo p (denotado $\mathbb{Z}_p$) é um campo (resultado obtido da teoria de números) e é conhecido como Campo de Galois (em inglês, Galois Field ou GF). Um CG consiste de elementos, baseados num elemento primitivo, normalmente igual a 2, com os seguintes valores

$$0, 2^0, 2^1, 2^2, ..., 2^{N-1}$$

Por exemplo, no conjunto de 2^m é conhecido como $GF(2^m)$ ou $(m = 4)$, $GF(16)$ cada elemento pode ser representado na forma polinomial

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

com $a_3a_2a_1a_0$ correspondendo aos números binários de 0000 a 1111. Nada impede que estes números sejam referidos como os decimais entre 0 e 15. A aritmética em campos finitos tem os processos de adição, subtração, multiplicação e divisão, mas eles funcionam diferente do que estamos acostumados (o campo é fechado, não esqueça). Ou seja, qualquer resultado encontrado tem que estar no campo finito.

Adição em GF Por exemplo na adição, somando x^3+x com x^3+x^2+1 deve resultar x^2+x+1 em $GF(16)$. Para fazer esta adição, usa-se a regra

- $a + b = 1$ quando $a \neq b$ e
- $a + b = 0$ quando $a = b$

Ou seja

$$\begin{array}{r} 1x^3 + 0x^2 + 1x + 0 \\ + \quad 1x^3 + 1x^2 + 0x + 1 \\ \hline = 0x^3 + 1x^2 + 1x + 1 \end{array} \begin{array}{r} 1010 \\ 1101 \\ 0111 \end{array} \begin{array}{r} 10 \\ 13 \\ 7 \end{array}$$

Generalizando esta operação, pode-se obter a seguinte tabela de adição em $GF(16)$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	4	7	6	5	2	3	0	13	12	15	14	9	8	11	10
6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Deve-se notar que a subtração gera exatamente o mesmo resultado da adição. Assim, por exemplo $2 - 2 = 0$ e $10 - 13 = 7$.

Multiplicação em GF Para a multiplicação precisa-se definir um polinômio gerador do campo, ou polinômio primitivo $p(x)$. Esse polinômio tem grau m no qual é irredutível, isto é não admite fatoração. Ele é necessário para estabelecer o processo de multiplicação num GF . Para $GF(16)$ um polinômio primitivo pode ser x^4+x+1 (outro poderia ser x^4+x^3+1). Todos os elementos diferentes de zero podem ser obtidos sabendo-se que o elemento primitivo (α) é uma raiz do polinômio gerador. Então, $\alpha^4 + \alpha + 1 = 0$ ou $\alpha^4 = -\alpha - 1$ (lembrar que $+$ pode ser substituído por $-$). A multiplicação num GF é o resto da divisão do produto dos dois elementos pelo polinômio primitivo.

Veja um exemplo: Seja 10×13 . Tem-se $x^3+x(\equiv 10) \times x^3+x^2+1(\equiv 13) = x^6+x^5+x^4+x$ (lembrar que $x^3+x^3=0$). Fazendo a divisão sugerida: $x^6+x^5+x^4+x$ dividido por x^4+x+1 a resposta é x^2+x+1 e o resto é x^3+x+1 ou $1011 \equiv 11_{10}$. Então em $GF(16)$, $10 \times 13 = 11$ e generalizando esta operação pode-se obter a seguinte tabela de multiplicação

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	0	2	4	6	8	10	12	14	3	1	7	5	11	9	15
3	0	3	6	5	12	15	10	9	11	8	13	14	7	4	1
4	0	4	8	12	3	7	11	15	6	2	14	10	5	1	13
5	0	5	10	15	7	2	13	8	14	11	4	1	9	12	3
6	0	6	12	10	11	13	7	1	5	3	9	15	14	8	2
7	0	7	14	9	15	8	1	6	13	10	3	4	2	5	12
8	0	8	3	11	6	14	5	13	12	4	15	7	10	2	9
9	0	9	1	8	2	11	3	10	4	13	5	12	6	15	7
10	0	10	7	13	14	4	9	3	15	5	8	2	1	11	6
11	0	11	5	14	10	1	15	4	7	12	2	9	13	6	8
12	0	12	11	7	5	9	14	2	10	6	1	13	15	3	4
13	0	13	9	4	1	12	8	5	2	15	11	6	3	14	10
14	0	14	15	1	13	3	2	12	9	7	6	8	4	10	11
15	0	15	13	2	9	6	4	11	1	14	12	3	8	7	5

Evariste Galois Antes de continuar, um pouco sobre a vida deste fantástico matemático francês. Nascido a 25 de outubro de 1811, o pai era administrador de uma escola para meninos em Bourg-la-Reine, hoje um subúrbio de Paris. Foi educado em casa pela mãe que tinha sólida formação humanista. Aos 12, entrou interno no Lycée Louis-le-Grand, uma excelente escola que vivia a agitação daqueles tempos, pós revolução francesa. A disciplina era mais rígida que a militar. Os alunos acordavam às 5h30 e tinham atividades até as 20h30. O desjejum era apenas pão e água. O menor desvio de disciplina era motivo para confinamento em solitária. Havia 12 celas para isso. Em 1832 ele apaixonou-se por Stephanie Potterin du Motel. Andaram trocando umas cartas meio ríspidas. Foi provocado a um duelo e aceitou (nas palavras dele, *a recusa era impossível*). Na véspera do duelo, resolveu passar a limpo alguns trabalhos matemáticos. Neles é possível ler *Je n'ai pas le temps = não tenho tempo*. O duelo ocorreu

na madrugada de 30 de maio de 1832 e ele morreu no dia seguinte no hospital.

Código Reed Solomon

Um código $RS(n, k)$ é construído com um polinômio gerador $g(x)$ com m fatores (onde m é a raiz quadrada de $n + 1$), com k símbolos de informação, e $m = n - k$ símbolos de paridade, com a possibilidade de correção de até t símbolos errados ($2t = m$). O campo de Galois será $GF(16)$ e cada símbolo constitui de uma palavra de 4 bits. Para este exemplo o polinômio gerador pode ser $g(x) = (x + \alpha^0)(x + \alpha^1)(x + \alpha^2)(x + \alpha^3)$ ou como $\alpha = 2$ fica $g(x) = (x + 1)(x + 2)(x + 4)(x + 8)$ fazendo a multiplicação em campos finitos fica: $g(x) = x^4 + 15x^3 + 3x^2 + x + 12$.

Agora, a mensagem a transmitir deve ser formada pelo polinômio $M(x)$ com k símbolos. Ela é multiplicada por x^m (multiplicação esta que simplesmente desloca à esquerda o polinômio por m casas). A seguir esta multiplicação é dividida pelo polinômio gerador. O resto obtido é somado a $M(x)$ e este conjunto é a mensagem que vai ser transmitida.

Acompanhe um exemplo completo disso tudo. Seja a mensagem $M(x) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11$ que na forma polinomial é $M(x) = x^{10} + 2x^9 + 3x^8 + 4x^7 + 5x^6 + 6x^5 + 7x^4 + 8x^3 + 9x^2 + 10x + 11$ multiplicando por x^4 fica

$$M(x) \times x^4 = x^{14} + 2x^{13} + 3x^{12} + 4x^{11} + 5x^{10} + 6x^9 + 7x^8 + 8x^7 + 9x^6 + 10x^5 + 11x^4 + 0x^3 + 0x^2 + 0x + 0$$

Dividindo esse polinômio pelo gerador $(x^4 + 15x^3 + 3x^2 + x + 12)$ o resto é $3x^3 + 3x^2 + 12x + 12$ e agora a mensagem a transmitir é $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3, 3, 12, 12$.

O erro $E(x)$ deve ser somado à mensagem transmitida e se ele tiver 2 ou menos coeficientes diferentes de zero, em qualquer uma das 15 posições, o conteúdo original vai poder ser recuperado na outra ponta, graças aos algoritmos de decodificação.

Estes são o pedaço mais complexo da coisa toda e sua implementação está além do escopo deste exercício. Eis como o decodificador funciona:

- Recebido a mensagem $R(x) = M(x) + E(x)$ (com possíveis E erros), calculam-se as síndromes S_i (neste caso são 4, e elas seriam obtidas substituindo-se em $R(x)$ x por $\alpha_0 = 1, \alpha_1 = 2, \alpha_2 = 4$ e $\alpha_3 = 8$).
- Através do algoritmo de Berlekamp's obtém-se o localizador de erros $\lambda(x)$ e o número de erros v .
- Dados $\lambda(x)$ e v , acham-se as raízes x_i de $\lambda(x)$, através do algoritmo de Pesquisa Chien.
- Agora, através do algoritmo de Forney, das síndromes e as raízes de $\lambda(x)$ acham-se os símbolos errados y_i na mensagem recebida.
- Finalmente, combinam-se todos os elementos acima e reconstitui-se a mensagem $M(x)$ original.

Para você fazer

Suponha que quer enviar uma mensagem usando a codificação Reed Solomon $RS(15, 11)$, formada por 11 símbolos (entre 0..15) e composta de

$$1 \quad 1 \quad 5 \quad 1 \quad 1 \quad 14 \quad 4 \quad 12 \quad 7 \quad 14 \quad 15$$

ou na forma de polinômio $x^{10} + x^9 + 5x^8 + x^7 + x^6 + 14x^5 + 4x^4 + 12x^3 + 7x^2 + 14x + 15$.

O polinômio gerador é o mesmo do exemplo acima, ou seja $x^4 + 15x^3 + 3x^2 + x + 12$, ou na forma dos seus coeficientes **1, 15, 3, 1 e 12**.

Calcule os 4 símbolos de paridade que devem ser agregados à mensagem original e transmitidos juntos desta, ao final da mesma.

Você deve encontrar 4 símbolos entre 0 e 15.



Grafos Definição formal: um grafo G é uma tripla $G = (V, A, F)$, onde V = conjunto de vértices do grafo; A = conjunto de arestas do grafo e F = função que relaciona arestas e vértices ($F : A \implies V \times V$). O desenho de um gráfico é apenas um instrumento auxiliar (embora muito poderoso) para visualizar as conexões.

Seja o exemplo
 $G_1 = (\{A, B, C, D\}, \{1, 2, 3, 4, 5, 6\}, \{1 \rightarrow A \times B, 2 \rightarrow B \times C, 3 \rightarrow B \times D, 4 \rightarrow B \times A, 5 \rightarrow C \times D, 6 \rightarrow D \times A\})$ G_1 seria representado como visto na figura representada no quadro.

Representações alternativas para o mesmo grafo:

$G_1 = (\{A, B, C, D\}, \{1, 2, 3, 4, 5, 6\}, \{1 = (A, B), 2 = (B, C), 3 = (B, D), 4 = (B, A), 5 = (C, D), 6 = (D, A)\})$
ou $G_1 = (\{A, B, C, D\}, \{1, 2, 3, 4, 5, 6\}, \{(1, A, B), (2, B, C), (3, B, D), (4, B, A), (5, C, D), (6, D, A)\})$

Observação: existe um excelente visualizador de grafos, freeware, denominado DOTGRAPH. Pode ser baixado em www.graphviz.org.

Alguns conceitos A seguir uma descrição dos principais conceitos associados a grafos. Ressalte-se que não existe uma terminologia universal para o tema, sendo a lista a seguir uma compilação das principais notações utilizadas pelos diversos autores.

Define-se ordem do grafo G , ao cardinal $|V|$, que nada mais é que o número de vértices do grafo.

Uma aresta pode ser definida por um par de vértices, que são conhecidos como extremidades da aresta e são conhecidos como vértices adjacentes

Um grafo é dito dirigido (ou digrafo) se suas arestas possuem orientação e nesse caso a aresta receberá o nome de arco. Visualmente falando, a orientação é representada por uma seta no destino da aresta.

Se o grafo não for dirigido ele é dito não dirigido (!). Nesse caso, a representação da aresta ligando os nodos a e b poderá ser representada como (a,b) ou (b,a) indistintamente.

Define-se origem, antecessor ou raiz de um arco ao primeiro vértice no par ordenado que define o arco.

Define-se destino, sucessor ou extremidade de um arco ao segundo vértice no par ordenado que o define.

Diz-se que dois arcos ou arestas são paralelos, quando ambos tem mesma origem e mesmo destino.

Define-se um arco ou aresta laço quando os dois vértices do seu par são iguais.

Se o grafo não possui laços ou arestas paralelas ele é denominado simples. Se as possui, diz-se que o grafo é multigrafo.

Um grafo completo é aquele que é simples e no qual cada par de vértices distintos é adjacente.

Define-se grau de um vértice $v \in V$, o número de vértices adjacentes a V .

Um vértice que não possui aresta incidente (vértice grau 0) é dito isolado.

Um vértice que só possui uma única aresta incidente (vértice de grau 1) é dito pendente.

Dois vértices a e b são ditos adjacentes quando a linha (a, b) existe no grafo.

Uma sequência de vértices adjacentes $v_1 \dots v_k$ é dita caminho de v_1 até v_k . Esse caminho é formado por $k - 1$ arestas. O valor $k - 1$ é o comprimento do caminho.

Um ciclo é um caminho que tem início e final no mesmo vértice. Se este caminho for elementar, o ciclo é dito elementar.

Um grafo que não contenha ciclos simples é chamado acíclico.

Um caminho que contenha cada vértice do grafo uma vez, é chamado hamiltoniano.

Um caminho que contenha cada aresta do grafo uma vez, é chamado euleriano.

Uma árvore é um grafo que não tem ciclos e é conexo.

Grafos usando listas encadeadas de arestas

Observação: contagens começam em 1 e não em zero.

Existem basicamente duas maneiras de representar grafos. A primeira usa um conjunto de listas encadeadas. Serão tantas listas quantos vértices houverem no grafo. Acompanhe no quadro negro um exemplo desta representação. A outra maneira é através de uma matriz de adjacência. Trata-se de uma matriz $n \times n$, onde o elemento i, j dessa matriz representa a aresta que conecta os vértices i e j . Tipicamente usa-se o primeiro caso para grafos esparsos, nos quais a quantidade de zeros seria muito grande na matriz de adjacência.

De qualquer maneira são representações que se equivalem, ou seja, sempre é possível passar de uma a outra.

Este exercício treinará a notação de listas encadeadas para representar grafos.

Definição: DIÂMETRO de um grafo é o maior caminho existente entre quaisquer par de vértices desse grafo.

Exemplo: Seja o grafo NÃO ORIENTADO, contendo 6 vértices e 7 arestas representado pelas tabelas a seguir, onde VÉRTICE é o número do vértice e PRIMEIRA ARESTA é o endereço da primeira aresta na tabela que representa as listas de arestas

VÉRTICE	1.ARESTA
1	8
2	16
3	13
4	2
5	9
6	15

Lista de Arestas			
	VERT	VALOR	PROX
1	2	10	-1
2	1	13	5
3	6	20	-1
4	3	18	14
5	5	20	-1
6	5	18	-1
7	6	10	-1
8	4	13	18
9	1	2	4
10	13	19	10
11	19	8	6
12	15	8	1
13	2	18	6
14	4	20	-1
15	1	20	1
16	3	18	7
17	6	12	20
18	5	2	3
19	3	14	12
20	6	17	16

Veja-se aqui como reconstituir o grafo deste exemplo:

O vértice 1 está ligado ao 4 por uma aresta valendo 13. O vértice 1 também está ligado ao 5 por uma aresta valendo 2 E também ao vértice 6 por uma aresta valendo 20.

Eis aqui, a título de esclarecimento, a matriz de adjacência do grafo

	1	2	3	4	5	6
1	0	0	0	13	2	20
2	0	0	18	0	0	10
3	0	18	0	0	18	0
4	13	0	0	0	20	0
5	2	0	18	20	0	0
6	20	10	0	0	0	0

Note que o grafo é não-orientado

Uma vez desenhado o grafo, deve-se obter a matriz de caminhos mínimos que neste caso é:

	1	2	3	4	5	6
1	4	30	20	13	2	20
2	30	20	18	43	32	10
3	20	18	36	33	18	28
4	13	43	33	26	15	33
5	2	32	18	15	4	22
6	20	10	28	33	22	20

Agora, as duas perguntas do exercício:

1. Qual o diâmetro do grafo ? (É o maior valor na matriz de CM): Resposta: 43, que vem a ser a ligação de 2 com 4 e 4 com 2

2. Quantas arestas "diminuíram de custo"? (Deve-se comparar a matriz CM com a matriz de adjacência originalmente construída) Resposta: 1.

note que a ligação 4 com 5 (e 5 com 4) tinha um custo de 20, mas na matriz de CM apareceu um caminho menor, valendo 15. É a única aresta onde isso ocorreu.

Para você fazer

Seja o seguinte grafo

VERTICES		LISTA DE ARESTAS			
VERT	PRIM	NUM	VERT	VALOR	PROX
1-	17	1-	5	4	-1
2-	3	2-	10	1	11
3-	8	3-	1	1	11
4-	5	4-	2	18	12
5-	7	5-	5	7	6
6-	16	6-	6	19	-1
		7-	2	4	13
		8-	1	11	4
		9-	1	4	18
		10-	4	19	-1
		11-	3	18	1
		12-	6	10	-1
		13-	4	7	-1
		14-	17	17	3
		15-	15	16	10
		16-	3	10	10
		17-	2	1	19
		18-	4	5	6
		19-	3	11	-1
		20-	6	12	6

Perguntas: 1. Qual o diâmetro deste grafo ?

R: _____.

2. Quantas arestas "diminuíram de custo"?

R: _____.

Note que APENAS as 14 (ou 7) arestas originais devem ser comparadas com os valores obtidos na matriz de caminhos mínimos.



Representação de grafos

Um grafo pode ser representado (entre outras formas) através de matrizes de incidência e de adjacência.

Dado um grafo com V vértices e A arestas, A matriz de incidência tem V linhas por A colunas. Cada coluna tem marcação (1,1) para grafos não orientado ou (1,2) para digrafos. (1=origem, 2=destino). Os demais valores da coluna são zero. A k-ésima aresta se encontra na k-ésima coluna. Laços são representados por um "3" único na coluna.

A matriz de adjacência tem V x V elementos. Se for não orientado, o grafo é simétrico em relação à diagonal principal. Se digrafo, a matriz não é simétrica. Laços são representados por um "1" na diagonal principal. As arestas são numeradas da esquerda para a direita e de cima para baixo.

Exercício

Verifique no quadro o grafo desenhado e escreva as matrizes de adjacência e de incidência para ele. A seguir, deixe de olhar para o desenho e responda

- 1. Quantos vértices tem o grafo ?
- 2. Quantas arestas tem o grafo ?
- 3. Qual o grau de cada vértice ?
- 4. Quantas arestas adjacentes possui a aresta número 5 ?
- 5. Quais são elas ?
- 6. Existem arestas paralelas no grafo ?
- 7. Existem laços ?
- 8. O grafo é simples ou é multigrafo ? (tem paralelas e/ou laços)
- 9. O grafo é completo ? (simples, e todos os vértices diretamente ligados)
- 10. Existem vértices isolados ?
- 11. Existe caminho entre o vértice 2 e o vértice 7 ?
- 12. Existe caminho Euleriano ?
- 13. Existe caminho Hamiltoniano ? (Problema do caixeiro viajante)

Caminhamento

A questão agora é como visitar todos os nodos. Se o critério exige conhecer todos os caminhos (na busca do menor trajeto, por exemplo) e se o grafo é completo, a quantidade de caminhos a examinar é de n!, onde n é o número de vértices. Se o grafo não é completo, o número de caminhos diminui, mas o problema continua NP.

Se se usar um critério mais relaxado, o processamento é bastante menor. A seguir, dois critérios usuais na visitação de um grafo. É de se notar que os dois são também importantes no contexto de árvores, e aqui é até mais fácil deles serem entendidos.

Caminhamento em Profundidade

- 1: Algoritmo CAM-PROFUNDIDADE {ADJ é a mat adjac. de grafo não direcionado}
- 2: P1 pilha de vertices
- 3: FLAG vetor aux (estado .V.)
- 4: INI é o vertice inicial*/
- 5: FLAG ← 1,1,1...1
- 6: P1 ← empilha INI
- 7: FLAG[INI] ← 2
- 8: enquanto não vazia(P1)
- 9: X ← desempilha P1
- 10: imprime X
- 11: FLAG[X] ← 3

- 12: VIZ ← nodos adj X c/e=1
 - 13: FLAG [VIZ] ← 2
 - 14: empilha VIZ
 - 15: fim{enquanto}
 - 16: fim algoritmo
- Obs: (a) empilhar primeiro o MAIOR ainda não processado; (b) Valores de FLAG: 2=na pilha; 3=já processado

Caminhamento em Largura

- 1: Algoritmo CAM-LARGURA {ADJ matriz de adjac. grafo não direcionado}
 - 2: F1 fila de vértices
 - 3: FLAG vetor aux (estado V)
 - 4: INI vértice inicial */
 - 5: FLAG ← 1,1,1...1
 - 6: F1 ← enfileira INI
 - 7: FLAG[INI] ← 2
 - 8: enquanto não vazia(F1)
 - 9: X ← desenfileira F1
 - 10: imprime X
 - 11: FLAG[X] ← 3
 - 12: VIZ ← nodos adj X c/e=1
 - 13: FLAG [VIZ] ← 2
 - 14: enfileira VIZ
 - 15: fim{enquanto}
 - 16: fim algoritmo
- Obs: (a) enfileirar primeiro o MENOR ainda não processado; (b) Valores de flag: 2=na pilha; 3=já processado

Exemplo

Para o exemplo, considere a matriz de adjacência

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1-	0	0	1	0	0	1	0	0	0	0	0	0	1	0
2-	0	0	0	0	1	0	1	0	0	1	0	1	0	1
3-	1	0	0	0	1	0	0	0	0	0	1	0	0	1
4-	0	0	0	0	0	0	0	0	1	0	0	0	0	0
5-	0	1	1	0	0	0	0	1	0	0	1	1	0	0
6-	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7-	0	1	0	0	0	0	0	0	0	1	0	0	0	0
8-	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9-	0	0	0	1	0	0	0	0	0	0	0	1	0	1
10-	0	1	0	0	0	0	1	0	0	0	0	1	1	0
11-	0	0	1	0	1	0	0	0	0	0	0	0	0	1
12-	0	1	0	0	1	0	0	0	1	1	0	0	1	0
13-	1	0	0	0	0	0	0	0	1	1	1	0	1	1
14-	0	1	1	0	0	0	0	0	1	0	0	0	1	0

- 1. Ache os nodos visitados no caminhamento em profundidade. Nodo inicial é o 3
- 2. Ache os nodos visitados no caminhamento em largura. Nodo inicial é o 5.

PS: Note que tanto no exemplo, quanto no exercicio a ser feito, ambos os grafos são NÃO DIRIGIDOS.

Para você fazer

Dada a matriz de ADJAC NCIA seguinte, resolva: Nodo inicial para o caminhamento em profundidade: 8 . Nodo inicial para o caminhamento em largura: 11

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1-	0	0	1	0	0	0	0	0	0	0	1	1	0	0
2-	0	0	0	0	0	0	1	0	1	0	0	0	0	0
3-	1	0	0	0	0	0	0	1	0	0	0	0	1	0
4-	0	0	0	0	0	0	0	1	0	0	0	0	1	0
5-	0	0	0	0	0	0	0	0	0	1	0	0	0	1
6-	0	0	0	0	0	0	0	0	0	0	0	1	0	0
7-	0	1	0	0	0	0	0	0	0	0	1	0	0	0
8-	0	0	1	1	0	0	0	0	1	0	1	0	0	1
9-	0	1	0	0	0	0	0	1	0	1	0	0	0	0
10-	0	0	0	0	1	0	0	0	1	0	0	0	0	1
11-	1	0	0	0	0	0	1	1	0	0	0	0	0	1
12-	1	0	0	0	0	1	0	0	0	0	0	0	0	1
13-	0	0	1	1	0	0	0	0	0	0	0	0	0	0
14-	0	0	0	0	1	0	0	1	0	1	1	1	0	0

R1= é o vetor dos nodos obtidos caminhando em profundidade (pilha)
R2= é vetor dos nodos obtidos no caminhamento em largura (fila)

R1[11]	R1[14]	R2[9]	R2[14]



Desenhando Grafos

O desenho agradável (bonito) de um grafo é um problema que surge constantemente em muitas aplicações (e mesmo em necessidades do dia a dia das pessoas). Por exemplo no desenho de uma árvore de diretórios, em diagramas esquemáticos de circuitos, em fluxogramas de resolução de tarefas, em estratégias de ataques a problemas entre inúmeros outros problemas.

Entretanto, a questão é: qual o significado do adjetivo agradável ou bonito acima empregado ? O ideal é que o desenho mostre a estrutura do grafo, de modo que o leitor possa entendê-lo. Também espera-se que o desenho seja esteticamente prazeroso. Lastimavelmente este segundo critério é muito vago, de maneira que não se pode diretamente implementá-lo em um algoritmo. Mas, é possível tentar esclarecer melhor o que se pretende:

- cruzamentos** Busca-se um desenho que minimize a quantidade de cruzamentos no grafo
- área** Busca-se um desenho que caiba no menor pedaço de papel possível enquanto se assegura que vertices diferentes ocuparão locais diferentes no desenho
- arestas longas** Devem ser minimizadas, dado que sua existência tende a distrair o leitor
- razão largura altura** Espera-se que o desenho obedeça uma proporção razoável, tipicamente na faixa 3/4 (o que inclui a tela do computador e uma folha de papel Ax ($x \in 0, 1, \dots, 6$)).

Tais objetivos parecem ser mutualmente contraditórios e o desafio de qual objetivo atender e qual ignorar é bastante complexo. É razoável supor que o desenho será tão melhor quanto menor a quantidade de vértices e arestas.

Onde buscar ajuda

- Georg Sander mantém uma página em [//www.cs.uni-sb.de/RW/users/sander/html/gstools.html](http://www.cs.uni-sb.de/RW/users/sander/html/gstools.html)
- Outro recurso é o pacote GraphEd. O pacote Sgraph é uma interface para o GraphEd que suporta extensões escritas pelo usuário em C. O GeaphEd é livre para aplicações não comerciais e pode ser obtido via FTP em forwiss.uni-passau.de (132.231.20.10) no diretório [/pub/local/graphed](http://pub/local/graphed).
- Graphlet é uma versão mais moderna do GraphEd, feita pelo mesmo grupo e está disponível em forwiss.uni-passau.de/Graphlet.
- GraphViz é um programa desenvolvido por Stephen North dos Laboratórios Bell. Faz bons desenhos mesmo em grafos de grande porte.

Usando o GRAPHVIZ

Baixe o programa e instále-o na máquina. A seguir crie os arquivos de entrada (contendo a especificação matemática) dos grafos e verifique o resultado obtido.

Execute os seguintes exemplos:

```
digraph G {
  a->b->c->d;
  b->d;
  d->e;
  a->e;
  b->e;
  e->a;
  d->a;
}

E também

digraph G {
  subgraph cluster0 {
    node [style=filled,color=red];
    style=filled;
    color=lightgrey;
    a0 -> a1 -> a2 -> a3;
    label = "process #1";
  }
  subgraph cluster1 {
    node [style=filled];
    b0 -> b1 -> b2 -> b3;
    label = "process #2";
    color=yellow
  }
  start -> a0;
  start -> b0;
  a1 -> b3;
  b2 -> a3;
  a3 -> a0;
  a3 -> end;
  b3 -> end;
  start [shape=Mdiamond];
  end [shape=Msquare];
}
```

Tabela de atributos de NODO

Nome	Default	Valores
color	black	cor do nodo
distortion	0.0	distorção do nodo para shape=polygon
fillcolor	lightgrey/black	cor de preenchimento do nodo
fixedsize	false	rótulo não tem efeito sobre o tamanho do nodo
fontcolor	black	cor das letras
fontname	Times-Roman	nome da fonte usada
fontsize	14 point	tamanho da fonte
height	.5	altura do nodo em polegadas
orientation	0.0	ângulo de rotação do nodo
regular	false	força o polígono a ser regular
shape	ellipse	formato do nodo; veja Seção 2.1 e Apêndice E
sides	4	número de lados para shape=polygon
width	.75	largura em polegadas

Atributos de aresta (edge)

Name	Default	Values
arrowhead	normal	estilo da flecha na ponta da aresta
arrowsize	1.0	fator de escala para a flecha
arrowtail	normal	estilo da cauda
color	black	cor da aresta
dir	forward	forward, back, both, ou none
fontcolor	black	cor do rótulo
fontname	Times-Roman	fonte
fontsize	14 point	tamanho do fonte
headlabel		rótulo colocado próximo à cabeça
headport		n,ne,e,se,s,sw,w,nw
label		rótulo da aresta
labelfontcolor	black	cor do rótulo
labelfontname	Times-Roman	fontes para rótulos
labelfontsize	14	tamanho para fonte de rótulos

Atributos de grafo

Name	Default	Values
bgcolor		cor de fundo
center	false	desenho no centro da página
fillcolor	black	cor de preenchimento do cluster
fontcolor	black	cor da fonte
fontname	Times-Roman	nome da fonte
fontsize	14	tamanho da fonte
label		rótulo do grafo
labeljust	centered	"l"e "r"para esquerda e direita
labelloc	top	"t"e "b"para superior e inferior
margin	.5	margem da página em polegadas
page		unidade da página, por exemplo "8.5,11"
ratio		razão desejada, fill ou auto
rotate		Se 90, estabelece "landscape"
size		tamanho máximo em polegadas
style		opções de desenho, por exemplo filled for clusters

Para executar este programa, instále-o e chame

```
dot arquivo.especs -Tpng -o desenho.png
```

🔗 Para você fazer

Construa o grafo que represente o fenômeno abaixo descrito. Para responder, escolha uma opção:

- Transcreva os comandos do GraphViz no verso desta folha, OU
- Gere o grafo, imprima-o e na mesma listagem imprima os comandos usados para a geração, OU
- Gere o grafo no seu micro, chame o professor e mostre o resultado a ele.

Construa um grafo descrevendo o que um técnico de informática deveria fazer para corrigir um defeito de um micro computador qualquer. Identificar pelo menos 8 tarefas e/ou decisões.

Caminho mínimo: algoritmo de Dijkstra

Este exercício usa a implementação de Rabuske [Rab92, UFSC, pág. 66]

Algoritmo de Menor Caminho de DIJKSTRA (adap. Rabuske)

```
1: algoritmo MENCAMDIJ
2: Variaveis ADJ = matriz de adjacência com ∞ em vez de zero
3: S = vetor de nodos já examinados
4: D = vetor de distâncias
5: V = conjunto de todos os vértices
6: para cada v pertencendo a V-v0 faça
7:   D[v] ← ADJ[v0;v]
8:   enquanto S ≠ V faça
9:     escolha o vertice w pertencendo a V-S tal que D[w] seja minimo
10:    coloque w em S
11:    para cada v pertencendo a V-S faça
12:      D[v] ← min (D[v], D[w]+ADJ[w,v])
13:    fim para
14:  fim enquanto
15: fim para
16: fim algoritmo
```

Acompanhe um exemplo de um grafo de 5 vértices
Matriz de adjacência original

99	2	99	99	10
99	99	3	99	7
99	99	99	4	99
99	99	99	99	99
99	99	8	5	99

Nodo inicial = 1

```
custo desde a origem até aqui = -1 2 99 99 10
menor custo para ir a algum lugar= 2 próximo nodo = 2 R[2]=2
linha de ADJ corresp a este nodo = 99 99 3 99 7
nodos já estudados = 1 2
```

```
custo desde a origem até aqui = -1 -1 5 99 9
menor custo para ir a algum lugar= 5 próximo nodo = 3 R[3]=5
linha de ADJ corresp a este nodo = 99 99 99 4 99
nodos já estudados = 1 2 3
```

```
custo desde a origem até aqui = -1 -1 -1 9 9
menor custo para ir a algum lugar= 9 próximo nodo = 4 R[4]=9
linha de ADJ corresp a este nodo = 99 99 99 99 99
nodos já estudados = 1 2 3 4
```

```
custo desde a origem até aqui = -1 -1 -1 -1 9
menor custo para ir a algum lugar= 9 próximo nodo = 5 R[5]=9
linha de ADJ corresp a este nodo = 99 99 8 5 99
nodos já estudados = 1 2 3 4 5
```

Resultado final = 0 2 5 9 9

Um exemplo

Acompanhe um exemplo de um grafo de ordem = 10
Matriz de adjacência original = (já com ∞ = 99)

	1	2	3	4	5	6	7	8	9	10
	==	==	==	==	==	==	==	==	==	==
1	99	6	99	8	4	99	99	13	15	99
2	20	99	99	99	17	4	99	99	99	99
3	99	14	99	99	99	15	99	14	12	2
4	17	99	14	99	99	99	5	99	99	99
5	99	1	99	99	99	99	9	99	99	99
6	99	99	14	99	99	99	99	99	14	99
7	9	99	99	1	99	4	99	14	99	99
8	99	99	4	99	99	99	99	99	20	18
9	17	99	99	99	99	3	99	99	99	99
10	99	10	99	99	99	99	99	99	99	99

Nodo inicial = 5

nodos já estudados = 5

```
custo desde a origem até aqui = 99 1 99 99 -1 99 99 9 99 99
menor custo para ir a algum lugar= 1
próximo nodo a estudar = 2 R[2] = 1
linha de ADJ corresp a este nodo = 20 99 99 99 99 17 4 99 99 99
nodos já estudados = 5 2
```

```
custo desde a origem até aqui = 21 -1 99 99 -1 18 5 9 99 99
menor custo para ir a algum lugar= 5
```

```
próximo nodo a estudar = 7 R[7] = 5
linha de ADJ corresp a este nodo = 9 99 99 1 99 4 99 14 99 99
nodos já estudados = 5 2 7
```

```
custo desde a origem até aqui = 14 -1 99 6 -1 9 -1 9 99 99
menor custo para ir a algum lugar= 6
próximo nodo a estudar = 4 R[4] = 6
linha de ADJ corresp a este nodo = 17 99 14 99 99 99 5 99 99 99
nodos já estudados = 5 2 7 4
```

```
custo desde a origem até aqui = 14 -1 20 -1 -1 9 -1 9 99 99
menor custo para ir a algum lugar= 9
próximo nodo a estudar = 6 R[6] = 9
linha de ADJ corresp a este nodo = 99 99 14 99 99 99 99 99 14 99
nodos já estudados = 5 2 7 4 6
```

```
custo desde a origem até aqui = 14 -1 20 -1 -1 -1 -1 9 23 99
menor custo para ir a algum lugar= 9
próximo nodo a estudar = 8 R[8] = 9
linha de ADJ corresp a este nodo = 99 99 4 99 99 99 99 99 20 18
nodos já estudados = 5 2 7 4 6 8
```

```
custo desde a origem até aqui = 14 -1 13 -1 -1 -1 -1 -1 23 27
menor custo para ir a algum lugar= 13
próximo nodo a estudar = 3 R[3] = 13
linha de ADJ corresp a este nodo = 99 14 99 99 99 15 99 14 12 2
nodos já estudados = 5 2 7 4 6 8 3
```

```
custo desde a origem até aqui = 14 -1 -1 -1 -1 -1 -1 -1 23 15
menor custo para ir a algum lugar= 14
próximo nodo a estudar = 1 R[1] = 14
linha de ADJ corresp a este nodo = 99 6 99 8 4 99 99 13 15 99
nodos já estudados = 5 2 7 4 6 8 3 1
```

```
custo desde a origem até aqui = -1 -1 -1 -1 -1 -1 -1 -1 23 15
menor custo para ir a algum lugar= 15
próximo nodo a estudar = 10 R[10] = 15
linha de ADJ corresp a este nodo = 99 10 99 99 99 99 99 99 99 99
nodos já estudados = 5 2 7 4 6 8 3 1 10
```

```
custo desde a origem até aqui = -1 -1 -1 -1 -1 -1 -1 -1 23 -1
menor custo para ir a algum lugar= 23
próximo nodo a estudar = 9 R[9] = 23
linha de ADJ corresp a este nodo = 17 99 99 99 99 3 99 99 99 99
nodos já estudados = 5 2 7 4 6 8 3 1 10 9
```

```
custo desde a origem até aqui = -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
Resultado final = 14 1 13 6 0 9 5 9 23 15
```



Para você fazer

Seja o seguinte grafo, como o nodo inicial igual a: NODO INICIAL: 2

	1	2	3	4	5	6	7	8	9	10
+	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
1	0	0	18	0	0	20	10	8	0	0
2	0	0	0	18	0	0	0	0	0	0
3	0	0	0	1	3	12	0	5	0	1
4	16	0	0	0	5	0	0	0	2	0
5	0	15	0	0	0	3	0	0	0	0
6	0	0	0	0	11	0	0	0	20	0
7	0	5	0	0	0	0	0	0	14	9
8	0	0	13	0	0	0	0	0	0	0
9	0	1	0	18	5	12	0	12	0	0
10	9	1	20	11	18	4	3	0	0	0

Calcule a distancia minima a cada um dos demais nodos

1	2	3	4	5
	0			
6	7	8	9	10

1

Caminhos mínimos: algoritmo de Floyd-Warshall

Este algoritmo calcula os caminhos mínimos de todas as origens para todos os destinos em um grafo valorado. O algoritmo admite grafos com laços e são também calculados os valores de caminhos começando e terminando no mesmo vértice (usando laços ou não). A entrada para o algoritmo é a matriz K_0 , obtida a partir da matriz de custos do grafo, efetuando-se a mudança de trocar ZERO por INFINITO.

A cada passo do algoritmo, uma nova matriz $K_n (n = 1, 2, \dots)$ é calculada. A última matriz é a resposta dos caminhos mínimos.

Note que a complexidade deste algoritmo é $O(n^3)$. Porque ?

- 1: Algoritmo FLOYD-WARSHALL
- 2: $C[n,n] \leftarrow C$ {é a matriz de custos}
- 3: Gerar K , a partir de C (trocando 0 por ∞)
- 4: para Q de 1 até N
- 5: para S de 1 até N
- 6: para T de 1 até N
- 7: se $K[S,Q] + K[Q,T] < K[S,T]$
- 8: $K[S,T] \leftarrow K[S,Q] + K[Q,T]$
- 9: fim{se}
- 10: fim{para}
- 11: fim{para}
- 12: fim{para}
- 13: fim algoritmo

Note que os comandos 7,8 e 9 podem ser substituídos por

$K[S,T] \leftarrow K[S,T] \text{ min } K[S,Q] + K[Q,T]$

Exemplo comentado passo-a-passo

Seja o grafo com ADJ =

5	0	1
1	1	3
8	0	0

que dá origem à matriz K_0 valendo

5	∞	1
1	1	3
8	∞	∞

Para $Q = 0, S = 0$ que dá origem

$Q[0,0] \leftarrow Q[0,0]$	[	$Q[0,0] + Q[0,0]$
$Q[0,1] \leftarrow Q[0,1]$	[	$Q[0,0] + Q[0,1]$
$Q[0,2] \leftarrow Q[0,2]$	[	$Q[0,0] + Q[0,2]$

Para $Q = 0, S = 1$ que dá origem

$Q[1,0] \leftarrow Q[1,0]$	[	$Q[1,0] + Q[0,0]$
$Q[1,1] \leftarrow Q[1,1]$	[	$Q[1,0] + Q[0,1]$
$Q[1,2] \leftarrow Q[1,2]$	[	$Q[1,0] + Q[0,2]$

Para $Q = 0, S = 2$ que dá origem

$Q[2,0] \leftarrow Q[2,0]$	[	$Q[2,0] + Q[0,0]$
$Q[2,1] \leftarrow Q[2,1]$	[	$Q[2,0] + Q[0,1]$
$Q[2,2] \leftarrow Q[2,2]$	[	$Q[2,0] + Q[0,2]$

K_1		
5	∞	1
1	1	3
8	∞	9

K_2		
5	∞	1
1	1	3
8	∞	9

K_3		
5	∞	1
1	1	3
8	∞	9

Heurística

Para a primeira matriz, riscar a primeira linha e a primeira coluna. Valores que estão sob o risco são escritos como estão. Os demais devem ser comparados com a soma dos elementos correspondentes riscados, e o menor dos dois deve ser escrito. Para a segunda matriz riscar a segunda linha e coluna e assim por diante.

Exemplo

Seja a seguinte matriz C

8	6	0	6	0	2
0	0	5	0	4	3
0	0	0	0	1	5
0	7	3	0	3	4
1	1	0	7	0	8
4	2	0	2	0	0

que gera a seguinte K_0

8	6	∞	6	∞	2
∞	∞	5	∞	4	3
∞	∞	∞	∞	1	5
∞	7	3	∞	3	4
1	1	∞	7	∞	8
4	2	∞	2	∞	∞

K_1					
8	6	∞	6	∞	2
∞	∞	5	∞	4	3
∞	∞	∞	∞	1	5
∞	7	3	∞	3	4
1	1	∞	7	∞	3
4	2	∞	2	∞	6

K_2					
8	6	11	6	10	2
∞	∞	5	∞	4	3
∞	∞	∞	∞	1	5
∞	7	3	∞	3	4
1	1	6	7	5	3
4	2	7	2	6	5

K_3					
8	6	11	6	10	2
∞	∞	5	∞	4	3
∞	∞	∞	∞	1	5
∞	7	3	∞	3	4
1	1	6	7	5	3
4	2	7	2	6	5

K_4					
8	6	9	6	9	2
∞	∞	5	∞	4	3
∞	∞	∞	∞	1	5
∞	7	3	∞	3	4
1	1	6	7	5	3
4	2	5	2	5	5

K_5					
8	6	9	6	9	2
5	5	5	11	4	3
2	2	7	8	1	4
4	4	3	10	3	4
1	1	6	7	5	3
4	2	5	2	5	5

K_6					
6	4	7	4	7	2
5	5	5	5	4	3
2	2	7	6	1	4
4	4	3	6	3	4
1	1	6	5	5	3
4	2	5	2	5	5



Para você fazer

Dado um grafo direcionado, com lacos, através de sua matriz de adjacência usando o algoritmo Floyd/Warshall,

	1	2	3	4	5	6	7
	+-----						
1	0	0	0	0	6	4	0
2	4	0	3	0	0	0	0
3	5	2	0	1	0	0	0
4	1	0	6	1	2	0	7
5	8	0	0	0	0	4	0
6	3	8	2	0	0	0	7
7	0	0	5	0	3	0	0

ache a matriz de caminhos mínimos e responda quais os valores do

Caminho mínimo do nodo 7 ao 2	
Caminho mínimo do nodo 7 ao 5	
Caminho mínimo do nodo 7 ao 5	
Caminho mínimo do nodo 3 ao 3	
Caminho mínimo do nodo 3 ao 3	

Observação: contagens começam em 1 e não em zero. Represente ∞ por 9999 na resposta.



Caminhos mínimos: Floyd modificado

Bibliografia: Introdução à Teoria dos Grafos (Rabuske, UFSC)

Suponha um grafo DIRIGIDO, representado por sua matriz de custos, com sua diagonal principal zerada (sem laços). Esta matriz deve ser assim interpretada: o valor em $C[i][j]$ representa o custo da ligação de i para j . Quando não há ligação, isto deve ser representado por ∞ . A matriz original é K_0 . A seguir, são calculadas as matrizes $K_1, K_2, \dots, K_n$, onde n é a ordem do grafo. Ao mesmo tempo são calculadas as matrizes $R_0, R_1, \dots, R_n$. K_n é a matriz de menores custos de ligação, e R_n é a matriz que indica as rotas.

O algoritmo é

```

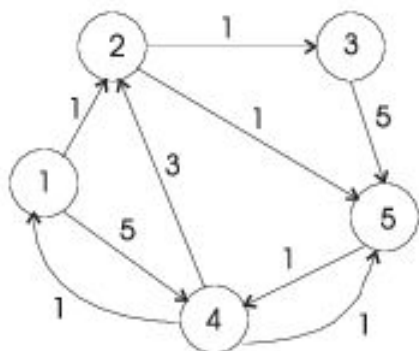
1: Algoritmo FLOYD-MOD
2: C[n,n] ← matriz de custos.
3: R[n,n] ← 0
4: Gerar K, a partir de C (troque 0 por infinito e ponha 0 na DP)
5: para Q de 1 até N faça
6:   para S de 1 até N faça
7:     para T de 1 até N faça
8:       se K[S][Q] + K[Q][T] < K[S][T] então
9:         K[S][T] ← K[S][Q] + K[Q][T]
10:      se R[S][Q] = 0 então
11:        R[S][T] ← Q
12:      senão
13:        R[S][T] ← R[S][Q]
14:      fim se
15:    fim se
16:  fim para
17: fim para
18: fim algoritmo
19: fim algoritmo
    
```

Ao final, as posições de R que tiverem ZERO devem receber o número da coluna de R.

A origem das contagens é 1 e não zero

Exemplo

Considere, por exemplo, o seguinte grafo



Seja por exemplo, o grafo acima com 5 vértices e 9 arestas. O grafo é dirigido, valorado e tem zeros na diagonal principal.

Note que inexistência de arco está representada por infinito (=99)

```

K0=0---1--99---5--99  K1=0  1  99  5  99  R1=0  0  0  0  0  0
99| 0  1  99  1  99  0  1  99  1  0  0  0  0  0
99| 99  0  99  5  99  99  0  99  5  0  0  0  0  0
|1| 3  99  0  1  1  (2) 99  0  1  0  (1) 0  0  0
99| 99  99  1  0  99  99  99  1  0  0  0  0  0  0

K1=0  |1| 99  5  99  K2=0  1  (2) 5  (2)  R2=0  0  (2) 0  (2)
--99--0---1--99---1  99  0  1  99  1  0  0  0  0  0  0
99  99| 0  99  5  99  99  0  99  5  0  0  0  0  0
1  |2| 99  0  1  1  2  (3) 0  1  0  {1} (1) 0  0  0
99  99| 99  1  0  99  99  99  1  0  0  0  0  0  0

K2=0  1  |2| 5  2  K3=0  1  2  5  2  R3=0  0  2  0  2
99  0  |1| 99  1  99  0  1  99  1  0  0  0  0  0
--99--99--0--99---5  99  99  0  99  5  0  0  0  0  0
1  2  |3| 0  1  1  2  3  0  1  0  1  1  0  0  0
99  99  99| 1  0  99  99  99  1  0  0  0  0  0  0  0

K3=0  1  2  |5| 2  K4=0  1  2  5  2  R4=0  0  2  0  2
99  0  1  99| 1  99  0  1  99  1  0  0  0  0  0
99  99  0  99| 5  99  99  0  99  5  0  0  0  0  0
---1---2---3---0---1  1  2  3  0  1  0  1  1  0  0  0
99  99  99  |1| 0  (2) (3) (4) 1  0  (4) (4) (4) 0  0

K4=0  1  2  5  |2|  K5=0  1  2  (3) 2  R5=0  0  2  (2) {2}
    
```

```

99  0  1  99  |1|  (3) 0  1  (2) 1  (5) 0  0  (5) 0
99  99  0  99  |5|  (7) (8) 0  (6) 5  (5) (5) 0  (5) 0
1  2  3  0  |1|  1  2  3  0  1  0  1  1  1  0  0
---2---3---4---1---0| 2  3  4  1  0  4  4  4  0  0
    
```

```

Note que em Rf,      Kf=0  1  2  3  2  Rf=1  2  2  2  2
valores ZERO são      3  0  1  2  1  5  2  3  5  5
substituídos pelo      7  8  0  6  5  5  5  3  5  5
número da coluna      1  2  3  0  1  1  1  1  4  5
                        2  3  4  1  0  4  4  4  4  5
    
```

Considere, por exemplo, a ida do nodo 3 para o nodo 1. Segundo X essa ida custa 7, e verificando a rota (em R), temos

- $R[3,1]=5$, logo vamos de 3 para 5, pagando 5
- $R[5,1]=4$, logo vamos de 5 para 4, pagando mais 1 ($5+1=6$)
- $R[4,1]=1$, logo chegamos em 1. Fomos de 4 p/ 1 pagando mais 1 ($5+1+1=7$)

Análise a ida de 4 para 2: Custo 2, Rota: 4-1-2

Idem para ida de 1 para 4: Custo 3, Rota: 2-5-4

Para você fazer

Dada a matriz C a seguir, calcule a matriz K de custos mínimos e a matriz R de roteamento, segundo Floyd modificado.

C =

	1	2	3	4	5	6	7
1	0	6	99	8	4	99	1
2	99	0	1	99	4	99	3
3	99	99	0	99	99	2	99
4	2	5	7	0	6	8	99
5	99	4	99	99	0	99	99
6	99	99	2	3	7	0	99
7	99	99	3	5	99	99	0

Responda agora

K[1][5]	R[4][7]	K[4][4]	R[5][6]	K[3][1]	R[4][4]



Um hipotético metrô em Curitiba Este exercício nasceu estudando o mapa do metrô de uma grande cidade. Lá são 14 linhas completamente interligadas, em uma cidade não muito maior do que a nossa. Para deixar o exercício mais divertido, transplantei o problema para cá, e colocaram-se 11 linhas de metrô em Curitiba. O objetivo do exercício é vivenciar uma aplicação prática do algoritmo de Floyd-Warshall que é usado para calcular o menor caminho entre dois pontos em uma rede interligada. O algoritmo é simples, embora demande muita máquina já que tem custo $O(n^3)$ onde n é o número de estações de metrô. Em nosso exemplo este número é de 115 estações.

Além de achar o menor caminho, você deve também calcular qual a rota a percorrer para conseguir obter este menor caminho. O algoritmo é apresentado a seguir.

```
1: Algoritmo FLOYD-MOD
2: C[n,n] ← matriz de custos.
3: R[n,n] ← 0
4: Gerar K, a partir de C (troque 0 por infinito e ponha 0 na DP)
5: para Q de 1 até N faça
6:   para S de 1 até N faça
7:     para T de 1 até N faça
8:       se K[S][Q] + K[Q][T] < K[S][T] então
9:         K[S][T] ← K[S][Q] + K[Q][T]
10:      se R[S][Q] = 0 então
11:        R[S][T] ← Q
12:      senão
13:        R[S][T] ← R[S][Q]
14:      fim se
15:    fim se
16:  fim para
17: fim para
18: fim para
19: fim algoritmo
```

Ao final, as posições de R que tiverem ZERO devem receber o número da coluna de R.

Considere as seguintes linhas de metrô

Linha 1: Santa Cândida-Pinheirinho (azulão) santa cândida; joão gulin; canadá; ludovico geronasso; holanda; cabral; são lucas; alto glória; passeio público; esquina marechais; shopping estação; shopping curitiba; coronel dulcício; praça japão; capelinha; pedro macedo; portão; itatiaia; pedro zagonel; capão raso; churchill; pinheirinho. Total de estações: 22

Linha 2: UFPR-UP-UTFPR-PUC/Pr -Aeroporto (verdão) universidade positivo; campo comprido; mossunguê; shopping barigui; campina siqueira; francisco rocha; praça japão; pedro macedo; portão; wal mart; uberaba; portal sãojósé; cuia grande; aeroporto. Total de estações: 15

Linha 3: São Lourenço-Pinhais (rosa) são lourenço; bar vitor; brasílio mouro; bosque papa; centro cívico; passeio público; ubaldino amaral; praça nações; colégio militar; pinheirão; pinhais; ZANAORIA PINHAIS. Total de estações: 12

Linha 4: São José-Santa Felicidade (cinza) são josé; boqueirão; carmo; joaquim torrea; passarela; hauer; shopping cidade; tre; engenheiros rebouças; shopping estação; esquina marechais; ruínas sãofrancisco; telepar; hospital graças; cascatinha; cruz; madalosso; santa felicidade. Total de estações: 18

Linha 5: circular centro (verde claro) rua teffe; telepar; badep; shopping curitiba; engenheiros rebouças; vila capanema; largo lazzarotto; ubaldino amaral; alto glória; centro cívico e retorna novamente à estação da rua teffe. Há aqui 10 estações.

Linha 6: Fazendinha-Bairro Alto (roxo) fazendinha; algacyr mader; potiguara; carlos klemtz; portão; ulisses vieira; francisco rocha; badep; esquina marechais; passeio público; hugo lange; fagundes varela; hospital vita; bairro alto; praça liberdade; avenida maringá; ZIMBRO. Total de estações: 17

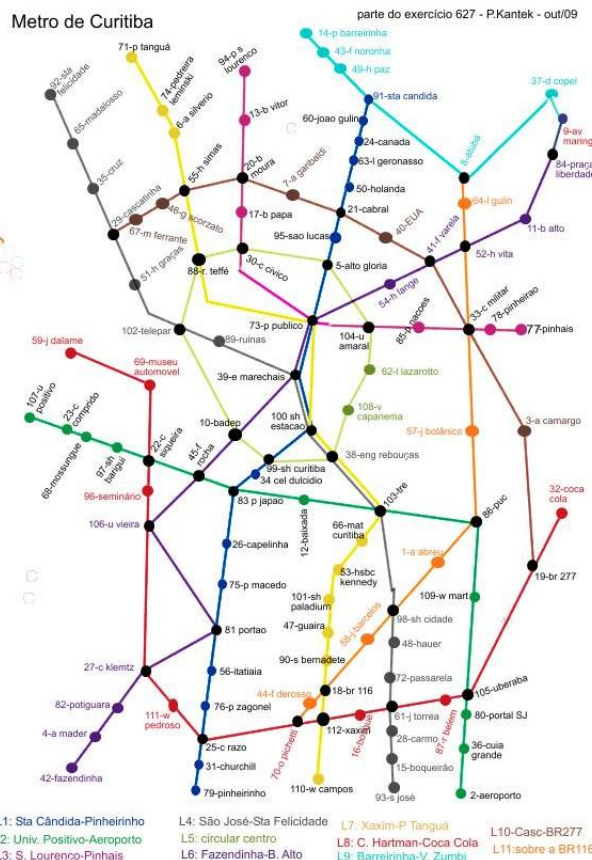
Linha 7: Xaxim-Parque Tanguá (amarelo) waldemar campos; xaxim; br116; santa bernadete; guaira; shopping paladium; hsbk kennedy; maternidade curitiba; tre; engenheiros rebouças; shopping estação; passeio público; rua teffe; hugo simas; amaury silvério; pedreira leminski; parque tanguá. Total de estações: 17

Linha 8: Candido Hartmann-Coca Cola (vermelho) joão dallami; museu automóvel; campina siqueira; seminário; ulisses vieira; carlos klemtz; waldemiro pedroso; capão raso; omar pichetti; xaxim; bosque maack; joaquim torrea; rio belém; uberaba; br277; coca cola. Total de estações: 16

Linha 9: Barreirinha-Vila Zumbi (azul claro) barreirinha; fernando noronha; hildo paz; santa cândida; atuba; depósito copel; avenida maringá. Total de estações: 7

Linha 10: Cascatinha-BR 277 (marrom) cascatinha; mbá ferrante; gardênio scorzato; hugo simas; brasílio mouro; anita garibaldi; cabral; estados unidos; fagundes varela; colégio militar; afonso camargo; br277; ZUZU AMIGA. Total de estações: 13

Linha 11: Sobre a BR116 (laranja) omar pichetti; francisco derosso; br116; joão barcelos; shopping cidade; abílio abreu; puc; jardim botânico; colégio militar; hospital vita; luiza gulin; atuba. Total de estações: 12



Como Fazer Note que somando as estações de cada linha, obtem-se 159, quando na realidade existem apenas 115 estações. A explicação é que muitas estações são compartilhadas entre várias linhas, o que caracteriza a formação de uma rede de transporte, onde o viajante pode trocar de linhas dentro da estação sem ter que pagar nova passagem. A estação que mais tem linhas é a do passeio público, onde passam as linhas 1, 3, 6 e 7.

1. Inicialmente, tente obter ou fazer um desenho da distribuição das linhas no espaço urbano. Este desenho serve de aquecimento e um bom lugar “por onde começar”.
2. Há que se ter uma lista de estações (sem duplicidades). Esta lista pode ser obtida das paradas de cada linha. Ela precisa ter alguma ordenação única, sugere-se a alfabética a fim de coincidir com as estações pedidas nesta folha. Este número de estações tem que coincidir com aquele dado no problema.
3. Agora a parte mais difícil: Construir a matriz de adjacência do problema (quadrada de ordem igual ao número de estações). Colocar 1 nas ligações entre estações. (Por exemplo, se a linha 1 conecta a praça do japão com a capelinha, e se a praça do japão é a estação 83 enquanto a capelinha é a estação 27, e se a matriz de adjacência por hipótese é MADJ, deve-se fazer $MADJ[83;27] \leftarrow 1$ e também $MADJ[27;83] \leftarrow 1$).
4. Tendo construído (e conferido) esta matriz é hora de entregá-la ao algoritmo de Floyd Warshall, lembrando de construir também a matriz de roteamentos.
5. Com estas duas matrizes é possível responder e acertar as 3 perguntas aqui colocadas.

Para você fazer

1. Quantos trechos de metrô compõe o menor caminho entre as estações 101-shopping paladium e 68-mossunguê?
2. Qual o número da sexta estação na rota mínima entre a estação 43-fernando noronha e 95-são lucas? (Obs: 1. a origem é a estação número 1; 2. Havendo mais de um caminho mínimo, você deve simular o algoritmo e responder o que ele responderá.)
3. Montada a matriz de caminhos mínimos, deve-se calcular a somatória obtida na diagonal secundária (a que vai de 1, n até n, 1 em uma matriz de n linhas por n colunas).

1	2	3



Cálculo de Fluxos - método: Ford-Fulkerson, algoritmo: Edmonds-Karp

Bibliografia: Algoritmos, Cormen et alli, pag. 509 e seguintes. Note que as contagens iniciam em 1.

Imaginando um grafo orientado como sendo uma rede de transporte de água (ou qualquer outra coisa) e supondo que o valor de cada aresta representa a capacidade máxima do cano que liga dois pontos, e supondo ainda que um dos vértices representa a caixa d'água e outro representa o local onde a água vai ser usada, cabe a pergunta: *QUAL A VAZÃO MÁXIMA DO SISTEMA ?*

O mesmo algoritmo vale para sistemas produtor-consumidor: produtos sendo transportados, pacotes de bits sendo enviados, pessoas saindo de um avião etc.

Nomenclatura Fluxo em rede $G = (W, E)$ é um grafo orientado em que cada aresta (u, w) pertencente a E , tem capacidade não negativa $c(u, w) \geq 0$. Distinguimos dois vértices "in" e "fi" que são a origem e o destino. Todos os vértices z residem em algum caminho, isto é: $in \rightarrow z \rightarrow fi$.

A vazão do sistema (V) é uma função de valor real indica a capacidade do sistema de transportar o objeto de estudo.

Problema da Vazão máxima

O problema de VAZÃO MÁXIMA é encontrar o maior valor para a VAZÃO.

Exemplo de um fluxo: (retirado da referência acima, pág. 522)

	1	2	3	4	5	6
1	0	16	13	0	0	0
2	0	0	10	12	0	0
3	0	4	0	0	14	0
4	0	0	9	0	0	20
5	0	0	0	7	0	4
6	0	0	0	0	0	0

sendo in=1 e fi=6

Método Ford-Fulkerson O método Ford-Fulkerson para calcular a vazão máxima é:

1. Fazer vazão V igual a zero
2. Enquanto houver caminho entre in e fi
3. Localizar um caminho entre in e fi .
4. Achar a capacidade mínima CM deste caminho.
5. Adicionar CM a V
6. Subtrair CM das arestas que pertencem ao caminho escolhido
7. A vazão máxima está em V

O método acima não diz nada a respeito de como escolher o caminho. O resultado final não se altera, mas a complexidade do problema sim.

Algoritmo Edmonds-Karp Para resolver esta indefinição, Edmonds e Karp desenvolveram uma estratégia para escolher o caminho. Trata-se de considerar cada aresta existente no grafo com valor unitário e escolher o caminho mínimo entre in e fi . Note que o valor deste caminho não tem nada a ver com o fluxo, é uma simples contagem de quantidade de arestas.

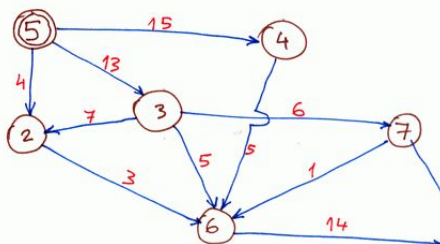
Usando o algoritmo de Edmonds-Karp, a complexidade passa a ser $O(W.E^2)$ onde W =número de vértices e E =número de arestas.

Para resolver este exercício você pode escolher qualquer caminho.

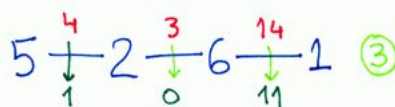
Acompanhe no exemplo. Seja o grafo

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	3	0
3	0	7	0	0	0	5	6
4	0	0	0	0	0	5	0
5	0	4	13	15	0	0	0
6	14	0	0	0	0	0	0
7	5	0	0	0	0	1	0

com in=5 e fi=1



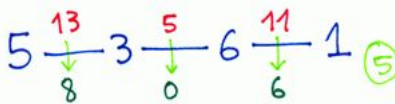
O primeiro caminho pode ser 5-2-6-1 com mínimo=3, $V=3$



e M fica

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	(0)	0
3	0	7	0	0	0	5	6
4	0	0	0	0	0	5	0
5	0	(1)	13	15	0	0	0
6	(11)	0	0	0	0	0	0
7	5	0	0	0	0	1	0

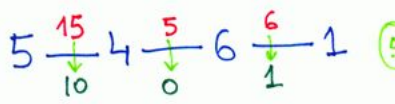
Outro caminho pode ser 5-3-6-1, com mínimo=5, logo $V=3+5$



e M fica

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	7	0	0	0	(0)	6
4	0	0	0	0	0	5	0
5	0	1	(8)	15	0	0	0
6	(6)	0	0	0	0	0	0
7	5	0	0	0	0	1	0

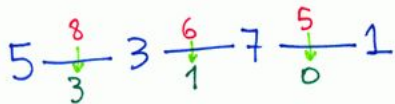
Outro: 5-4-6-1, com mínimo=5, $V=3+5+5$



e M fica

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	7	0	0	0	0	6
4	0	0	0	0	0	(0)	0
5	0	1	8	(10)	0	0	0
6	(1)	0	0	0	0	0	0
7	5	0	0	0	0	1	0

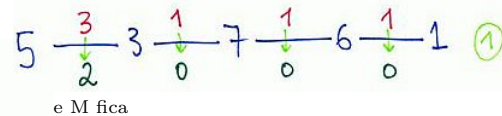
Outro: 5-3-7-1, com mínimo=5, $V=3+5+5+5$



e M fica

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	7	0	0	0	0	(1)
4	0	0	0	0	0	0	0
5	0	1	(3)	10	0	0	0
6	1	0	0	0	0	0	0
7	(0)	0	0	0	0	1	0

Outro: 5-3-7-6-1, com mínimo=1 e $V=3+5+5+5+1$



e M fica

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	7	0	0	0	0	(0)
4	0	0	0	0	0	0	0
5	0	1	(2)	10	0	0	0
6	(0)	0	0	0	0	0	0
7	0	0	0	0	0	(0)	0

E como não há mais caminhos, V está calculado e vale 19.

Para você fazer

1. Ache a VAZÃO MÁXIMA para este fluxo em rede com in= 3 e fi= 6

	1	2	3	4	5	6	7
1	0	3	0	0	4	0	0
2	0	0	0	0	0	17	0
3	3	0	0	18	0	0	12
4	0	0	0	0	8	0	0
5	0	0	0	0	0	4	6
6	0	0	0	0	0	0	0
7	0	6	0	10	0	0	0

A vazão máxima é _____.

2. Ache a VAZÃO MÁXIMA para este fluxo em rede com in= 6 e fi= 7

	1	2	3	4	5	6	7
1	0	3	0	10	0	0	18
2	0	0	0	0	0	0	5
3	9	0	0	9	0	0	0
4	0	0	0	0	0	0	19
5	7	1	0	0	0	0	0
6	0	0	1	0	1	0	0
7	0	0	0	0	0	0	0

A vazão máxima é _____.

3. Ache a VAZÃO MÁXIMA para este fluxo em rede com in= 3 e fi= 5

	1	2	3	4	5	6	7
1	0	9	0	1	0	0	6
2	9	0	0	0	2	0	0
3	12	0	0	0	0	7	0
4	0	9	0	0	17	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	4
7	0	0	0	0	11	0	0

A vazão máxima é _____.



Ordenação topológica usando grafos

Tem-se aqui uma aplicação sobre disciplinas e pré-requisitos (Sort Topológico).

Para este exercício considere um curso de informática que contém 14 disciplinas, assim distribuídas:

1. ALB Algebra Booleana
2. LAI Laboratório de informática
3. VCP Visual C++
4. VBA Visual Basic
5. BDA Banco de Dados
6. EDA Estrutura de Dados
7. COM Compiladores
8. CAN Cálculo Numérico
9. DIR Direito
10. CAL Cálculo
11. GAN Geometria Analítica
12. LPO Língua Portuguesa
13. ACU Arte Culinária
14. ADR Arte Dramática

Existe uma tabela de pré-requisitos, como por exemplo:

1. VBA é pré-requisito de LAI	no grafo, faça VBA ---->--- LAI
2. VBA é pré-requisito de CAL	...
3. VBA é pré-requisito de ACU	
4. BDA é pré-requisito de LAI	
5. BDA é pré-requisito de LPO	
6. BDA é pré-requisito de ADR	
7. COM é pré-requisito de ALB	
8. CAN é pré-requisito de LAI	
9. DIR é pré-requisito de LAI	
10. CAL é pré-requisito de VCP	
11. CAL é pré-requisito de ACU	
12. CAL é pré-requisito de ADR	
13. GAN é pré-requisito de ALB	
14. LPO é pré-requisito de EDA	
15. LPO é pré-requisito de COM	
16. LPO é pré-requisito de DIR	
17. LPO é pré-requisito de ADR	
18. ACU é pré-requisito de ALB	
19. ADR é pré-requisito de LAI	
20. ADR é pré-requisito de COM	

O objetivo do exercício deve ser construir a sequência de disciplinas que deverão ser cursadas por um aluno, sujeito às seguintes regras:

- Uma disciplina só pode ser cursada DEPOIS que seus pré-requisitos o forem.
- Em caso de que mais de uma disciplina possa ser feita, deve-se escolher aquela que tiver o menor número. Note que na vida real, esta regra 2 não é obrigatória. Aqui sim, para podermos conferir com o gabarito do exercício.

Para resolver este exercício, você deve

1. Construir um grafo dirigido com base nas informações acima, onde os vértices são as disciplinas e as arestas são as relações de pré-requisitos.
2. Construir a matriz de adjacência para este grafo

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4-	0	1	0	0	0	0	0	0	0	0	1	0	0	1
5-	0	1	0	0	0	0	0	0	0	0	0	0	1	0
6-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7-	1	0	0	0	0	0	0	0	0	0	0	0	0	0
8-	0	1	0	0	0	0	0	0	0	0	0	0	0	0
9-	0	1	0	0	0	0	0	0	0	0	0	0	0	0
10-	0	0	1	0	0	0	0	0	0	0	0	0	1	1
11-	1	0	0	0	0	0	0	0	0	0	0	0	0	0
12-	0	0	0	0	0	0	1	1	0	1	0	0	0	1
13-	1	0	0	0	0	0	0	0	0	0	0	0	0	0
14-	0	1	0	0	0	0	1	0	0	0	0	0	0	0

3. Totalizar a matriz na vertical, obtendo o grau de incidência de cada vértice.

1	2	3	4	5	6	7	8	9	10	11	12	13	14

4. A disciplina a cursar será o primeiro vértice com grau de incidência = 0.

Neste caso VBA (disciplina 4)

5. Cursada uma disciplina, todas as que a tem como pré-requisito devem ter seu grau de incidência diminuído de UMA unidade. A disciplina escolhida deve receber INFINITO.

Fica:

1=3 2=(4) 3=1 4=(999) 5=0 6=1 7=2 8=0
 9=1 10=(0) 11=0 12=1 13=(1) 14=3

6. Retornar a 3, até cursar todas as disciplinas.

Neste caso BDA (disciplina 5) e fica
 3 4 1 999 999 1 2 0 1 0 0 (0) 1 (2)

e assim por diante...

7. A sequência final é:

disciplinas: 4 5 8 10 3 11 12 6 9 13 14 2 7 1, ou
 VBA, BDA, CAN, CAL, VCP, GAN, LPO,
 EDA, DIR, ACU, ADR, LAI, COM, ALB

Para você fazer

Instruções: Utilize a tabela de disciplinas no início desta folha. 1. Responda com a sigla das disciplinas (e não os números).

1-LAI e pré-requisito de VBA	11-DIR e pré-requisito de CAL
2-VCP e pré-requisito de ALB	12-GAN e pré-requisito de BDA
3-VCP e pré-requisito de BDA	13-GAN e pré-requisito de LPO
4-VCP e pré-requisito de CAL	14-LPO e pré-requisito de ALB
5-VCP e pré-requisito de GAN	15-ACU e pré-requisito de ALB
6-VBA e pré-requisito de GAN	16-ACU e pré-requisito de COM
7-EDA e pré-requisito de LAI	17-ACU e pré-requisito de CAN
8-EDA e pré-requisito de LPO	18-ADR e pré-requisito de LAI
9-CAN e pré-requisito de VCP	19-ADR e pré-requisito de VBA
10-CAN e pré-requisito de CAL	20-ADR e pré-requisito de CAL

Resolva esta instância e responda, qual o código (3 letras) da disciplina

número 5	número 8	número 13	número 14

2. Responda com a sigla das disciplinas (e não os números).

1-LAI e pré-requisito de VBA	11-EDA e pré-requisito de CAL
2-LAI e pré-requisito de COM	12-COM e pré-requisito de VBA
3-VCP e pré-requisito de VBA	13-COM e pré-requisito de CAN
4-VCP e pré-requisito de CAN	14-COM e pré-requisito de LPO
5-VCP e pré-requisito de ACU	15-DIR e pré-requisito de CAL
6-VCP e pré-requisito de ADR	16-DIR e pré-requisito de GAN
7-BDA e pré-requisito de EDA	17-CAL e pré-requisito de ALB
8-BDA e pré-requisito de CAL	18-CAL e pré-requisito de CAN
9-BDA e pré-requisito de ADR	19-LPO e pré-requisito de CAL
10-EDA e pré-requisito de CAN	20-ADR e pré-requisito de GAN

Resolva esta instância e responda, qual o código (3 letras) da disciplina

número 1	número 3	número 12	número 14

Em resumo:

a→b significa a é pré-requisito de b

  
 - 1 - /

Algoritmo de Kruskal: árvore de cobertura mínima

Este problema foi estudado pela primeira vez em 1926 pelo engenheiro Boruvka, quando recebeu a incumbência de eletrificar a Morávia (sudeste da antiga Tchecoslováquia).

Para entender o problema, suponha que você tem uma usina hidrelétrica e um monte de casas espalhadas numa zona rural grande. O objetivo é levar energia elétrica a todas as casas. Ligando todas as casas à usina (ligação em estrela), haverá um grande desperdício de fios e postes.

Outra possibilidade é uma ligação em anel, pela qual são ligadas as casas que estão no perímetro externo da zona, mas permanece a indefinição de como ligar as casas que estão dentro desse perímetro.

Para uma solução ótima (e matematicamente quando se usa o adjetivo ótimo, está se dizendo que é a melhor solução possível), é preciso matematizar o problema, descrevendo-o em termos exatos e passíveis de tratamento matemático. Neste problema, tais termos são o grafo e a matriz de custos. O resultado do problema da ligação é conhecido como árvore de cobertura mínima que nada mais é do que um sub-grafo que conecta todos os vértices e apresenta o menor custo possível.

Dado um grafo não dirigido, cujas arestas comportam custos (distâncias, tempos, volumes, atrasos, postos de pedágio...), este algoritmo gera uma árvore (não há ciclos) que conecta todos os vértices pelo custo mínimo.

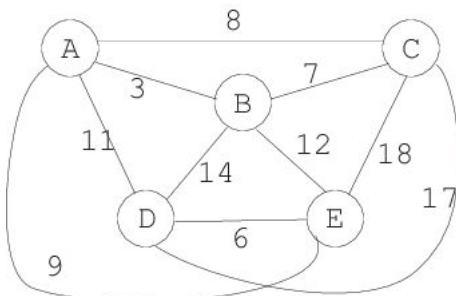
É uma aplicação de "algoritmo guloso"

Suponhamos que o grafo é apresentado através da (diagonal superior da) sua matriz de custos. Por exemplo, seja um grafo de 5 vértices:

Cuja matriz de custos é

	A	B	C	D	E
A	-	3	8	11	9
B	-	-	7	14	12
C	-	-	-	17	18
D	-	-	-	-	6
E	-	-	-	-	-

e cuja figura seja



A etapa 1 do algoritmo é a criação de uma lista de arestas e seus custos

A,B,3 A,C,8 A,D,11 A,E,9 B,C,7
 B,D,14 B,E,12 C,D,17 C,E,18 D,E,6

A etapa 2 é ordenar esta lista em ordem crescente de custo e fica:

A,B,3 D,E,6 B,C,7 A,C,8 A,E,9
 A,D,11 B,E,12 B,D,14 C,D,17 C,E,18

Etapa 3: Construir um conjunto para cada vértice do grafo

- 1: Inicializar a árvore geradora mínima como vazia
- 2: enquanto houver mais de um conjunto
- 3: Selecionar a aresta de menor custo
- 4: se as extremidades estão em conjuntos diferentes
- 5: junte os conjuntos e acrescente a aresta na árvore
- 6: senão
- 7: despreze esta aresta
- 8: fim{se}
- 9: fim{enquanto}

Etapa 4: o conjunto (único) contém todos os vértices e a árvore está OK!

Acompanhe o exemplo

- Conjuntos: {A}, {B}, {C}, {D}, {E}, Arvore=[] (vazia)
- Menor: A,B,3. Como A e B estão em conjuntos diferentes, junte-os e fica: {A, B}, {C}, {D}, {E}. Arvore=[(AB)].
- Menor: D,E,6. Como D e E... Fica: {A,B}, {C}, {D,E}. Arvore=[(AB), (DE)]
- menor: B,C,7. Como B e C... Fica {A,B,C}, {D,E}. Arvore=[(AB), (DE), (BC)]
- menor: A,C,8. Como A e C já são do mesmo conjunto, a aresta é desprezada
- menor: A,E,9. Como A e E estão em dois conjuntos, fica:

- {A,B,C,D,E}. Arvore=[(AB), (DE), (BC), (AE)] com um custo de 25 e acabou o algoritmo.

Outro exemplo

Acompanhe este exemplo: Seja o grafo 8 x 8:

	1	2	3	4	5	6	7	8
1	0	7	22	38	64	38	2	37
2	0	0	18	46	44	11	17	52
3	0	0	0	56	43	49	33	59
4	0	0	0	0	44	43	7	43
5	0	0	0	0	0	26	36	45
6	0	0	0	0	0	0	60	47
7	0	0	0	0	0	0	0	44
8	0	0	0	0	0	0	0	0

Que gera a seguinte lista de arestas (já ordenada):

CS	OR	DE	CS	OR	DE	CS	OR	DE	CS	OR	DE
==	==	==	==	==	==	==	==	==	==	==	==
2	1	7	26	5	6	43	4	6**	47	6	8
7	1	2**	33	3	7	43	4	8**	49	3	6
7	4	7**	36	5	7	44	2	5**	52	2	8
11	2	6	37	1	8	44	4	5**	56	3	4
17	2	7	38	1	4**	44	7	8**	59	3	8
18	2	3	38	1	6**	45	5	8	60	6	7
22	1	3	43	3	5**	46	2	4	64	1	5

Observação importante: Note as arestas marcadas com **. Elas estão empacadas. Observe como, neste caso o algoritmo preserva a ordem de construção da tabela.

Esta lista quando aplicada no algoritmo gera a seguinte árvore mínima:

1,7 1,2 4,7 2,6 2,3 5,6 1,8;
 com um custo mínimo de 108

Para você fazer

1.

	1	2	3	4	5	6	7	8	9	10
1	0	35	44	51	66	61	43	16	33	17
2	0	0	57	50	89	63	17	92	23	5
3	0	0	0	45	84	58	85	39	36	55
4	0	0	0	0	5	82	56	4	87	75
5	0	0	0	0	0	29	71	43	59	29
6	0	0	0	0	0	0	61	9	6	82
7	0	0	0	0	0	0	0	48	63	33
8	0	0	0	0	0	0	0	0	41	30
9	0	0	0	0	0	0	0	0	0	98
10	0	0	0	0	0	0	0	0	0	0

Pede-se a ÚLTIMA aresta que forma a árvore mínima:
 de _____ a _____,

Pede-se também o CUSTO TOTAL DA ÁRVORE DE COBERTURA MÍNIMA: _____

2.

	1	2	3	4	5	6	7	8	9	10
1	0	93	11	81	1	45	74	85	68	20
2	0	0	73	65	32	66	10	14	39	8
3	0	0	0	6	45	11	38	8	62	33
4	0	0	0	0	19	52	26	67	97	78
5	0	0	0	0	0	8	34	13	29	3
6	0	0	0	0	0	0	95	81	6	89
7	0	0	0	0	0	0	0	21	94	49
8	0	0	0	0	0	0	0	0	61	93
9	0	0	0	0	0	0	0	0	0	30
10	0	0	0	0	0	0	0	0	0	0

Pede-se a ÚLTIMA aresta que forma a árvore mínima:
 de _____ a _____,

Pede-se também o CUSTO TOTAL DA ÁRVORE DE COBERTURA MÍNIMA: _____



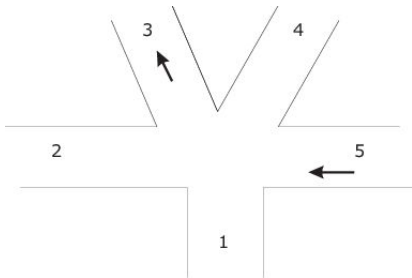
Coloração de grafos

Este é um dos mais famosos problemas NP. Isto significa que sua complexidade é exponencial e que a única maneira (conhecida) de resolver cabalmente o problema é testar todas as possibilidades. Para pequenas instâncias até que é possível. Para instâncias médias ou grandes é impraticável.

Colorir um grafo, é aplicar cores a seus vértices, de maneira que nenhum par de vértices adjacentes tenha a mesma cor, e de maneira que se use o menor número possível de cores.

Um exemplo de uso

Suponha-se um cruzamento de ruas, conforme mostrado no desenho: Existem

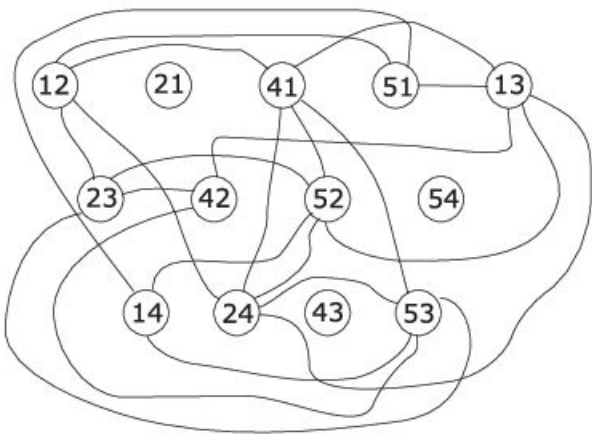


neste cruzamento da figura ?? 5 ruas. Destas, as ruas 3 e 5 tem mão única. As demais (1, 2 e 4) tem duas mãos. Pretende-se instalar um sinaleiro na esquina, e não existe a pretensão de proibir qualquer conversão. Analisando a esquina percebe-se existirem 13 conversões possíveis. (12, 13, 14, 21, 23, 24, 41, 42, 43, 51, 52, 53 e 54).

Nitidamente, existem conversões que podem ser feitas ao mesmo tempo (por exemplo 12 e 54), bem como existem conversões que não podem ser feitas ao mesmo tempo sob risco de colisão dos veículos (por exemplo 14 e 53). Deve-se estudar o problema, propor grupos de conversões que possam ser simultâneas, e finalmente identificar quantas fases deve ter o sinaleiro a fim de que as 13 conversões possam comетidas. Obviamente, o número de fases do sinaleiro deve ser o mínimo possível.

Uma maneira de resolver este problema é construir um grafo. Os vértices serão as 13 conversões possíveis. As arestas do grafo serão as conversões que não podem ser simultâneas. (por exemplo, 12-23, 24-53, etc). O *número cromático* deste grafo nos dirá a quantidade de fases que o sinaleiro deverá ter.

Eis como ficaria o grafo:



Outra possibilidade é obter o grafo através de sua matriz de adjacência:

	12	13	14	21	23	24	41	42	43	51	52	53	54
12					1	1	1			1			
13						1	1	1		1	1		
14										1	1	1	
21													
23	1							1			1	1	
24	1	1					1				1	1	
41	1	1				1					1	1	
42		1			1							1	
43													
51	1	1	1										
52		1	1		1	1	1						
53			1		1	1	1	1					
54													

A literatura registra 3 possibilidades de resolver este problema:

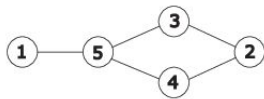
- Testar todas as possibilidades. Possível para grafos pequenos.
- Buscar informação adicional. Estudar o problema na busca de características que permitam eliminar possibilidades (evitando assim ter que testar todas as possibilidades).
- Usar uma heurística. ("um macete"). Usar esta estratégia conduz a resultados rápidos, não necessariamente ótimos, mas quase sempre aceitáveis.

Uma classe de heurísticas que pode ser usada aqui é a chamada solução pelo algoritmo *guloso*. A idéia é colorir tantos vértices quantos forem possíveis com a primeira cor. Depois, todos os que forem possíveis com a segunda cor e assim sucessivamente.

Algoritmo guloso

- 1: **enquanto** enquanto houver vértices sem cor **faça**
- 2: Escolha um vértice ainda incolor e aplique a ele a nova cor
- 3: Verifique todos os vértices ainda sem cor. Se eles não tiverem aresta com um vertice desta cor, aplique a mesma cor a eles.
- 4: **fim enquanto**

O único detalhe que fica faltando é qual a seqüência de vértices a considerar. Este detalhe pode fazer diferença na implementação. Se se usar uma ordem qualquer (por exemplo os números de nodos), o resultado pode não ser o melhor. Veja-se o exemplo da figura anexa. Se a numeração dos nodos for obedecida, serão necessárias 3 cores. (cor 1=1 e 2; cor 2=3 e 4; cor 3=5). Entretanto é razoável verificar que o grafo pode ser pintado com apenas 2 cores (cor 1=1, 3 e 4; cor 2=2 e 5). Seu **número cromático** é 2.



O truque que permite melhorar esta estratégia é considerar que quanto maior é o grau de um nodo, mais difícil é pintá-lo. Assim, ao invés de usar uma ordem arbitrária qualquer, sugere-se ordenar os nodos por grau (em ordem decrescente) e começar pelos mais difíceis (de maior grau). Em caso de empate nesta disputa, comece pelo vértice empatante de menor número. Escolhido o vértice, pesquise os demais nodos em ordem crescente sempre começando pelo menor (1, 2, ... n).

Para você fazer

Você deve achar o número mínimo de cores necessárias para colorir o grafo a seguir descrito pela sua matriz de adjacência. O grafo tem 12 vértices e 21 arestas e é não orientado. Eis a matriz de adjacência

0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	1	1	1	0	0	0	0	1	0
2	0	0	0	0	0	1	1	0	0	1	1	0
3	0	0	0	0	0	0	1	0	0	1	0	0
4	1	0	0	0	1	1	0	0	0	1	1	0
5	1	0	0	1	0	0	1	0	0	0	1	0
6	1	1	0	1	0	0	1	0	0	0	0	0
7	0	1	1	0	1	1	0	0	0	1	1	0
8	0	0	0	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	0	1	0
10	0	1	1	1	0	0	1	1	0	0	0	0
11	1	1	0	1	1	0	1	0	1	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

Responda a seguir

Quantas cores são necessárias para colorir este grafo usando o algoritmo guloso acima ?	qual o número da cor do nodo 9 ?	qual o número da cor do nodo 12 ?

1



- 1 - /

Aplicações de grafos: quebra-cabeça dos tijolos

Imagine 10 peças de dominó. Em cada uma delas você deve escrever uma das séries de 3 palavras a seguir listadas:

1	girafa	camelo	urso
2	tatu	girafa	tigre
3	peixe	lebre	tigre
4	asno	peixe	mosca
5	camelo	mosca	foca
6	vaca	guaxinim	morcego
7	morcego	lebre	urso
8	elefante	vaca	asno
9	elefante	anu	tatu
10	guaxinim	anu	foca

O objetivo do quebra-cabeças é formar um dominó com as 10 peças, no qual peças contíguas devem compartilhar um animal comum.

Nesta configuração, eis um possível formato do tabuleiro de dominó:

5 - 1 - 2 - 3 - 7 - 6 - 10 - 9 - 8 - 4 - 5

Entre as peças 5 e 1, o elo comum é o camelo. Entre as peças 1 e 2 é a girafa. Entre a 2 e a 3 é o tigre. Entre a 3 e a 7 é a lebre. Entre a 7 a e 6 é o morcego, entre a 6 e a 10 é o guaxinim, entre a 10 e a 9 é o anu, entre a 9 e a 8 é o elefante, entre a 8 e a 4 é o asno e entre a 4 e a 5 é a mosca. Preencha a solução no desenho a seguir para entendê-la.

Tente este, agora

1	girafa	camelo	asno
2	elefante	vaca	foca
3	anu	camelo	porco
4	morcego	foca	girafa
5	elefante	lebre	porco
6	morcego	gato	leão
7	urso	vaca	gavião
8	zebra	asno	urso
9	zebra	lebre	leão
10	anu	gavião	gato

Parece que esta segunda configuração, não pode ser resolvida.

Aquecimento

Verifique se esta instância de comida (sujeita as mesmas regras acima) pode ser resolvida e qual poderia ser uma resposta, em caso positivo.

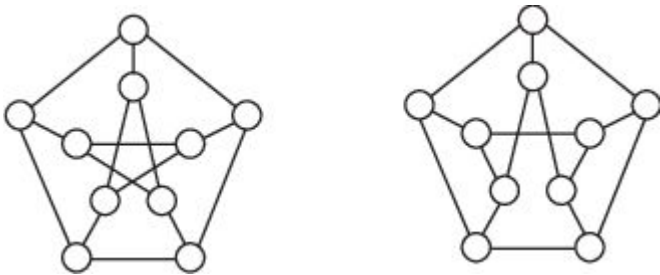
1	feijão	uva	banana
2	carne	aspargo	lentilha
3	batata	laranja	cebola
4	uva	abacaxi	carne
5	cenoura	batata	aspargo
6	feijao	cenoura	macarrao
7	macarrao	mimosa	lentilha
8	mimosa	repolho	aipim
9	abacaxi	aipim	cebola
10	banana	laranja	repolho

Como resolver

Antes de atacar o problema, deve-se estudar o conceito de ciclo hamiltoniano. Em contraposição ao ciclo Euleriano (caminho fechado – origem é contígua ao destino – que passa por todas as arestas, lembrar das pontes de Könningsberg) o ciclo hamiltoniano passa por todos os vértices uma e só uma vez e ao final encosta no início.

Exercício

Nos grafos a seguir, procure localizar a existência de pelo menos um ciclo hamiltoniano. O primeiro grafo tem nome, é chamado de Grafo de Petersen.



O primeiro grafo (Petersen) não admite ciclo hamiltoniano (uma demonstração está em www.cut-the-knot.org/pigeonhole/Petersen.shtml). Já o segundo grafo admite o ciclo (onde ???)

O quebra cabeça acima, pode ser colocado em um dos dois grafos, a saber: São 10 vértices (que correspondem aos 10 tijolos do enunciado). São 15 arestas, que podem corresponder aos nomes em comum em cada tijolo.

Deve-se pensar em um grafo, no qual cada tijolo é um vertice e as arestas que os conectam são as coisas (animais, comidas, etc) que estão presentes em dois vértices. No exemplo que foi resolvido acima, podemos ter:

Portanto, para resolver o quebra cabeça, basta desenhar um dos dois grafos. Se for possível colocar o problema no segundo, ele é solucionável e a solução passa pelos vértices do ciclo hamiltoniano. Se ele cair no grafo de Petersen o problema não tem solução. Simples, não ?

🔗 Para você fazer

A seguir, duas instâncias do quebra cabeça. Você deve responder:

- 0 se nenhuma das instâncias admitir solução
- 1 se apenas a primeira tiver solução
- 2 apenas a segunda tiver solução
- 3 ambas tiverem solução

Instância 1

1	tatu	asno	elefante
2	vaca	gaviao	tigre
3	cavalo	girafa	cachorro
4	tartaruga	peixe	elefante
5	porco	morcego	vaca
6	gaviao	cachorro	tatu
7	porco	asno	cavalo
8	gato	peixe	morcego
9	anu	tartaruga	tigre
10	girafa	anu	gato

Instância 2

1	guaxinim	cachorro	girafa
2	gaviao	leao	zebra
3	foca	gaviao	tigre
4	morcego	peixe	elefante
5	foca	gato	morcego
6	leao	asno	mosca
7	cachorro	gato	mosca
8	asno	vaca	elefante
9	girafa	vaca	tigre
10	peixe	guaxinim	zebra

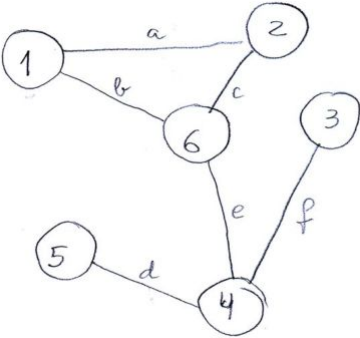
Resposta

1

Coloração de grafos: aplicações

Na matemática um grafo pode ser pensado como um conjunto de objetos sendo que alguns deles estão conectados de alguma forma. Os objetos são conhecidos como *vértices* e as conexões como *arestas*. Grafos podem ser usados para modelar problemas em uma série de áreas do conhecimento: química, engenharia elétrica, redes sociais, logística, navegação por satélite, entre muitas outras.

A seguir um exemplo de grafo no qual os vértices são {1,2,3,4,5,6}, as arestas {a,b,c,d,e,f} que estão conectadas como segue {(a,1,2), (b,1,6), (c,2,6), (d,1,5), (e,4,6), (f,3,4)}.



O problema da coloração é um dos mais famosos na área e ele pode ser usado na solução de muitos problemas como se verá a seguir. Deve-se colorir cada vértice do grafo com uma cor, sujeitando-se a duas regras

- Dois vértices conectados não podem ter a mesma cor.
- O número de cores deve ser *mínimo*.

No exemplo acima uma solução de coloração poderia ser cor 1: vértices 6, 3 e 5. Cor 2: vértices 1, 2 e 4. A quantidade de cores mínima é 2.

Uma maneira adequada de visualizar o problema da coloração é supor que os vértices são itens que precisam ser divididos em grupos. Imagine que um professor pretende dividir os alunos de uma turma em grupos de trabalho. O professor quer separar as panelinhas, assim dois alunos que são amigos não devem ficar no mesmo grupo. Por razões óbvias, o professor gostaria de trabalhar com o menor número de grupos possível. Neste caso, usaremos um grafo. Os vértices são os alunos e as arestas conectam 2 amigos entre si. Ao colorir o grafo, tem-se o menor número de grupos possível e nenhum grupo tem 2 amigos.

Uma segunda aplicação pode ser a construção de horários. Aqui há diversos eventos (aulas, provas, ensaios, etc) e cada um ocupa uma faixa de tempo (por exemplo, segundas das 10 as 12; quintas das 14 as 16,...) Nossa tarefa é associar eventos a horários seguindo algumas regras. Uma importante é que se uma pessoa (ou qualquer outro recurso) precisa estar em 2 atividades, elas não podem ocorrer no mesmo horário. Aqui os vértices podem ser os eventos e as arestas ligam eventos compartilhados por alguma (s) pessoa (s). Depois de colorir o grafo, cada cor vai representar uma faixa de tempo (horário).

Uma terceira aplicação pode ser a alocação de um grupo de taxis para transportar um grupo de pessoas. Cada grupo de pessoas deve usar um carro num determinado horário para fazer uma viagem de duração conhecida. A questão a responder é qual o número mínimo de taxis capazes de dar conta ? Pode-se construir um grafo onde os vértices são as viagens. As arestas conectam viagens que se sobrepõem no tempo. Agora cada cor é um taxi necessário para atender o grupo.

Um quarto exemplo pode ser dado em uma indústria química que precisa guardar diversos reagentes em galpões. Alguns reagentes são instáveis e não podem ser guardados no mesmo barracão. A pergunta a responder é: quantos galpões são necessários ? Agora o grafo tem como vértices os reagentes e as arestas conectam os reagentes que não podem ser aramezados juntos. As cores são os galpões.

Outra aplicação está em elaborar mapas de semáforos em cruzamentos complexos. A explicação completa esta é vivx635.

Um sexto exemplo é o nosso conhecido sudoku. Ele pode ser resolvido usando a coloração de grafos. Agora os vértices são as 81 casas do sudoku. Dois vértices tem aresta se:

- estão na mesma linha
- estão na mesma coluna
- estão na mesma sub-matriz

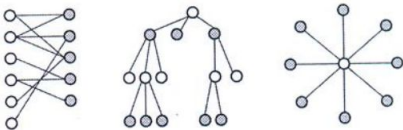
Neste caso, o número de cores é 9 (os dígitos de 1 a 9) e o objetivo é obter a coloração (ou seja: pintar os vértices com as cores). Na solução já aparece uma solução incompleta (que são os números pré-preenchidos) e o desafio é encontrar os números faltantes.

Há diversos algoritmos para solucionar um sudoku. Talvez o mais simples seja o da busca exaustiva com backtracking (vivx257), mas que a depender da quantidade de lugares vagos na grade pode demorar bastante. Aqui vão-se usar algoritmos de coloração de grafos para obter a solução.

- Deixe as casas preenchidas 1..9 como estão.
- Numere as casas zeradas a partir de 10.
- Monte as incompatibilidades
- Rode o programa DSATUR
- Interprete a resposta (em solution.txt). Não esqueça de notar a indireção nas 9 primeiras cores.

O problema

Admite muita teorização e estudo. Há livros sobre isso (por exemplo *A guide to graph colouring* de R.M.R. Lewis - cujo PDF pode ser encontrado na Internet). Há detalhes interessantes. Por exemplo, um grafo completo (todos os vértices são vizinhos de todos os vértices) com *n* vértices, exige *n* cores para ser colorido. Já um grafo bipartido (denotado $G = (V_1, V_2, E)$) são grafos cujos vértices podem ser particionados em 2 conjuntos V_1 e V_2 tais que as arestas *E* só existem entre V_1 e V_2 . Como resultado V_1 e V_2 são conjuntos independentes e com isso o grafo *G* pode ser colorido usando apenas 2 cores. Eis alguns exemplos de grafos bipartidos:



Origem A origem do problema remonta ao século XIX, quando Francis Guthrie, um sul africano postulou a conjectura das 4 cores afirmando que ao colorir um mapa qualquer (no caso o dos condados da Inglaterra) de modo que áreas com fronteira comum (maior que um único ponto) não usassem a mesma cor. A conjectura foi provada em 1976 por Kenneth Appel e Wolfgang Haken. Agora virou o Teorema das 4 cores e foi a primeira grande prova apoiada por computadores. Para os tradicionalistas, o teorema ainda não foi provado. Ela só se aplica a grafos planares que são grafos que podem ser desenhados no plano SEM que suas arestas se cruzem: um bom exemplo é o caso da luz, água e gás em 3 casas. Este problema não tem solução no plano.

Algoritmos e programas

Entrada e Saída de dados Existe um formato (mais ou menos) padronizado denominado DIMACS. DIMACS é a uma acrônimo para Center for Discrete Mathematics And Theoretical Computer Science. Este centro é uma iniciativa de diversas universidades e centros de pesquisa americanos, liderada pela Universidade Rutgers. Este formato para grafos não dirigidos é bem simples, no que nos interessa para o problema da coloração, comporta 3 tipos de dados - identificado por uma letra na primeira posição do registro, a saber: *c* para comentário; *p* para problema e *e* para aresta. O comando *c* aceita qualquer coisa. O comando *p* exige EDGE seguido do número de vértices e do número de arestas. O comando *e* exige os dois vértices conectados pela aresta. Eis um exemplo

c Grafo mostrado lá encima

p edge 6 6
e 1 2
e 1 6
e 2 6
e 4 5
e 4 6
e 3 4

A resposta vem em um arquivo denominado SOLUTION.TXT e ele começa com um único valor indicando quantos vértices foram coloridos. A seguir, uma série de cores (começando na cor 0) uma para cada vértice.

Exemplo Aplicando a mesma definição abaixo para a solução de:

0	3	6	0	0	7	0	0	4
2	0	0	3	8	0	0	0	5
0	0	0	1	0	0	8	0	3
0	0	0	4	6	0	5	0	0
0	0	8	0	0	0	6	0	0
0	0	5	0	9	8	0	0	0
7	0	2	0	0	5	0	0	0
1	0	0	0	3	2	0	0	6
4	0	0	6	0	0	2	8	0

Obteve-se na cruz central da sub-matriz central os seguintes valores: 6 5 7 3 9.

Para você fazer

A seguir, vamos resolver um SUDOKU usando o algoritmo DSATUR. DSATUR (cujo acrônimo significa *degree of saturation*) é muito parecido ao algoritmo guloso, mas na hora de escolher o próximo nodo a colorir, (isto é ao ordenar a lista de nodos candidatos a receber a cor) usa uma heurística ao considerar o número de cores diferentes que já foram alocados aos vértices adjacentes do candidato. Este algoritmo foi proposto por Bréaz em 1979 e tem resultado muito bom. Você deve obter este utilitário, chamado DSATUR.EXE no local habitual e deve submeter a ele no formato DIMACS o seguinte sudoku de solução única:

0	1	5	7	0	0	0	4	0
4	0	0	8	0	0	0	0	0
8	6	0	0	0	0	2	0	0
0	0	4	6	3	8	9	0	2
0	0	0	0	0	0	0	0	0
2	0	6	9	1	4	5	0	0
0	0	9	0	0	0	0	2	6
0	0	0	0	0	7	0	0	5
0	4	0	0	0	9	7	1	0

Responda a seguir os 5 valores da cruz central da submatriz central da resposta:

*****		*****
*****		*****
*****		*****
*****		*****
*****		*****
*****		*****



Método PERT-CPM

Em 1957 estávamos em plena Guerra Fria. Começava o projeto dos mísseis Polaris (250 empreiteiros, 9000 sub-empreiteiros, 70000 peças diferentes a fabricar - nunca antes fabricadas, atrasos no projeto inaceitáveis). Criou-se o método PERT (*Program Evaluation and Review Technique* - Técnica de avaliação e controle de programas). Ao mesmo tempo na Du Pont (indústria química) surgiu o método CPM (*Critical Path Method* - Método do Caminho Crítico). A partir de 1962, uma junção de ambos começou a ser conhecido como PERT-CPM.

Dado um projeto (conjunto de tarefas, suas interdependências e prazos, tendo como finalidade um determinado objetivo), o analista escreve uma rede. Nela, colocam-se as atividades e os eventos. Para cada atividade, determina-se a duração (exata ou probabilística) e a interdependência (eventos que a antecedem e que a sucedem).

Na rede, eventos conectam atividades e atividades conectam eventos. Existem diversas formas de representação, mas a mais comum é o método americano. Nele:

- seta** representa uma atividade. Por cima da seta escreve-se o nome da atividade e por baixo a sua duração.
- circulo** representa um evento. Dentro do círculo, o número ou nome do evento.

Quando duas atividades são feitas uma após a outra, apenas por comodidade, sem haver relação de dependência, os eventos em questão são interligados por uma atividade fantasma (uma seta tracejada). Atividades fantasma têm duração zero e podem, por conveniência ser deslocadas pela rede.

Dois eventos não podem ter mais de uma atividade entre eles. Se existirem 2 eventos realmente paralelos, introduz-se um evento fictício e uma atividade fantasma, a fim de obedecer à regra.

Quando uma atividade tem como pré-requisito uma condição externa à rede, (por exemplo, *tempo seco*), um evento fictício e uma atividade fantasma (tempo seco, neste caso) são introduzidas na rede.

Se a lista de tarefas é puramente linear e para começar uma a anterior deve ter terminado, não há muito que fazer e o desenho da rede tende a ser perda de tempo. Entretanto, se algum grau de paralelismo puder ser conseguido na realização de tarefas, então o método PERT-CPM tem muito a ajudar. No exemplo do míssil Polaris, o uso da técnica baixou o tempo do projeto originalmente estimado para 5 anos, para pouco mais de 3.

Método

1. Liste as atividades do projeto. Se forem muitas, cogite trabalhar com macro-tarefas (e sub-redes associadas). Avalie a necessidade de quebrar tarefas ou, ao contrário, de juntá-las.
2. Estime a duração de cada tarefa. Escolha uma unidade adequada (dia, semana, mês ?) que seja única para toda a rede.
3. Desenhe a rede, começando no evento **início** e terminando no evento **fim**, da esquerda para a direita, dispondo tarefas (com nomes e duração) e eventos.

A questão agora, é **qual a duração total do projeto?** A resposta é a soma das tarefas que estiverem no pior caminho (em termos de duração) entre os eventos de início e fim. Este caminho (que pode não ser único) é chamado de **caminho crítico**. Quaisquer tarefas que estejam neste caminho e que sofram algum atraso certamente atrasarão todo o projeto. Em contrapartida, se elas sofrerem adiantamento, isto não garante que o projeto todo seja adiantado, pois agora, o caminho crítico poderá ser outro.

Para calcular o CC, atribui-se o tempo 0 (zero) ao evento início. Para cada atividade seguinte, soma-se o seu tempo com o tempo do(s) evento(s)

precedente(s). Escreve-se o tempo de cada evento entre parênteses, sobre o círculo do evento. Este tempo também é chamado **TEMPO MAIS CEDO** ou simplesmente **CEDO** (early) e ele é o tempo necessário para que um evento seja atingido, desde que não haja atrasos nas atividades precedentes.

Se a um determinado evento chegam mais de uma atividade, comparam-se todas as somas possíveis e escolhe-se a maior delas.

Quando se obtiver o tempo do evento fim, este valor será a duração do projeto. O caminho que vai do início ao final, passando pelas atividades de maior tempo, é o caminho crítico.

As atividades não críticas (fora do caminho) podem eventualmente sofrer atraso sem que isto atrase o projeto.

Ao chegar ao último evento, a data cedo é a menor duração possível para o projeto. Esta data (apenas no último evento) deve ser copiada para baixo e passa a constituir o tempo mais tarde do último evento.

Outro tempo importante é o **TEMPO MAIS TARDE**, ou simplesmente **TARDE** (late) que é a data limite de realização de um evento. Qualquer execução que passar desta data, atrasará o projeto. O tempo TARDE é escrito em um pequeno retângulo sobre os parênteses do tempo CEDO.

Para calcular o tempo tarde, parte-se do final e para cada atividade que chega a ele, subtrai-se a duração das tarefas que a ele chegam. Se mais de uma tarefa sai de um evento, o tempo tarde é o menor valor dos resultados-subtrações.

Eventos que fazem parte do CC tem obrigatoriamente o **CEDO = TARDE**. Finalmente, a folga de um evento, é a diferença entre o **TARDE - CEDO**.

Exemplo

Seja a seguinte rede:

tarefa	origem	destino	duração
A	1	2	7
B	2	3	2
C	1	3	5
D	2	4	4
E	2	5	6
F	3	6	3
G	5	6	5
H	2	7	2
I	5	7	2
J	2	8	2
K	3	8	5
L	4	8	6
M	7	8	9
N	7	9	3
O	4	9	3
P	1	9	1
Q	8	9	8
R	9	10	1
S	5	11	2
T	8	11	2
U	11	12	3
V	6	12	2
W	10	12	3

Desenhando a rede e executando o algoritmo acima, chegaremos aos valores de

evento	cedo	tarde	C.Crit
1	0	0	*
2	7	7	*
3	9	19	
4	11	18	
5	13	13	*
6	18	34	
7	15	15	*
8	24	24	*
9	32	32	*
10	33	33	*
11	26	33	
12	36	36	*

Conclui-se portanto que a duração do projeto é de 36 unidades e que o caminho mínimo tem 8 eventos, a saber: o início (1), 2, 5, 7, 8, 9, 10 e o final (12).

Para você fazer

1. Suponha a seguinte rede

tarefa	origem	destino	duração
A	1	2	9
B	1	3	5
C	3	4	7
D	1	4	8
E	3	5	6
F	4	5	2
G	3	6	8
H	3	7	4
I	1	8	5
J	7	8	9
K	5	9	1
L	6	9	5
M	8	9	1
N	1	10	6
O	9	10	3
P	10	11	9
Q	7	12	4
R	12	13	8
S	2	13	9
T	3	13	9
U	11	13	8

1. Desenhe o grafo (a rede) de precedências

2. Calcule os tempos mais cedo

3. Calcule os tempos mais tarde

4. assinale o caminho crítico (passa pelos nodos onde cedo=tarde)

5. Responda: qual a duração do projeto e quantos nodos tem o CC (caminho crítico) ?

2. Suponha a seguinte rede

tarefa	origem	destino	duração
A	1	2	7
B	2	3	4
C	3	4	4
D	3	5	5
E	5	6	7
F	4	6	8
G	5	7	2
H	1	8	4
I	6	8	3
J	7	8	4
K	7	9	7
L	4	10	8
M	9	10	4
N	7	10	7
O	2	10	5
P	9	11	4
Q	8	11	9
R	10	12	7
S	11	12	6

1. Desenhe o grafo (a rede) de precedências

2. Calcule os tempos mais cedo

3. Calcule os tempos mais tarde

4. assinale o caminho crítico (passa pelos nodos onde cedo=tarde)

5. Responda: qual a duração do projeto e quantos nodos tem o CC (caminho crítico) ?

Para saber mais: Planejamento com PERT-CPM, Henrique Hirschfeld, Editora Atlas.

Problema de localização de um hipermercado

Seja uma cidade na qual eventuais investidores pretendem instalar um hipermercado. O objetivo deste exercício é obter a localização ótima para o empreendimento.

Como dado de entrada do modelo, vai-se obter a distribuição espacial de habitantes na cidade. Neste caso por simplicidade, vai-se trabalhar com uma cidade de ruas e avenidas paralelas entre si, formando um grid retangular (à la Nova Iorque). Se a cidade real não tiver esta configuração, a modificação no algoritmo é simples. Basta tomar as coordenadas de cada centro de quarteirão e substituir a distância euclidiana dos índices pela distância real entre 2 conjuntos de coordenadas.

Na configuração abaixo, mostra-se um bairro dessa cidade, na forma de uma tabela de 30 linhas por 15 colunas. O número que aparece na matriz já é o número de consumidores previstos por quarteirão. Em outras palavras $M_{i,j}$ indica o número de consumidores que moram no quarteirão da linha i com a coluna j .

Para prosseguir na simulação, há que se ter uma regra de distribuição desses consumidores. Qualquer regra pode ser usada, mesmo uma *ad hoc*. Eis aqui uma proposta de regra

quem mora	quarteirões	% de consumidores compram
a menos de	3	75%
entre	3,01 e 5	50%
entre	5,01 e 7	25 %
entre	7,01 e 10	10%
mais de	10	apenas 2%

Uma observação importante quando você programar este modelo é deixar os números da regra como parâmetros externos ao código de maneira a poder variá-los sem ter que modificar a programação. Até porque, para dar mais realismo ao modelo, cada HM já instalado pode ter a sua regra própria de atração de clientes, diferente da dos seus concorrentes.

Os habitantes que ficam fora desta conta são atendidos por quitandas, mercadinhos, escambo, feiras livres etc. Neste exercício só nos interessam os hipermercados.

Para efeito de simplificação, todos os habitantes de um quarteirão são considerados morando no centro geométrico do quarteirão.

A pergunta que o exercício vai responder é onde localizar o novo hipermercado da cidade. Entretanto, já pode haver diversos HM instalados na cidade. Obviamente a localização deles e a influência que já exercem sobre a malha de habitantes certamente precisa ser considerada.

Para cada HM já instalado na cidade, deve-se criar uma nova matriz (do mesmo tamanho = 30 x 15) e nela devem ser colocados os potenciais habitantes servidos por este HM. Usar a regra acima descrita para fazer este cálculo.

Depois que todos os HM já existentes tiverem sua matriz de influência calculada, é hora de fazer algumas inferências:

- Considere $M_{i,j}$ como sendo a quantidade total de consumidores. Este dado é fornecido pelo exercício.
- Considere $H_{k,i,j}$ como sendo a influência do HM k , no quarteirão i, j da cidade (usando a regra acima).
- Crie uma nova matriz H , resultado da totalização de todas as matrizes H_k
- Comparando $H_{i,j}$ com $M_{i,j}$

Se $H_{i,j} \geq M_{i,j}$ sobram zero consumidores no local i, j . Neste caso, portanto, $R_{i,j} = 0$.

Se $H_{i,j} < M_{i,j}$ a diferença entre estes dois valores é a quantidade de consumidores remanescentes e este resultado vai para $R_{i,j}$.

A variável R vai comandar a simulação a partir deste ponto. Lembrando que o objetivo é localizar qual o quarteirão i, j onde um eventual hipermercado seja instalado de maneira a que ele tenha O MAIOR N MERO POSS VEL DE CONSUMIDORES.

Para responder a esta pergunta, este programa vai localizar o HM em todos os quarteirões possíveis na cidade (um de cada vez, é claro). Localizado o HM em um dado quarteirão, o programa deve calcular qual a influência (quantos consumidores) o HM vai criar. Para tanto, usar a regra acima, sobre a matriz R e totalizar os consumidores. Colocar esta soma em uma nova variável $S_{i,j}$.

Note que é um modelo consumidor de recursos de CPU, já que tem complexidade $O(n^4)$. Ao final deve-se pesquisar qual o maior valor de S e uma vez encontrado, a resposta da localização do HM é dada pela linha e coluna do maior valor de S .

Exemplo

Seja uma cidade 10x6 conforme visto:

```

49 64 73 79 51 83
55 31 33 28 71 50
56 47 87 64 47 41
72 76 57 93 72 50
89 86 69 48 36 39
46 68 84 51 71 38
51 43 48 91 90 37
35 61 89 38 90 78
82 38 82 67 71 93
51 33 52 40 58 36
```

Se existirem 2 mercados em 3,3 e em 8,6, as duas influências serão

```

36 48 54 59 38 41      4 6 7 7 5 20
41 23 24 21 53 25      5 3 8 7 17 12
42 35 65 48 35 30      5 11 21 16 11 20
54 57 42 69 54 25      18 19 28 46 36 25
66 64 51 36 27 19      22 43 34 24 18 29
23 34 63 25 35 19      11 34 42 38 53 28
25 21 24 45 45 18      12 21 24 68 67 27
8 15 44 9 22 19        17 30 66 28 67 58
20 9 20 16 17 23        20 19 41 50 53 69
5 3 13 4 5 3           12 16 26 30 43 27
```

O que sobra de consumidores após retirar os clientes que os mercados já instalados atingem (ou satisfazem) é

```

9 10 12 13 8 22
9 5 1 0 1 13
9 1 1 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
12 0 0 0 0 0
14 1 0 0 0 0
10 16 0 1 1 1
42 10 21 1 1 1
34 14 13 6 10 6
```

Finalmente, o potencial de consumo em cada possível localização do novo mercado é

```

89 93 95 94 89 78
96 96 103 101 93 80
108 115 113 116 101 89
126 125 129 120 110 88
141 143 143 134 122 99
151 148 156 146 127 104
160 160 162 145 133 101
159 159 164 141 132 91
---
154 153 154 142 123 101
150 149 150 138 121 95
```

E a melhor localização está na linha 8 coluna 3.

Para você fazer

Suponha uma cidade contendo a seguinte distribuição de habitantes

```

56 64 65 31 40 76 74 48 61 43 71 68 66 54 75
71 42 31 27 63 55 69 43 73 54 64 49 71 77 40
57 49 73 87 72 72 65 49 56 62 91 41 86 56 89
70 55 44 35 70 49 51 34 59 55 27 33 49 86 47
70 57 52 66 63 69 64 68 53 46 84 70 43 48 41
29 40 61 49 72 47 61 71 66 72 47 51 49 72 53
77 60 58 66 51 47 56 85 48 52 81 63 72 65 34
46 49 53 18 83 49 38 72 69 58 36 92 59 64 83
39 68 71 23 82 47 72 75 63 49 73 66 77 66 84
26 32 40 74 43 54 41 86 78 87 67 70 59 48 36
35 23 87 80 61 50 89 38 73 58 69 90 94 69 38
21 67 29 74 30 33 38 51 91 53 88 38 62 57 69
38 40 40 44 58 62 37 82 39 57 77 73 80 32 81
50 19 79 66 81 69 60 70 70 76 72 70 39 54
72 30 68 57 54 61 43 81 93 55 58 50 62 37 60
55 38 69 36 30 46 67 55 48 42 36 49 35 44 61
15 64 59 60 58 52 67 53 77 42 52 51 62 24 37
31 55 52 74 84 60 72 69 40 37 57 28 56 46 38
37 60 73 60 38 42 29 45 82 82 80 57 21 81 90
67 71 20 31 50 51 52 51 67 51 33 75 46 43 32
14 31 57 40 31 40 10 27 82 61 33 63 32 77 10
27 45 47 44 57 87 8 21 53 41 60 44 71 46 34
20 31 62 34 36 49 78 61 41 59 53 66 63 87 31
62 62 76 66 25 15 3 58 15 44 60 44 31 17 55
18 41 15 51 64 20 41 45 33 66 56 35 51 22 63
12 50 43 20 18 24 51 20 46 45 53 47 61 30 64
55 4 70 50 49 75 28 6 42 19 27 21 74 30 21
86 26 43 45 26 58 0 13 28 37 9 58 49 57 62
53 25 15 40 58 40 37 39 74 4 43 39 47 52 27
29 48 0 68 61 3 62 10 28 29 17 58 50 38 4
```

Existem 3 hipermercados já instalados nessa cidade e eles estão localizados nos seguintes quarteirões:

- 12, 12
- 2, 15
- 3, 13

Utilize o algoritmo acima e descubra qual a melhor localização para um novo hipermercado que quer se instalar na cidade. Se houver empate, pegue o primeiro número maior.

l	c



Central 0800

Suponha que você é um consultor na solução de problemas (um dos mais nobres papéis de um engenheiro de computação ou de um analista de sistemas). Seu nome foi fortemente recomendado para estudar o ambiente de negócios de uma importante multinacional que vai se instalar no Estado do Paraná. Um dos requisitos que você deve propor é a LOCALIZAÇÃO DA CENTRAL 0800. Em outras palavras, você deve sugerir em que cidade deve ser criada o SAC (serviço de apoio ao cliente).

Como se sabe, os centros de apoio (telefônico) ao cliente funcionam de maneira centralizada recebendo ligações faturadas no destino (0800) e prestando informações pelo telefone aos interessados.

Os grandes custos associados nesta empreitada são os de pessoal (salários), infraestrutura local e tráfego telefônico. A você cabe estudar o tráfego telefônico vis-a-vis o porte da cidade escolhida. Para tanto, foi-lhe fornecida uma tabela aproximada de onde se espera seja gerado o tráfego telefônico até a central.

Há aqui, uma dificuldade óbvia. Como as ligações são faturadas em degraus tarifários (quanto mais longe mais cara é a ligação), a primeira hipótese é localizar a central na cidade que mais gera tráfego. Entretanto, as maiores cidades têm custos maiores de implantação (terreno, construção, deslocamento de empregados, etc).

Por exemplo, suponhamos um caso hipotético que tenha a seguinte geração de tráfego.

CIDADE	PORTE	LOCALIZ.	MIN/M S
ADRIANOPOLIS	1	495 146	12311
TELEM BORBA	1	341 185	9320
ORTIGUEIRA	1	310 202	8389
JACAREZINHO	2	399 315	18285
PONTA GROSSA	3	377 102	21656
PARANAVAI	2	158 323	13534
CASCAVEL	3	53 118	27080
GUAIRA	1	-25 213	9091
PALMAS	2	200 -54	11018
CURITIBA	4	467 63	30866
MARINGA	3	211 287	27416
FOZ IGUAÇU	3	-59 55	23223

Os portes associados às cidades são: 1=pequeno; 2=médio; 3=grande e 4=capital. A localização está medida em quilômetros referentes a um sistema de eixos cartesianos que tem a origem um pouco ao sul de Cascavel.

A coluna de minutos por mês identifica a quantidade de tráfego telefônico originado nessa cidade e destinado à central. Como o número é um 0800 o tráfego é integralmente faturado no destino.

Por hipótese, a companhia telefônica selecionada estabeleceu uma tarifa baseada na seguinte regra:

- Franquia de 2000 minutos/mês de qualquer origem.
- Tarifa básica de R\$ 0,8315/minuto.
- Ligações com distância inferior a 80Km (lineares) pagam 100% da tarifa básica. (degrau tarifário A).
- Ligações com 80 ou mais quilômetros, até 200 Km, pagam 180% da tarifa básica. (DT=B).
- Ligações com 200 ou mais quilômetros, até 500 Km, pagam 251% da tarifa básica. (DT=C)
- Ligações com 500 ou mais quilômetros pagam 370% da tarifa básica. (DT=D).

Para contrabalançar o tráfego maior nas maiores cidades (o que implica em custos maiores de implantação nas grandes cidades), você deve dar um acréscimo no valor final diretamente proporcional ao porte da cidade. Assim centrais localizadas em cidades de porte 1 devem ter 10% de acréscimo no custo final.

Cidades de porte 2 devem ter 20% e cidades de porte 3 devem ter 30% de acréscimo. A capital deve ter 40% de acréscimo.

O processamento então deve ser:

- Examinar a tabela de cidades estudando a localização, porte e tráfego do problema.
- Criar uma tabela de distâncias (pitagóricas, ou seja em linha reta) entre todas as cidades envolvidas.
- Criar uma tabela de custos de telefonia por degrau tarifário.
- Traduzir a tabela de distâncias para tabela de degraus tarifários entre as cidades envolvidas.
- Supor que o centro se instalará em cada uma das 12 cidades citadas no seu problema. Em cada caso, determinar o custo de telefonia para todas as 12 cidades, aplicando os degraus tarifários, a franquia e o acréscimo por porte.
- Selecionar a cidade de menor custo final.

Processo de solução

A partir dos dados do problema chega-se a esta matriz de distância entre as cidades:

0	159	193	194	126	381	443	524	356	88	317	561
159	0	35	142	90	229	296	367	277	175	165	421
193	35	0	144	120	194	270	335	279	210	130	397
194	142	144	0	214	241	398	436	419	261	190	527
126	90	120	214	0	311	324	417	236	98	249	439
381	229	194	241	311	0	230	214	379	404	64	345
443	296	270	398	324	230	0	123	226	418	231	129
524	367	335	436	417	214	123	0	349	514	247	162
356	277	279	419	236	379	226	349	0	292	341	281
88	175	210	261	98	404	418	514	292	0	340	526
317	165	130	190	249	64	231	247	341	340	0	356
561	421	397	527	439	345	129	162	281	526	356	0

A partir desta tabela de distância, cria-se uma tabela de degraus tarifários:

	A	T	O	J	P	P	C	G	P	C	M	F
ADRIANOPOLIS	A	B	B	B	B	C	C	D	C	B	C	D
TELEM BORBA	B	A	A	B	B	C	C	C	C	B	B	C
ORTIGUEIRA	B	A	B	B	B	C	C	C	C	B	C	B
JACAREZINHO	B	B	B	A	C	C	C	C	C	C	B	D
PONTA GROSSA	B	B	B	C	A	C	C	C	C	B	C	C
PARANAVAI	C	C	B	C	A	C	C	C	C	C	A	C
CASCAVEL	C	C	C	C	C	C	A	B	C	C	C	B
GUAIRA	D	C	C	C	C	C	B	A	C	D	C	B
PALMAS	C	C	C	C	C	C	C	A	C	C	C	C
CURITIBA	B	B	C	B	C	D	C	A	C	A	C	D
MARINGA	C	B	B	C	A	C	C	C	A	C	A	C
FOZ IGUAÇU	D	C	C	D	C	C	B	B	C	D	C	A

Agora, calcula-se quanto custaria instalar a central 0800 em cada uma das 12 cidades, obtendo-se os seguintes valores: (antes e depois dos acréscimos pelo porte)

ADRIANOPOLIS	361479.72	397627.70
TELEM BORBA	316198.38	347818.22
ORTIGUEIRA	326430.59	359073.64
JACAREZINHO	364130.36	436956.43
PONTA GROSSA	336861.25	437919.62
PARANAVAI	342597.71	411117.25
CASCAVEL	344557.51	447924.76
GUAIRA	395288.85	434817.73
PALMAS	381439.99	457727.99
CURITIBA	362522.87	507532.02
MARINGA	328662.14	427260.78
FOZ IGUAÇU	402002.00	522602.61

Como se pode ver o menor custo é de TEL MACO BORBA que é de 347.818,22.

Para você fazer

Suponha o mesmo problema, mas com a seguinte distribuição de tráfego telefônico:

CIDADE	PORTE	LOCALIZ.	MIN/M S
1-CANDIDO ABREU	1	269 157	14204
2-LONDRINA	3	291 297	40849
3-JACAREZINHO	2	399 315	22682
4-FOZ IGUAÇU	3	-59 55	41036
5-IRATI	2	338 62	27897
6-GUAIRA	1	-25 213	13255
7-ORTIGUEIRA	1	310 202	17019
8-MARINGA	3	211 287	40453
9-CASCAVEL	3	53 118	36804
10-FCO BELTRAO	2	91 -8	23824
11-GUARAPUAVA	3	254 67	39712
12-LARANJ SUL	1	159 68	13389

Sujeito às seguintes regras:

- Franquia em minutos/mês: 2000
- Tarifa Básica em reais: 00.8222
- Ligações com distância inferior a 100 Km, pagam 100 % da tarifa básica.
- Ligações com distância igual ou inferior a 230 Km, pagam 190 % da tarifa básica.
- Ligações com distância igual ou inferior a 490 Km, pagam 320 % da tarifa básica.
- Ligações com distância superior a 490 Km, pagam 400 % da tarifa básica.
- Cidades de porte 1 tem 10% de aumento, porte 2, 20%, 3 30% e a capital tem 40% de aumento.

Responda agora:

Qual a cidade de menor custo
CUSTO em R\$ /mês



Algoritmo Húngaro

É um algoritmo de otimização combinatória que resolve problemas de assinalamentos em tempo polinomial ($O(n^3)$). A primeira versão, conhecida como “Método Húngaro” foi inventada e publicada por Harold Kuhn em 1955. Foi revisada por James Munkres em 1957. Desde então o algoritmo é conhecido como “Algoritmo Húngaro”, “Algoritmo de Assinalamento de Munkres” ou “Algoritmo Kuhn-Munkres”. Kuhn baseou-se no trabalho de dois outros matemáticos húngaros: Dénes König e Jenő Egerváry. A vantagem do método de Kuhn é que ele é fortemente polinomial.

O problema

Imagine que você tem uma lista de tarefas e uma lista de pessoas que podem cumprir todas as tarefas. Ambas as listas tem o mesmo comprimento, ou seja, qualquer pessoa pode desempenhar qualquer tarefa e qualquer tarefa pode ser feita por qualquer um. Entretanto, nada impede que alguma inadequação de uma pessoa a uma tarefa seja representada como custo= ∞ , desde que alguém sempre faça cada tarefa e cada pessoa faça alguma coisa.

Uma matriz de ordem quadrada indica os custos de uma pessoa (na linha) fazer uma tarefa (na coluna). Por exemplo, em

3	5	10
11	7	2
4	9	8

Que deve ser entendida como:

	tarefa 1	tarefa 2	tarefa 3
pessoa 1	3	5	10
pessoa 2	11	7	2
pessoa 3	4	9	8

Existem 3 pessoas e 3 tarefas. O custo da pessoa 1 fazer a tarefa 2 é de 5 unidades. O custo da pessoa 3 fazer a tarefa 1 é de 4 unidades, e assim por diante.

O objetivo, ao final do algoritmo, é dizer quem deve fazer o que, de maneira que o custo total seja MÍNIMO. A propósito, no exemplo acima, o menor custo seria 1 faz 2 (custo=5), 2 faz 3 (custo 5+2=7) e 3 faz 1 (custo total=5+2+4=11). O problema surge quando a instância não é tão pequena ou simples para ser resolvida por inspeção visual.

O algoritmo

Em termos matemáticos, o problema pode ser colocado como “dados n trabalhadores e n tarefas, e uma matriz $i \times j$, onde $n = i = j$, com os custos de assinalar o trabalhador i à tarefa j , achar o assinalamento de custo mínimo”.

O algoritmo tem as seguintes etapas:

- Em cada linha da matriz, subtraia o menor valor da linha
- Em cada coluna da matriz, subtraia o menor valor da coluna
- Cobertura de zeros: Para cada zero na matriz:
 - Verifique se o zero é único na linha. Se for, aloque a tarefa à pessoa, risque a coluna, e vá para o próximo zero não riscado.
 - Verifique se o zero é único na coluna. Se for, aloque a tarefa à pessoa, risque a linha e vá para o próximo zero não riscado.
- Teste de fim: cada tarefa já é feita por alguém ? Se sim, o algoritmo acabou.
- Existe algum zero não riscado ? Se sim, este algoritmo (como está) é incapaz de resolver o problema.
- Ache o menor valor que não está riscado. Subtraia todos os valores não riscados, deste e retorne a [3].

Acompanhando a execução do algoritmo

- Seja uma lista de 7 pessoas e 7 tarefas com a seguinte matriz de custos

8	9	4	3	5	7	12
1	12	7	7	2	7	6
4	8	9	2	1	1	6
3	8	10	8	4	7	6
11	6	2	10	10	6	5
8	8	12	11	3	10	1
7	9	12	3	1	5	9

- Apos a execução de [1], fica

5	6	1	0	2	4	9
0	11	6	6	1	6	5
3	7	8	1	0	0	5
0	5	7	5	1	4	3
9	4	0	8	8	4	3
7	7	11	10	2	9	0
6	8	11	2	0	4	8

- Após a execução de [2], fica

5	2	1	0	2	4	9
0	7	6	6	1	6	5
3	3	8	1	0	0	5
0	1	7	5	1	4	3
9	0	0	8	8	4	3
7	3	11	10	2	9	0
6	4	11	2	0	4	8

- Após a execução do primeiro [3], houve as alocações: 1,4; 2,1; 3,6; 5,2; 6,7 e 7,5. Perceba-se que a tarefa 3 não é feita por ninguém.

5	2	1	0	2	4	9
0	7	6	6	1	6	5
3	3	8	1	0	0	5
0	1	7	5	1	4	3
9	0	0	8	8	4	3
7	3	11	10	2	9	0
6	4	11	2	0	4	8

Análise dos zeros da matriz anterior:

- L1,C=4: como é o único na linha, é alocado e risca-se a coluna 4.
L2,C=1: como é o único na linha, é alocado e risca-se a coluna 1.
L3,C=5: não é unico na linha. Não é único na coluna, abandonado.
L3,C=6: é o único na coluna. É alocado e risca-se a linha 3.
L4,C=1: já foi riscado.
L5,C=2: não é o único na linha, mas é na coluna. Aloca-se e risca L5.
L5,C=3: já foi riscado.
L6,C=7: é o unico na linha. Aloca-se-o e risca-se a coluna 7.
L7,C=5: único na linha. Alocado e risca-se a coluna 5.

Os valores que não foram riscados foram 2 1 4 7 6 6 1 7 4 3 11 9 4 11 4 e o menor deles é 1.

- Após a subtração, fica

5	1	0	0	2	3	9
0	6	5	6	1	5	5
3	3	8	1	0	0	5
0	0	6	5	1	3	3
9	0	0	8	8	4	3
7	2	10	10	2	8	0
6	3	10	2	0	3	8

Agora, o ciclo recomeça desde o início.

- L1,C3: não é único na linha nem na coluna. Abandonado.
L1,C4: único na coluna. Alocado e riscando a linha 1.
L2,C1: único na linha. Alocado e riscando a coluna 1.
L3,C5: não é único na linha nem na coluna. Abandonado.
L3,C6: não é único na linha, mas o é na coluna. Aloca e risca L3.
L4,C1: já foi riscado.
L4,C2: único na linha (o outro foi riscado). Aloca e risca C2.
L5,C2: já foi riscado.
L5,C3: único na linha, aloca e risca a coluna 3.
L6,C7: único na linha. Aloca e risca a coluna 7.
L7,C5: único na linha. Aloca e risca a coluna 5.

Que nos dá a resposta do problema ou

0	0	0	1	0	0	0
1	0	0	0	0	0	0
0	0	0	0	0	1	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0

Que é a solução final.

Para você fazer

Seja um problema de 7 pessoas e 7 tarefas, dado pela matriz de custos

3	10	1	7	11	6	4
7	8	1	7	4	9	1
4	12	5	12	8	8	7
4	2	6	1	10	10	3
5	10	2	4	7	11	12
1	5	7	10	8	12	7
3	10	7	12	4	1	12

Ache o assinalamento de menor custo e indique qual a tarefa que cada pessoa fará

	pessoa 1	pessoa 2	pessoa 3	pessoa 4	pessoa 5	pessoa 6	pessoa 7



O problema dos residentes e hospitais

Uma instância I do problema dos residentes e hospitais (HR) envolve um conjunto $R = \{r_1, r_2, \dots, r_n\}$ de residentes e um conjunto $H = \{h_1, h_2, \dots, h_m\}$ de hospitais. Cada hospital $h_j \in H$ tem uma capacidade (inteiro, positivo), denotada por c_j . Cada residente $r_i \in R$ tem uma lista de preferência na qual está listado em estrita ordem um subconjunto de H . Um par $(r_i, h_j) \in R \times H$ é dito *aceitável* se h_j aparece na lista de preferência de r_i . Neste caso, diz-se que r_i acha h_j aceitável. Similarmente, cada hospital $h_j \in H$ tem uma lista de preferência na qual são listados, em rigorosa ordem, os residentes que acham h_j aceitável.

Dados um residente x e dois hospitais y e z diz-se que x prefere y a z se x acha ambos (y e z) aceitáveis, e y vem antes de z na lista de preferências de x .

Seja A o conjunto de pares aceitáveis em I , e seja L a cardinalidade de A ($L = |A|$). Um assinalamento M é um subconjunto de A . Se $(r_i, h_j) \in M$, diz-se que r_i é assinalado a h_j e que h_j é assinalado a r_i .

Para cada $q \in R \cup H$ o conjunto de assinalamentos de q em M (que podem ser tanto residentes quanto hospitais) é denotado por $M(q)$.

Se $r_i \in R$ e $M(r_i) = \emptyset$, diz-se que r_i está não assinalado, em caso contrário diz-se que r_i está assinalado. Similarmente, cada hospital $h_j \in H$ é infrassubscrito, cheio ou supersubscrito de acordo com $|M(h_j)|$ é menor, igual ou maior do que c_j respectivamente.

Um casamento M é um assinalamento tal que $|M(r_i)| \leq 1$ para todo $r_i \in R$ e $|M(h_j)| \leq c_j$ para todo $h_j \in H$ (ou seja, nenhum residente está assinalado para uma hospital não aceitável, cada residente está assinalado no máximo a um hospital e nenhum hospital está supersubscrito). Por conveniência notacional, dado um casamento M e um residente $r_i \in R$ tal que $M(r_i) \neq \emptyset$, onde não haja ambiguidade, a notação $M(r_i)$ pode também ser usada para se referir a um elemento único de $M(r_i)$.

Um par $(r_i, h_j) \in A \setminus M$ bloqueia o casamento M , ou é um par bloqueador para M se não existe candidato c e instituição i , não designados entre si, tal que o candidato prefere essa instituição em vez de aquela que foi designado e paralelamente a instituição prefere esse candidato a algum que tenha sido designado pelo casamento. Este par (c, i) é chamado par bloqueante e ele causa instabilidade no casamento.

Um casamento M é dito estável se ele não admite pares bloqueadores. Dada uma instância I de um problema HR, a solução é encontrar um casamento estável em I .

O problema HR foi definido por Gale e Shapley sob o nome “Problema da Admissão à Faculdade”. No seu trabalho seminal o primeiro problema considerado foi o clássico problema do casamento estável, que é um caso especial do problema HR, onde $n = m$, $A = R \times H$ e $c_j = 1$ para todo $h_j \in H$. Neste caso os residentes e hospitais são conhecidos como homens e mulheres respectivamente. Os autores demonstraram que qualquer instância I do problema HR admite ao menos uma solução estável. A prova deste resultado é construtiva, ou seja um algoritmo que ache um casamento estável em I é mostrado. Este algoritmo passou a ser conhecido com o nome de “Algoritmo Gale/Shapley”.

Uma versão estendida desse algoritmo é aqui mostrada. O algoritmo envolve uma sequência de operações de aplicação e exclusão. A cada interação dentro do ciclo *while* algum residente r_i ainda não assinalado com uma lista de preferências não vazia é aplicado ao primeiro hospital h_j na sua lista e torna-se temporariamente assinalado a h_j , já que este assinalamento poderá ser quebrado à frente. Se h_j torna-se supersubscrito como resultado desse assinalamento, então h_j rejeita seu pior residente já assinalado r_k . E, este residente deve voltar para a fila de estudantes ainda não alocados. A seguir, se h_j está cheio (estas duas condições são independentes e podem ou não acontecer no mesmo *while*), então para cada residente r_l que h_j acha menos desejável do que o seu pior residente r_k o algoritmo *exclue* o par (r_l, h_j) o que implica em excluir h_j da lista de preferência de r_l e vice-versa (excluir r_l da lista de preferências de h_j).

```

1:  $M \leftarrow \emptyset$ 
2: enquanto algum residente  $r_i$  está não assinalado  $\wedge r_i$  tem lista não vazia faça
3:    $h_j \leftarrow$  o primeiro hospital na lista de  $r_i$ 
4:    $M \leftarrow M \cup \{(r_i, h_j)\}$ 
5:   se  $(h_j)$  está supersubscrito então
6:      $r_k \leftarrow$  o pior residente em  $M(h_j)$ 
7:      $M \leftarrow M \setminus \{(r_k, h_j)\}$ 
8:   fim se
9:   se  $h_j$  está cheio então
10:     $r_k \leftarrow$  o pior residente em  $M(h_j)$ 
11:    para cada sucessor  $r_l$  na lista  $M(h_j)$  faça
12:      exclua o par  $(r_l, h_j)$ 
13:    fim para
14:  fim se
15: fim enquanto

```

Já que este algoritmo envolve residentes sendo aplicados a hospitais, também é conhecido como Algoritmo Gale/Shapley orientado a residentes. (ou algoritmo RGS). Ele termina com um casamento estável. Dependendo da estrutura de dados usada, o algoritmo RGS roda em tempo $O(L)$. O casamento produzido é simultaneamente o melhor possível para todos os residentes.

A contrapartida do algoritmo RGS, conhecida como Algoritmo de Gale/Shapley orientada a hospitais (HGS) oferece um casamento estável único que satisfaz otimamente aos hospitais.

Aplicações Usado em esquemas centralizados de interessados em vagas, por exemplo: residentes a hospitais, concluintes de segundo grau a universidades, etc. Talvez a mais conhecida aplicação seja o *National Resident Matching Program (NRMP)* nos EUA, o qual assinala cerca de 31.000 concluintes do curso de medicina a hospitais ao longo de todo o país. Sistemas similares existem no Canadá, Escócia e Japão. Também é usado na alocação de escolares em Nova York, recrutamento de universitários na França e admissão universitária na Espanha. **Um Exemplo**

```

Os hospitais sao:           J U R A
As capacidades deles sao: 1 1 2 3
A sequencia de alocao e: 2 5 7 6 4 3 1
As escolhas dos hospitais:
J - 2 5 7 6 3 1 0
U - 2 5 7 6 4 3 1
R - 5 7 4 3 1 0 0
A - 2 5 7 6 4 3 1

As escolhas dos residentes:
1- J R U A
2- A J U
3- U R A J
4- U A R
5- U R J A
6- U A J
7- U A J R

-- r=2
acabei de alocar o r=2 para h=A
-- r=5
acabei de alocar o r=5 para h=U
excluindo piores do que 5 no hospital U
-- r=7
acabei de alocar o r=7 para h=A
-- r=6
acabei de alocar o r=6 para h=A
excluindo piores do que 6 no hospital A
-- r=4
acabei de alocar o r=4 para h=R
-- r=3
acabei de alocar o r=3 para h=R
excluindo piores do que 3 no hospital R
-- r=1
acabei de alocar o r=1 para h=J
excluindo piores do que 1 no hospital J
0 resultado final foi:
2-A 5-U 7-A 6-A 4-R 3-R 1-J

```

Para você fazer

```

Os hospitais sao:           D L Z Q P G X R
As capacidades deles sao: 1 2 1 1 1 1 2 3
Seq: 6 3 7 4 1 11 5 10 2 8 9
As escolhas dos hospitais:
D - 6 7 4 1 11 5 10 2 8 9 0
L - 3 7 4 1 11 5 10 8 9 0 0
Z - 3 7 4 1 11 5 10 2 8 9 0
Q - 6 3 7 4 1 11 5 10 2 8 9
P - 6 3 7 4 1 5 10 2 8 0 0
G - 6 3 4 1 11 10 2 8 9 0 0
X - 6 3 11 5 10 2 8 9 0 0 0
R - 6 3 7 4 11 5 10 8 9 0 0

As escolhas dos residentes:
1- G Q Z P L D
2- Q G Z X D P
3- G X Z L P R Q
4- P G L R Q D Z
5- D Q P R Z L X
6- X D R G Q P
7- D R L Z Q P
8- R X D L G Q P Z
9- Z Q R X D G L
10- D G Z Q P R L X
11- X L Q Z G D R

```

Resolva a instância acima do problema dos residentes e dos hospitais, usando o algoritmo de Gale-Shapley e obtenha os hospitais de todos os residentes. Responda abaixo os hospitais para os residentes

11	5	10	9



Dividir para conquistar: o torneio de tênis

Seja o problema de organizar um torneio de tênis com n jogadores. Neste torneio cada jogador deve jogar uma vez contra cada um dos seus $n - 1$ competidores. Além disso deve jogar uma vez por dia, tendo no máximo um dia de folga em todo o torneio. Por exemplo, as seguintes tabelas são possíveis para torneios com 5 ou 6 jogadores

J	d1	d2	d3	d4	d5	J	d1	d2	d3	d4	d5
1	2	3	4	5	-	1	2	3	4	5	6
2	1	5	3	-	4	2	1	5	3	6	4
3	-	1	2	4	5	3	6	1	2	4	5
4	5	-	1	3	2	4	5	6	1	3	2
5	4	2	-	1	3	5	4	2	6	1	3
						6	3	4	5	2	1

Para resolver este problema consideramos que se n é par, o fim do torneio termina em $n - 1$ dias. Se n é ímpar, o torneio acaba em n dias. O caso mais simples é $n = 2$, cuja solução é fácil (um contra o outro). Se $n > 2$, usa-se a técnica da divisão e conquista. Vai-se supor já calculada a metade superior da tabela. Portanto, já está preenchido o quadrante superior esquerdo. Agora, os outros 3 quadrantes não são difíceis de preencher:

1. O quadrante inferior esquerdo deve enfrentar os jogadores de número superior entre eles, o que se obtém somando $n/2$ aos valores do quadrante superior esquerdo.
2. O quadrante superior direito enfrenta os jogadores com maiores e menores números. É preenchido fazendo os jogadores 1 a $n/2$ jogarem contra $(n/2) + 1$ a n respectivamente no dia $n/2$ e depois rodando os valores $(n/2) + 1$ a cada dia.
3. Analogamente, o quadrante inferior direito enfrenta os de maior número com os de menor número. Obtém-se colocando os jogadores $(n/2) + 1$ até n contra os jogadores 1 até $(n/2)$ no dia $n/2$ e depois rodando os valores 1 a n a cada dia, mas em sentido contrário a como foi feito no quadrante superior direito.

Algoritmo Criada a matriz "tabela" contendo o tamanho correto, o algoritmo é

```

1: função torneio (n : inteiro)
2: jog, dia, m : inteiro
3: se n=2 então
4:   tabela[1][1] ← 2
5:   tabela[2][1] ← 1
6: senão
7:   se (n MOD 2) ≠ 0 então
8:     torneio (n+1)
9:     para jog ← 1 até n faça
10:      para dia ← 1 até n faça
11:       se tabela[jog][dia] = n+1 então
12:         tabela[jog][dia] ← 0
13:       fim se
14:     fim para
15:   fim para
16:   senão
17:     m ← n DIV 2
18:     torneio m
19:     se (m MOD 2) = 0 então
20:       para jog ← m+1 até n faça
21:         para dia ← 1 até m-1 faça
22:           tabela[jog][dia] ← tabela[jog-m][dia] + m
23:         fim para
24:       fim para
25:       para jog ← 1 até m faça
26:         para dia ← m até n-1 faça
27:           se jog+dia ≤ n então
28:             tabela[jog][dia] ← jog+dia
29:           senão
30:             tabela[jog][dia] ← jog+dia-m
31:           fim se
32:         fim para
33:       fim para
34:       para jog ← m+1 até n faça
35:         para dia ← m até n-1 faça
36:           se jog > dia então
37:             tabela[jog][dia] ← jog-dia
38:           senão
39:             tabela[jog][dia] ← (jog+m)-dia
40:           fim se
41:         fim para
42:       fim para
43:     senão
44:       para jog ← m+1 até n faça
45:         para dia ← 1 até m faça
46:           se tabela[jog-m][dia] = 0 então
47:             tabela[jog][dia] ← 0
48:           senão
49:             tabela[jog][dia] ← tabela[jog-m][dia] + m
50:           fim se

```

```

51:   fim para
52: fim para
53: para jog ← 1 até m faça
54:   para dia ← 1 até m faça
55:     se tabela[jog][dia] = 0 então
56:       tabela[jog][dia] ← jog+m
57:       tabela[jog+m][dia] ← jog
58:     fim se
59:   fim para
60: fim para
61: para jog ← 1 até m faça
62:   para dia ← m+1 até n-1 faça
63:     se (jog+dia) ≤ n então
64:       tabela[jog][dia] ← jog+dia
65:     senão
66:       tabela[jog][dia] ← jog+dia-m
67:     fim se
68:   fim para
69: fim para
70: para jog ← m+1 até n faça
71:   para dia ← m+1 até n-1 faça
72:     se jog > dia então
73:       tabela[jog][dia] ← jog-dia
74:     senão
75:       tabela[jog][dia] ← (jog+m)-dia
76:     fim se
77:   fim para
78: fim para
79: fim se
80: fim se
81: fim se
82: fimfuncao

```

Alguns exemplos

+-----							
1	2						
2	1						
	1	2	3				
+-----							
1	2	3	0				
2	1	0	3				
3	0	1	2				
	1	2	3				
+-----							
1	2	3	4				
2	1	4	3				
3	4	1	2				
4	3	2	1				
	1	2	3	4	5	6	7
+-----							
1	2	3	4	5	6	7	0
2	1	4	3	6	7	0	5
3	4	1	2	7	0	5	6
4	3	2	1	0	5	6	7
5	6	7	0	1	4	3	2
6	5	0	7	2	1	4	3
7	0	5	6	3	2	1	4
	1	2	3	4	5	6	7
+-----							
1	2	3	4	5	6	7	8
2	1	4	3	6	7	8	5
3	4	1	2	7	8	5	6
4	3	2	1	8	5	6	7
5	6	7	8	1	4	3	2
6	5	8	7	2	1	4	3
7	8	5	6	3	2	1	4
8	7	6	5	4	3	2	1



Para você fazer Suponha a geração de uma tabela com 96 jogadores. Pergunta-se qual a sequência de chamadas recursivas à função torneio.

resposta



Esta folha está baseada no excelente livro BHAR-GAVA, Aditya. **Entendendo Algoritmos** - um guia ilustrado para programadores e outros curiosos. São Paulo: Novatec, 2017.

Como se mede um algoritmo ?

Um algoritmo consome 2 recursos: tempo de processador e memória. A análise tradicionalmente recai sobre o tempo. Como a demora de um programa depende da linguagem, computador, plataforma, SO, etc, busca-se aqui uma descrição matemática que independa de aspectos de implementação. Até porque, usualmente quer se discutir desempenho ANTES de implementar o algoritmo, isto é, ele ainda não existe.

Esta teoria atende pelo nome (inadequado) de Teoria da Complexidade. Aqui, complexidade é sinônimo de desempenho. Busca-se uma família de funções que descrevam o tempo em função do tamanho da entrada. Para estudar a demora antes do algoritmo ser criado, as simplificações são 2:

- Só se contam as instruções dentro de loops
- Cada instrução demora 1 unidade de tempo, independente do que fizer.

Grosso modo, eis os casos mais comuns para uma entrada de tamanho *n*

complexidade	explicação
$O(k)$	o algoritmo demora um tempo constante, independente do tamanho da entrada
$O(n)$	o algoritmo demora proporcionalmente ao tamanho da entrada
$O(n^2)$	proporcional ao quadrado da entrada
$O(n^3)$	proporcional ao cubo da entrada
$O(k^n)$	exponencial em relação a entrada
$O(\log_2 n)$	proporcional ao logaritmo da entrada
$O(n.\log_2 n)$	a entrada vezes o logaritmo da entrada

Alguns exemplos: $O(n)$ =leitura e impressão de um arquivo; $O(n^2)$ = classificação bolha, classificação por seleção, menor caminho (dijkstra); $O(n^3)$ =menor caminho (floyd-warshall); $O(k^n)$ =caixeiro viajante; $O(\log_2 n)$ =busca em uma árvore binária, busca binária (veja a seguir); $O(n.\log_2 n)$ =quick sort.

Busca Binária

A busca binária é um algoritmo que busca encontrar um elemento em uma lista. A lista deve estar ordenada, e se o elemento pertence à lista, a resposta é o seu índice (ou seja: em que lugar da lista este elemento aparece). Se ele não pertence a resposta é um índice inválido que sinaliza este fato. Nós usamos um algoritmo parecido quando procuramos uma palavra num dicionário, ou olhamos uma lista de aprovados para ver se estamos lá, ou em buscas similares.

Vamos estudar o algoritmo. Suponha que você tem que descobrir um número que eu inventei, entre 1 e 1000. Depois de cada chute eu responderei: acertou, mais alto ou mais baixo, dependendo de onde estiver o número que eu inventei em relação ao seu chute.

Se você começar em 1, 2, 3 e assim por diante, a cada palpite apenas um número é excluído e se eu pensei em um número próximo de 1000 serão quase 1000 tentativas.

A busca binária começa chutando o meio do intervalo: 500. Se eu responder “acima”, de cara são eliminados 500 candidatos. O próximo chute deve ser 750, chute este que elimina mais a metade do restante. De metade em metade chega-se ao número escolhido em no máximo 10 tentativas. É muito melhor do que as quase 1000 da estratégia anterior.

De um modo geral a busca binária gasta $\log_2 N$ tentativas, onde *N* é o tamanho da lista. Você pode ter esquecido o que é o logaritmo, mas lembre: Em $\log_x Y = ?$ a pergunta a responder é quantas vezes *x* deve ser multiplicado por *x* até chegar ao valor *Y*. Daqui que $\log_2 1024 = 10$ já que 2 deve ser multiplicado por 2 10 vezes para chegar a 1024. Acompanhe o algoritmo em Python

```
def pesqbin(lista,item):
    baixo=0
    alto=len(lista)-1
    while baixo <= alto:
        metade = (baixo+alto)//2
        chute=lista[metade]
        if chute==item:
            return meio
        if chute>item:
            alto=meio-1
        else:
            baixo=meio+1
    return -1
```

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade. Na alocação sequencial:

1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrem. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes.

Na alocação encadeada:

1. Os dados estão em qualquer lugar da memória. Para chegar no próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas.

Recursão

A recursão é um processo de repetir um procedimento sucessivas vezes até resolver o problema. A novidade é que o procedimento é sempre o mesmo (embora com parâmetros cada vez menores). É odiado por muitos programadores, mas adorado por outros tantos. No paradigma funcional (Haskell, Lisp, APL,...) é praticamente o único mecanismo de repetição empregado. Um algoritmo recursivo sempre pode ser reescrito sem recursão (o que chamamos interativo), mas com a palavra Leigh Caldwell do *Stack Overflow: loops podem melhorar o desempenho do seu programa. A recursão melhora o desempenho do seu programador. Escolha o que for mais importante para a sua situação.*

Um programa recursivo tem 3 características:

1. chama a si próprio

2. apresenta um caso básico, quando não há recursão

3. na chamada recursiva, a cada vez o universo é menor, em direção ao caso básico

veja-se um exemplo de uma função escrita na forma recursiva e na interativa

```
def fati(x):
    r=1
    while x>1:
        r=r*x
        x=x-1
    return r

def fatr(x):
    if x>1:
        return x*fatr(x-1)
    else:
        return 1
```

Dividir para conquistar

Suponha que você tem uma grande área de terra retangular para construir canteiros de legumes. Cada canteiro precisa estar em uma área quadrada do maior tamanho possível. Qual a solução ótima ? D-C é uma possível resposta. Como a técnica é sempre recursiva, vamos buscar o caso base: pode ser quando a dimensão maior é múltipla da menor. Nesta hora divide-se o terreno em quadrados e está resolvido o problema. Enquanto este caso base não aparecer vai-se recortar um quadrado de aresta igual à menor aresta e rechamar o código. Acompanhe o algoritmo em Python

```
def dcfazenda(x,y):
    print("gerará "+str(x//y)+" modulos de "+str(y)+" por "+str(y))
    if x%y==0:
        return
    else:
        return dcfazenda(y,x%y)
```

Quicksort

É o algoritmo campeão de desempenho para ordenar listas. Lembra-se da importância da lista estar ordenada na busca binária ? Aliás, em francês um computador chama *ordinateur* e em espanhol *ordenador*. Escolhido um elemento aleatório da lista aqui chamado de *pivot* a lista é subdividida em duas sub-listas: a dos elementos menores do que o pivot e outra sublista dos elementos maiores do que ele. O algoritmo é chamado recursivamente para cada uma das sublistas e no final a resposta é a sublista menores (já ordenada) + pivot + sublista maiores (já ordenada). Acompanhe o algoritmo

```
def quick(lis):
    if len(lis)<2:
        return lis
    else:
        pivot=lis[0]
        menor=[i for i in lis[1:] if i<=pivot]
        maior=[i for i in lis[1:] if i>pivot]
        return quick(menor)+[pivot]+quick(maior)
```

No caso médio este algoritmo é muito rápido. Ele só deixa de sê-lo no caso particular de que o vetor já esteja (quase) ordenado. Neste caso o pivot (quase) sempre vai ser o menor elemento e a cada recursão o vetor resultante só será uma unidade menor. Daí que normalmente escolhe-se como pivot alguém do meio do vetor original. O comando melhor seria: `pivot=lista[len(lista)//2]`.

📖 Para você fazer

Conduza uma pesquisa na Internet e faça um resumo pessoal dos seguintes algoritmos/sites:

1. Troca de Chaves Diffie-Hellman
2. Algoritmo HyperLogLog
3. Algoritmo Flajolet-Martin
4. Algoritmo Hash e SimHash (um hash que tem sensibilidade local)
5. o site better explained
6. o algoritmo de similaridade do cosseno
7. o algoritmo da maior subsequência comum
8. algoritmo da mochila (aplicação do alg. guloso)



Esta folha está baseada no excelente livro BHAR-GAVA, Aditya. **Entendendo Algoritmos** - um guia ilustrado para programadores e outros curiosos. São Paulo: Novatec, 2017.

Tabelas Hash

Uma busca linear demora $O(n)$, uma busca binária demora $O(\log_2 n)$. Que tal seria ter uma busca que demorasse $O(1)$? Bem melhor, não é ? Esta busca existe e é feita usando-se uma tabela hash. Uma tabela hash usa como índice o resultado de uma função hash. Esta função recebe um valor (o item a ser procurado) e devolve o seu índice na lista original. Tal função deve ter 2 requisitos:

- ser consistente: cada vez que o mesmo item for consultado ela deve devolver o mesmo valor
- itens diferentes devem gerar índices diferentes

Como o espaço de chaves (itens) sempre é muito maior do que o espaço de índices, deve-se pensar no que fazer quando ocorrer a **colisão**. Trata-se do que acontece quando dois itens diferentes são mapeados para o mesmo índice. Na linguagem Python, tabelas hash são chamadas de dicionários, e usando-os você nunca terá que se preocupar com as colisões: é problema do Python. Python reconhece dicionários pelo uso de chaves {}. Acompanhe o exemplo: seja criar uma lista de preços de itens de papelaria

```
precos = {}
precos["folha"]=0.03
precos["lapis_b2"]=2.20
precos["grampeador"]=18.90
precos["borracha"]=5.50
```

```
print(precos)
```

A consulta a um item é feita fazendo >>> precos["folha"] e ele responde 0.03. Já a consulta a um item inexistente como >>> precos["cartolina"] gera um KeyError. Se quiser testar a existência antes de deixar acontecer o erro use

```
if precos.get("cartolina"):
    print("existe este item")
else:
    print("nao existe este item")
```

Não vai se explicar aqui como gerar a função hash, já que o Python (e a maioria das linguagens modernas) faz isso para você. Se quiser, busque este tópico em qualquer livro de estruturas de dados. Ou no VIVX521. É poesia (matemática) pura.

Busca em Largura

Grafos

Um grafo é um objeto matemático composto por pontos (vértices) e conexões (arestas). Qualquer mapa pode ser entendido como um grafo, senão vejamos:

tipo	vértice	aresta
mapa rodoviário	cidade	estrada
rede computadores	computador	link
amigos do FB	usuário	amizade
curso de engenharia	disciplinas	pré-requisitos
família	pessoas	ascendência
fluxograma	atividades	precedências

Um vértice pode estar isolado, conectado a um outro vértice, a muitos vértices ou a todos os vértices. A conexão pode ser unidirecional (de *a* para *b*) ou bidirecional. Neste último caso, a bidirecionalidade pode ser entendida como havendo os dois sentidos. Ou seja a conexão de *a* e *b* sem sentido, pode ser vista como tendo os dois sentidos (de *a* para *b* e de *b* para *a*).

Nosso primeiro algoritmo de grafos é saindo de um vértice, percorrer todos os vértices do grafo. Se

houver vértices isolados, este algoritmo não tem solução. Se cada vértice só estiver conectado a outros 2 pode haver uma solução trivial. Mas nossa tarefa é resolver o caso mais geral: cada vértice tem muitas conexões.

Existem diversas estratégias, mas vai-se resolver aqui a chamada busca (ou caminhamento – são sinônimos) em largura, cuja descrição é: *saindo de um vértice, visite todos os vértices a ele conectados, antes de passar a outro vértice. Repita isto até ter visitado todos os vértices do grafo.*

Algoritmo de Dijkstra

No algoritmo de Dijkstra (lê como Dícter) as arestas do grafo contêm valores não negativos (preço, demora, vazão, capacidade, disponibilidades...). Este tipo de grafo é chamado grafo valorado. O algoritmo responde a pergunta: saindo de um vértice qual o caminho “mais barato” para chegar a algum outro vértice ? O algoritmo começa com uma lista dos vizinhos do vertice de saída com os custos associados a cada vizinho. Agora o algoritmo se desloca a cada vizinho e refaz essa busca. Se algum vizinho do vizinho tiver custo menor para ser alcançado esse custo é atualizado (para baixo). Quando o vértice final for alcançado o algoritmo acabou. Note-se que no grafo não podem haver arestas negativas nem ciclos.

Na implementação abaixo, usou-se uma matriz de custos. É uma matriz quadrada de ordem igual ao número de vértices do grafo. Nela, o valor $C[i, j]$ corresponde ao custo para ir do nodo *i* ao nodo *j*.

```
def dijk(t,ini,fim):
    precede=numpy.zeros((len(t)))
    perm=numpy.zeros((len(t)))
    dist=numpy.zeros((len(t)))
    for i in range(len(dist)):
        dist[i]=9999
    perm[ini]=1
    dist[ini]=0
    corrente=ini
    while corrente != fim:
        menordist=9999
        dc=dist[corrente]
        for i in range(len(dist)):
            if perm[i]==0:
                novadist=dc+t[corrente][i]
                if novadist<dist[i]:
                    dist[i]=novadist
                    precede[i]=int(corrente)
                if dist[i]<=menordist:
                    menordist=dist[i]
                    k=i
            corrente=k
        perm[corrente]=1
    print(dist[fim])
    g=fim
    while ini !=g:
        print(int(g))
        g=precede[int(g)]
    print(ini)
import numpy
t=numpy.array([[9999,11,21,44,20,5,43,42,9],
               [8,9999,23,55,51,32,28,3,27],
               [2,18,9999,54,47,49,59,35,4],
               [17,26,10,9999,9,4,14,57,11],
               [7,13,29,16,9999,22,31,33,12],
               [1,2,1,38,10,9999,15,22,50],
               [53,6,16,18,34,8,9999,46,36],
               [13,39,45,3,7,5,48,9999,17],
               [19,40,30,25,52,20,24,41,9999]])

print(t)
dijk(t,0,3)
```

Algoritmo Guloso

Existem muitos problemas na ciência da computação cuja solução exata só é possível para instâncias muito pequenas do problema. No jargão são conhecidos como “problemas NP-completos”. Um famoso é o problema do caixeiro viajante (*um caixeiro deve sair de uma cidade, visitar um conjunto de outras cidades e retornar à primeira, gastando o menor trajeto possível*). Para instâncias maiores é impossível resolver exatamente o problema. Nestes casos, precisa-se usar algum tipo de aproximação. Uma muito boa atende pelo nome de “estratégia gulosa”que pode ser jocosamente descrita como *em cada momento, abocanhe o maior pedaço*. Seja o seguinte problema: Você quer inaugurar um programa de rádio e quer atingir todos os estados brasileiros. Há um conjunto de rádios, cada uma com seus custos e sua área de abrangência. A pergunta

é: qual o menor (mais barato) conjunto de rádios que atinge todos os estados brasileiros ? Veja uma instância possível do problema:

Radio	Estados que abrange	Custo
WQ1	PR, SC, SP, RS	1000/minuto
WQ2	SP, RJ	800/minuto
WQ3	MG, SP, RJ, ES, BA	1100/minuto
...		

Note que o problema está na sobreposição: praticamente todas as rádios transmitem para SP (o maior mercado) e é essa sobreposição que complica a solução. O algoritmo guloso neste caso fica:

1. Localize a rádio que abranja o maior número de estados ainda não cobertos. Não há problema se esta rádio atingir estados já cobertos.
2. Adicione esta rádio ao conjunto mínimo
3. repita o processo até ter todos os estados cobertos no conjunto mínimo.

Este algoritmo pode não achar a melhor resposta, mas sempre acha uma boa resposta e demora muito menos que o algoritmo exato. Compare os tempos

número de estações	alg. exato $O(n!)$	alg. guloso $O(n^2)$
5	3.2 seg	2.5seg
10	102 seg	10 seg
32	13 anos	102 seg
100	4×10^{21} anos	16 minutos

👉 Para você fazer

Conduza uma pesquisa na Internet e faça um resumo pessoal dos seguintes algoritmos/sites:

1. Troca de Chaves Diffie-Hellman
2. Algoritmo HyperLogLog
3. Algoritmo Flajolet-Martin
4. Algoritmo Hash e SimHash (um hash que tem sensibilidade local)
5. o site better explained
6. o algoritmo de similaridade do cosseno
7. o algoritmo da maior subsequência comum
8. algoritmo da mochila (aplicação do alg. guloso)



Esta folha está baseada no excelente livro BHAR-GAVA, Aditya. **Entendendo Algoritmos** - um guia ilustrado para programadores e outros curiosos. São Paulo: Novatec, 2017.

Programação Dinâmica

A idéia aqui é ir resolvendo sub-problemas e depois juntar essas soluções na solução do problema geral. Ela é útil quando você tenta otimizar algo em relação a um limite. Esta técnica permite construir algoritmos que resolvem problemas computacionais, sobretudo na área de otimização combinatória. Tais problemas precisam ter 2 características para poderem usar a técnica:

- subestrutura ótima: quando a solução ótima contém em seu interior soluções ótimas para subproblemas e
- superposição de problemas: ocorre quando o algoritmo de força bruta reexamina o mesmo problema muitas vezes.

Exemplos de problemas que podem ser resolvidos usando programação dinâmica: multiplicação de séries de matrizes, problema da mochila, otimização combinatória, etc. Os problemas desta folha são os problemas 67, 81 e 82 do projeto Euler.

Maior soma em pirâmide

Suponha uma matriz como

```
3
7 4
2 4 6
8 5 9 3
```

Precisa-se calcular a soma máxima começando no vértice superior e descendo sempre para os 2 números adjacentes. Neste caso é fácil ver que a soma máxima é $3+7+4+9=23$. Se a matriz tiver 15 linhas (e 15 números na última linha), já serão 16384 possíveis caminhos: o computador ainda os percorre muito rápido. Já em uma matriz de 100 linhas, se seu programa testar um trilhão de caminhos por segundo, serão necessários vinte bilhões de anos para testar todas as possibilidades.

Começando no alto do triângulo acima (no 3), deve-se escolher ir para a direita ou a esquerda. Para decidir isto, precisa-se resolver 2 problemas que são:

```
7          4
2 4      e 4 6
8 5 9      5 9 3
```

Pode-se ir quebrando a série de problemas até a última linha, cuja resposta é evidente: é o número que está lá. Resolvido, pode-se retornar para cima, usando a regra

a
b c

$a + \max(b, c)$. Entretanto, ao quebrar o problema original em 2, e cada um deste em outros 2, fica-se com 4 problemas. (As pirâmides: 7,2,4; 4,4,6 se quebram em 2,8,5; 4,5,9 e em 4,5,9; 6,9,3). Note que há problemas que se repetem (4,5,9 no exemplo). Ora, eles vão ter o mesmo resultado não importando como se chegou a eles. Logo, basta calcular tais problemas uma vez, e aqui a programação dinâmica mostra seu valor.

o algoritmo Começando de baixo para cima e aplicando a regra $a + \max(b, c)$, fica

```
3
7 4
2 4 6
8 5 9 3
```

fica

```
3
7 4
```

10 13 15

e depois

```
3
20 19
```

E finalmente 23, que é a resposta correta.

o algoritmo Neste caso o algoritmo é (origem=1):

```
1: leia(mat)
2: i ← número de linhas da mat
3: enquanto (i > 0)
4:   j ← 1
5:   enquanto (j ≤ i)
6:     mat[i][j] ← mat[i][j] + max(mat[i+1][j],
7:                                   mat[i+1][j+1])
8:     j++
9:   fim(enquanto)
10: i--
11: imprima mat[1][1]
```

Em Python:

```
def pd666(mat):
    import math
    print(mat)
    i=len(mat)-2
    while i>=0:
        j=0
        while j<=i:
            mat[i][j]=mat[i][j]+max(mat[i+1][j],
                                     mat[i+1][j+1])
            j=j+1
        i=i-1
    print(mat)
    return mat[0][0]
import numpy
a=numpy.array([[3,0,0,0],[7,4,0,0],
               [2,4,6,0],[8,5,9,3]])
print(pd666(a))
```

Árvores

Uma estrutura fundamental em praticamente toda a ciência da computação. Ela cria índices com desempenho logarítmico, (como a busca binária), mas sem a necessidade de prévia ordenação das chaves. Logo é muito melhor. Cria índices em qualquer banco de dados como Oracle, Sybase, MySQL, DB2 etc etc. Implementa algoritmos de compressão (Huffman, LZW e praticamente todos os demais). Serve como base para algoritmos de correção ortográfica (Tries e árvores Patricia).

Vai-se resolver aqui um problema de compressão de dados usando o algoritmo de Huffman. São 3 versos de Os Lusíadas devidamente comprimidos, usando a árvore

```
[[0,-81,-74],[1,-66,-88],[2,-72,-86],[3,-71,-85],
[4,-45,-67],[5,1,0],[6,-84,2],[7,-76,-80],
[8,-77,-82],[9,-68,-78],[10,3,-79],[11,5,4],
[12,-73,6],[13,-83,7],[14,9,8],[15,11,10],
[16,12,-69],[17,13,-47],[18,14,-65],
[19,16,15],[20,18,17],[21,20,19]]
```

Na qual os números negativos indicam um caracter ASCII (assim -65 equivale a A) e os números positivos indicam cursores para o nodo da árvore que é apontado.

Esta árvore gera o seguinte dicionário de compressão:

```
D00000; N00001; M00010; R00011; A001; S0100;
L01010; P01011; /011; I1000; T10010; H100110;
V100111; E101; B110000; X110001; Q110010;
J110011; -11010; C11011; G11100; U11101 e O1111
```

Se o string a descomprimir for (já em binário)

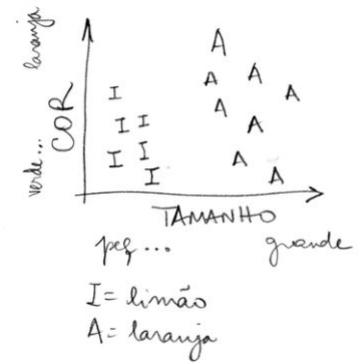
```
1010111100111101010101110000100011101100
00100011110010111010010101001110101100010
0011000010001110111000111011010101010100
00110010101110100101100101000100001000110
...
```

O resultado será (o começo apenas...)

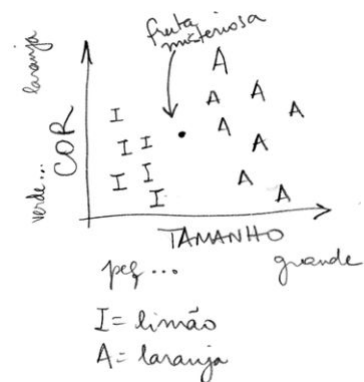
E/JULGAREIS/QUAL/E/MAIS/EXCELE...

k-vizinhos mais próximos

Olhe para uma fruta: é uma laranja ou um limão? Sabe-se que uma laranja é mais amarelada e maior do que um limão. Chamando A de laranja e I de limão, pode-se desenhar um gráfico a partir da cor e do tamanho. Ficaria Assim:



Olhando o gráfico quanto mais alaranjada e maior for uma fruta mais ela é uma laranja. Mas, e se aparecer uma fruta misteriosa, o que ela será?



A resposta é o algoritmo dos k-vizinhos. Basta procurar um número k de vizinhos à fruta misteriosa. A vizinhança é dada pelo teorema de Pitágoras e pela Geometria Analítica. Mas, pode-se usar qualquer outra métrica. A resposta é que se a maioria dos vizinhos da fruta misteriosa for uma laranja, provavelmente ela será uma laranja. Este procedimento simples é uma *classificação* e ele é usado em Machine Learning por exemplo, para o Netflix dar uma recomendação do filme que você deve assistir, se estiver sem inspiração para escolher.

Antes de encerrar uma lembrança: o exemplo acima usou 2 dimensões para poder representar os pontos num espaço bidimensional. Entretanto, na prática costumam-se usar muitas mais dimensões: é comum 10, 15 ou 20 dimensões. A beleza da matemática é que tais espaços multidimensionais são reais na matemática.

Para você fazer

Conduza uma pesquisa na Internet e faça um resumo pessoal dos seguintes algoritmos/sites:

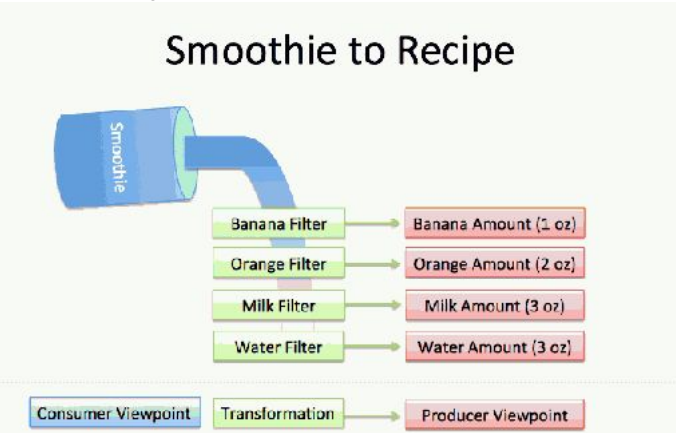
1. Troca de Chaves Diffie-Hellman
2. Algoritmo HyperLogLog
3. Algoritmo Flajolet-Martin
4. Algoritmo Hash e SimHash (um hash que tem sensibilidade local)
5. o site better explained
6. o algoritmo de similaridade do cosseno
7. o algoritmo da maior subsequência comum
8. algoritmo da mochila (aplicação do alg. guloso)



Esta folha está baseada no excelente livro BHARGAVA, Aditya. **Entendendo Algoritmos** - um guia ilustrado para programadores e outros curiosos. São Paulo: Novatec, 2017.

Transformada de Fourier

Fourier foi um cientista francês que acompanhou Napoleão na invasão do Egito e depois ficou por lá. É um grande matemático e físico. Trabalhou muito no conceito de calor e é um dos suportes da Revolução Industrial. A transformada de Fourier é *um dos raros algoritmos que conseguem ser brilhantes, elegantes e ter milhares de formas de utilização* (BHA17). A melhor analogia para o algoritmo vem do **Better Explained**: dada uma vitamina de frutas, a transformada de fourier consegue transformá-lo em mamão, maçã, açúcar, sorvete etc. Ele é usado no algoritmo MP3 para transformar uma música em suas frequências individuais (e daí o MP3 exclui as frequências que são irrelevantes, por inaudíveis, por isso ele consegue comprimir o som). O compressor de imagens JPG faz a mesma coisa, mas com cores e regiões. Idem para o aplicativo Shazam que descobre qual é a música que está tocando, já que ele compara as frequências fundamentais da música e não a música em si. Veja a explicação da vitamina retirada do site <https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/>

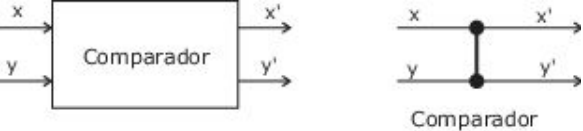


Algoritmos Paralelos

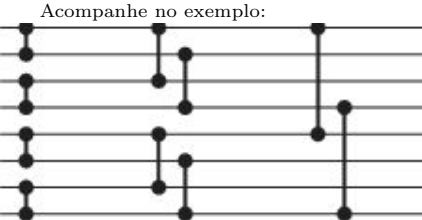
Como já se viu, o melhor algoritmo de ordenação tem custo $O(n \times \log_2 n)$. O que não foi dito naquela ocasião, mas que é verdade, é que este custo é para ambientes mono-processados, ou seja aqueles nos quais as operações são seriais, só é possível fazer uma operação de cada vez. Suponha agora que você dispõe de um sistema que permita processamento paralelo (diversas operações a cada ciclo). Qual seria um bom algoritmo para obter uma ordenação mais rápida ? Trata-se do conceito de redes de ordenação. Usando-as, pode-se criar um algoritmo de ordenação de n elementos com custo $O(\log^2 n)$. Para relembrar o significado dessas fórmulas, preencha:

se a entrada tiver	$O(n \times \log_2 n)$	$O(\log^2_2 n)$
10		
100		
1000		
1000000		

Uma rede de comparações é um conjunto de fios e de comparadores. Um comparador é um dispositivo com duas entradas (x, y) e duas saídas (x', y') que executa as seguintes funções: $x' = \min(x, y)$ e $y' = \max(x, y)$. Um comparador pode ser apresentado como uma caixa (à moda LISP), ou mais praticamente como uma única linha vertical.



No desenho acima, as entradas aparecem à esquerda do comparador e as saídas à direita. Cada comparador opera com tempo $O(1)$, ou seja o tempo gasto entre a chegada das entradas e a obtenção das saídas é uma constante. Uma rede de ordenação é um conjunto de comparadores interconectados por fios, de tal maneira que a saída de um comparador pode ser a entrada de outro.



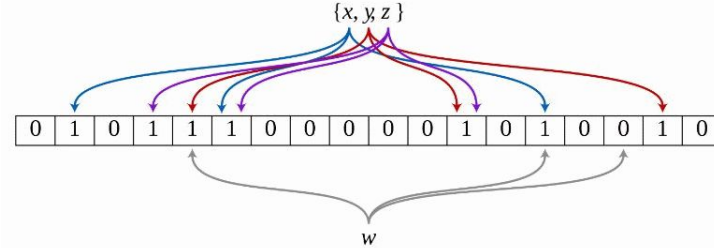
A rede de comparações acima, consegue responder qual é o maior e o menor elementos de uma entrada contendo 8 valores, usando apenas 3 comparações. Lembrando, 3 é o $\log_2 8$. Faça um chinês com os valores 2, 7, 1, 9, 3, 6, 4 e 8. Existe um algoritmo paralelo que está se tornando muito popular, principalmente com a popularização de CPUS multi-núcleo. Trata-se do algoritmo *map-reduce* (disponível no framework livre Apache Hadoop). É composto de duas partes: a função map e a função reduce. A primeira (map) pega uma função qualquer e um array e aplica esta função a cada elemento do array. A segunda, pega um array e transforma-o em um único número aplicando aos elementos do array uma dada função. Quando você tiver um conjunto de dados com bilhões de linhas o MapReduce pode fornecer uma resposta em minutos enquanto um BD tradicional pode levar horas.

Filtros de Bloom

Uma estrutura de dados probabilística, inventada por Burton Bloom em 1970, que é usada para verificar se um elemento é membro de conjunto de maneira muito rápida e eficaz. O falso positivo (achar que o elemento é membro, mas ele não é) é possível – embora com probabilidade tão baixa quanto se queira, mas o falso negativo (achar que o elemento não é membro e ele é) nunca ocorre. Sua construção usa os seguinte artefatos:

- Um conjunto $x_1, x_2, \dots, x_n$ de elementos de um universo X .
- Um vetor S de m bits originalmente todos valendo 0.
- Um conjunto de k funções hash, todas diferentes $h_1, h_2, \dots, h_k$ para as quais dado um valor X devolvem um valor no domínio $(1, \dots, m)$ ou seja uma posição no vetor S .

Para agregar um elemento novo de X aplica-se a ele todas as funções k e o resultado dessas funções é tornado 1 em S . Para buscar se um novo elemento pertence ou não a X , a busca retorna “SIM” se todas as funções consultadas possuem um 1 em S . Da maneira como o filtro é construído, é impossível eliminar um elemento. A solução pode ser um segundo filtro de Bloom contendo os eliminados. Veja no exemplo a seguir o funcionamento de um filtro de Bloom (tirado da wikipedia)



Aqui está o conjunto $\{x, y, z\}$. As flechas mostram as posições no vetor de bits dos resultados após a aplicação das funções de hash a cada um dos elementos. O candidato w não está no conjunto $\{x, y, z\}$ porque ele tem um ou mais valores de hash valendo 0.

Para você fazer

Conduza uma pesquisa na Internet e faça um resumo pessoal dos seguintes algoritmos/sites:

1. Troca de Chaves Diffie-Hellman
2. Algoritmo HyperLogLog
3. Algoritmo Flajolet-Martin
4. Algoritmo Hash e SimHash (um hash que tem sensibilidade local)
5. o site better explained
6. o algoritmo de similaridade do cosseno
7. o algoritmo da maior subsequência comum
8. algoritmo da mochila (aplicação do alg. guloso)



Algoritmo dos k-vizinhos

Suponha que você é o programador do Netflix e quer implantar um supersistema de sugestão de filmes. Para isso, você vai criar e implantar o algoritmo conhecido como os k-vizinhos. A idéia é simples. Cada um dos filmes do catálogo vai receber uma nota entre o 0 (ruim) e 5 (excelente). Você não pode obrigar os usuários a dar notas. Então, você vai dar um mimo (um brinde ou um desconto) toda vez que o usuário atribuir essas notas a um filme que acabou de ver.

Mensalmente, você vai coletar todas as notas atribuídas aos filmes e vai calcular uma média da nota de cada filme. É essa nota que o algoritmo vai usar.

A nota vai colocar o filme em questão em um eixo ou dimensão e o nosso espaço de filmes vai ser hexa-dimensional. (6 dimensões). Note aqui a beleza da matemática: enquanto na física (no mundo, no universo) estamos limitados a 3 ou 4 dimensões e sabemos via Teorema de Pitágoras, calcular distâncias, nada nos impede de usar o mesmo teorema num espaço hexa-dimensional. Ninguém tem idéia de como seria no universo esse espaço, mas na matemática teórica ele está perfeitamente definido. Santa Matemática!

As dimensões atribuídas aos filmes serão:

riso Quanto comédia é este filme: se você riu muito, atribua 5, se você não riu nada, ou se mesmo tendo a pretensão de ser uma comédia o filme não te fez rir (ou fez chorar) atribua 0. Atribua notas proporcionais ao quesito.

filme cult Quanto famoso e cultuado o filme é. Ou você acha que é. Afinal é a sua opinião que está sendo pedida. Atribua notas proporcionais ao quesito.

atuação do ator principal Que nota você dá à atuação do personagem principal do filme. Atribua notas proporcionais ao quesito.

efeitos especiais adequados Quanto bem inseridos no filme estão os efeitos especiais, se ele existem. Atribua notas proporcionais ao quesito.

drama Como é construído o drama do filme. Atribua notas proporcionais ao quesito.

romance Qual a nota para romance que o filme merece. Atribua notas proporcionais ao quesito.

No arquivo anexo, existem 1000 filmes, identificados por um código numérico, e seguido pelas 6 notas médias atribuídas no último mês pelos usuários do serviço.

Do ponto de vista matemático, cada filme destes define um ponto no espaço de 6 dimensões.

Sugestão

Para que o sistema possa fazer uma recomendação adequada, o interessado deve dar o código do último filme de que ele gostou. Por hipótese é um dos códigos do acervo no arquivo. Agora, o seu algoritmo vai usar este filme de que o usuário gostou como padrão e vai calcular a distância hexadimensional de todos os 999 filmes restantes até o padrão. Feito isso, o algoritmo deve ordenar o acervo em ordem crescente por distância e devolver o segundo, terceiro e quarto filmes do acervo. Note que o primeiro terá distância 0 já que ele é o padrão. Por isso, ele ficou de fora da nossa relação.

Supondo que o filme k é o padrão. A distância de qualquer filme i para o filme k é dada pelo Teorema de Pitágoras expandido para 6 dimensões:

dik = sqrt(sum from m=1 to 6 of (xim - xkm)^2)

Note que o filme indicado pelo cliente é indicado pelo seu código, então antes de descobrir qual é o padrão, você precisa varrer o acervo para localizar o índice k do padrão.

Depois disso, pode aplicar o algoritmo acima.

Exemplo

Para testar o seu algoritmo, localize no lugar de sempre o arquivo EXEMPLO668. Ele é uma instância completa deste exercício. As primeiras 5 linhas do arquivo são

3	1.8	1.405	1.4125	4.02	3.195	2.3125
4	1.87	2.3625	4.55	0.655	4.35	1.9375
6	1.655	2.5975	0.1675	1.52	4.3375	1.545
7	0.7975	1.215	0.645	3.7	4.7	2.44
14	2.2425	4.295	4.475	4.8525	4.8525	2.92

Se o padrão escolhido for o filme da metade do arquivo (índice=500) o cálculo das distâncias vai gerar (apenas as primeiras 5 linhas)

3	1.8	1.405	1.412	4.02	3.195	2.312	4.4559
4	1.87	2.362	4.55	0.65	4.35	1.937	5.4483
6	1.65	2.597	0.167	1.52	4.337	1.545	4.6661
7	0.79	1.215	0.645	3.7	4.7	2.44	5.3738
14	2.24	4.295	4.475	4.85	4.852	2.92	4.4343

Finalmente, ordenando este segundo arquivo pela distância vai ficar

1543	4.19	4.55	1.607	2.927	3.24	4.05	0
1044	4.065	4.022	1.985	2.172	4.34	3.75	1.51
2495	4.707	4.06	1.697	2.305	2.345	3.187	1.56
418	3.742	3.225	1.602	3.71	3.137	4.097	1.60
181	3.7	4.182	1.437	4.362	3.555	4.24	1.61

O programa Python

```
def dist(x,y):
    su=(x[1]-y[1])**2
    su=su+(x[2]-y[2])**2
    su=su+(x[3]-y[3])**2
    su=su+(x[4]-y[4])**2
    su=su+(x[5]-y[5])**2
    su=su+(x[6]-y[6])**2
    su=su*0.5
    return su
def f668(arq,qual):
    import numpy
    bd = numpy.zeros((1000,8))
    f=open(arq,'r')
    for i in range(1000):
        ll = f.readline()
        ll = ll+" 0.0"
        #acrescenta a distancia (por enquanto é 0.0)
        bd[i] = [float(x) for x in ll.split()]
    for i in range(1000):
        bd[i][7]=dist(bd[qual],bd[i])
    bd=bd[bd[:,7].argsort()]
    # ordena a matriz pela coluna 7 - sort
    print (int(bd[1][0]),int(bd[2][0]),
           int(bd[3][0]))
    #imprime os filmes. o 0 é o próprio
    f668("c:/p/n/668/EXEMPLO668.mxd",499)
    #499 é o filme 500
```

Aplicações

Este algoritmo pode ser adaptado – sempre com o objetivo de fazer machine learning – a diversas situações possíveis

- 1. Na área médica visando estabelecer compatibilidade entre doador e receptor de algum órgão a transplantar. Os valores numéricos podem ser fatores laboratoriais ou clínicos importantes para a compatibilidade/rejeição. Aplicar o algoritmo neste universo de doadores potenciais vai levantar quais os mais próximos ao doador em todo o universo.
- 2. Na área comportamental para uma possível criação de um aplicativo de encontros amorosos. A expertise aqui deve se concentrar nas perguntas (critérios) a valorar e também na atribuição de valores para garantir match de interesses. A regra da distância eventualmente terá que ser reescrita.
- (a) idade ≡ regra de comparação criativa
- (b) sexo ≡ regra de comparação criativa
- (c) situação econômica ≡ regra de comparação criativa
- (d) peso/altura/quantidade de cabelo/características
- (e) cor de pele/diversão preferida/práticas apimentadas

- (f) gosto/disponibilidade para viagens/necessidade de sigilo
- (g) precisão/urgência, pretensão futura: tipos de vínculos

3. Na taxonomia animal, um zoólogo diante de uma espécie nova pode valorar características físicas (número de pernas, peso médio, número de olhos, articulações, apêndices...) e consultar fatias crescentes do reino animal estudando os resultados de cada consulta.

4. Estudando o histórico de compras de uma pessoa em um portal on line um aplicativo poderia sugerir novas compras. Por exemplo suponha valorar itens em um sapato feminino: altura, número, cores, fechado/aberto, de prender ou não,...

A cada dois meses, o aplicativo pode varrer o arquivo, recuperar itens encalhados, aplicar um desconto a eles, e indicá-los a uma compradora, que não vai resistir...

5. Carros, peças de espetáculos (disk-ingressos), livros, passagens aéreas, compras de supermercado (preço, portabilidade, tipo de item, nacional/importado, cara/barato), pratos em disk-entrega...

Para você fazer

O acervo que você vai consultar está no repositório com o nome de

F668D001.mxd

1. Processe o algoritmo considerando que o padrão escolhido foi o filme de código

2455

e informe aqui o código do filme das 3 sugestões do seu algoritmo achou

1	2	3

2. Processe o algoritmo considerando que o padrão escolhido foi o filme de código

2898

e informe aqui o código do filme das 3 sugestões do seu algoritmo achou

4	5	6

3. Processe o algoritmo considerando que o padrão escolhido foi o filme de código

2317

e informe aqui o código do filme das 3 sugestões do seu algoritmo achou

7	8	9



Algoritmos probabilísticos: filtro de Bloom

Os algoritmos probabilísticos vieram tratar um problema que corria o risco de ficar impossível de lidar: o crescimento exponencial das bases de dados e do tamanho das coisas a serem processadas. Por exemplo:

- Como garantir que um número de 1024 bits (2^{1024} =um número decimal com mais de 300 dígitos) seja primo. Apenas para decompor o número e garantir que ele não tem divisores, gasta-se o mais do que o tempo do universo (mais de 13 bilhões de anos).
- Como garantir que um documento recém publicado é uma versão única. Precisa-se compará-lo com bilhões de possíveis candidatos e a busca tem que ser feita byte a byte em cada um.

Uma família importante de algoritmos surge aqui: os algoritmos probabilísticos: eles usam atalhos sem precisar olhar todos os candidatos de forma exaustiva (força bruta). A contrapartida (“não existe almoço grátis”) é que eventualmente o resultado gerado pode ser incorreto.

A maneira de lidar com este problema é chamar o algoritmo várias vezes, usualmente, tantas quantas se queira: em geral a cada chamada a probabilidade do erro diminui.

Erros

Há que se definir aqui dois tipos de erros: o falso positivo e o falso negativo. Antes de estudar estes erros no âmbito da ciência da computação veja-se um exemplo trivial: ao pesquisar uma possível doença em uma pessoa, o falso positivo seria acusar a doença estando a pessoa sã. O falso negativo seria afirmar que a pessoa não tem a doença e ela está doente. Na ciência da computação, o falso positivo é quando ao estudar a pertinência de um elemento a um conjunto o algoritmo erra e afirma: o elemento PERTENCE ao conjunto e como ele não pertence este é um erro.

O falso negativo para o mesmo problema é afirmar que o elemento não pertence ao conjunto e ele – pelo contrário – pertence. Daí o erro.

Conforme o problema, um dos dois erros pode ser mais provável ou mesmo mais importante que o outro. Cada caso é um caso.

Filtro de Bloom

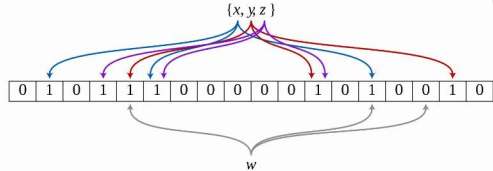
Uma estrutura de dados probabilística, inventada por Burton Bloom em 1970, que é usada para verificar se um elemento é membro de conjunto de maneira muito rápida e eficaz. O falso positivo (achar que o elemento é membro, mas ele não é) é possível – embora com probabilidade tão baixa quanto se queira, mas o falso negativo (achar que o elemento não é membro e ele é) nunca ocorre. Sua construção usa os seguinte artefatos:

- Um conjunto $x_1, x_2, \dots, x_n$ de elementos de um universo X .
- Um vetor S de m bits originalmente todos valendo 0.
- Um conjunto de k funções hash, todas diferentes $h_1, h_2, \dots, h_k$ para as quais dado um valor X devolvem um valor no domínio $(1, \dots, m)$ ou seja uma posição no vetor S .

Para agregar um elemento novo de X aplica-se a ele todas as funções k e o resultado dessas funções é tornado 1 em S . Para buscar se um novo elemento pertence ou não a X , a busca retorna “SIM” se todas as funções consultadas possuem um 1 em S .

Da maneira como o filtro é construído, é impossível eliminar um elemento. A solução pode ser um segundo filtro de Bloom contendo os eliminados. Veja no exemplo a seguir o funcionamento de

um filtro de Bloom (tirado da wikipedia)



Aqui está o conjunto $\{x, y, z\}$. As flechas mostram as posições no vetor de bits dos resultados após a aplicação das funções de hash a cada um dos elementos. O candidato w não está no conjunto $\{x, y, z\}$ porque ele tem um ou mais valores de hash valendo 0.

Nesta folha, vamos estudar a relação de pertinência de candidatos em um universo (denominado B) de elementos. Cada um terá 150 bytes aleatórios e esperar-se 2.000 ocorrências em B . O algoritmo de força bruta para estudar pertinência exigiria, no pior caso, examinar $150 \times 2000 = 300.000$ bytes. Vai-se fazer o mesmo gastando muito menos recursos. Imagine estes números elevados a expoentes grandes...

Vão-se definir 5 funções hash (denominadas h_1, h_2, h_3, h_4 e h_5) e todas elas vão gerar um resultado numérico inteiro entre os intervalos 0..19.999. Consequentemente o nosso filtro terá um comprimento de 20.000 bits. Claro que este número é inteiramente arbitrário e estimado pelo arquiteto da estrutura.

Vai-se definir um filtro (de nome F) que será um vetor binário de 20.000 bits. Após aplicar as 5 funções hash h_i a cada elemento de B vai-se marcar com um 1 a pertinência de B_j no filtro F .

Finalmente, para estudar o funcionamento do algoritmo, vão-se gerar 10 candidatos C com 50% de probabilidade de pertencerem ao conjunto original B . Depois, far-se-á uma pesquisa de pertinência probabilística de C_k sobre B usando F .

Funções hash propostas

Dado um elemento B , cada função hash devolverá um inteiro (i) entre 0 e 19999 e este inteiro determinará a levantada do bit i em F .

1.

$$i = \sum_{j=0}^{len(X)} ord(B_j)$$

Durante o somatório, deve-se examinar o overflow para evitar dificuldades com o tamanho dos inteiros envolvidos. Como este limite de overflow é inteiramente arbitrário vai-se usar o limite 1.000.000.000 (1 bilhão). Então, se durante a soma o valor ultrapassar 1 bilhão ele será substituído por valor mod 1.000.000.000 e segue o baile. Termina a função o calculo do resto do valor final por 20.000 mais um.

2. A segunda função só vai examinar os bytes j do elemento, onde j é a sequência de números primos: 2, 3, 5, 7, 11, 13... Também vai ser feita a soma da função ord de cada byte assim escolhido. O mesmo cuidado de overflow a 1.000.000.000 deve ser tomado. A função termina obtendo-se o modulo de 20.000 do resultado final mais um.
3. O elemento vai ser quebrado em pedaços de 64 bits (havendo necessidade de stuffing bits ao final colocar até 7 bytes valendo zeros binários). Esses blocos de 64 bits vão ser quebrados em 2 inteiros de 32 bits cada. Deve-se calcular o mod 20.000 de cada um desses 2 valores e deve-se reter o maior valor alcançado. Este valor retido é o resultado final da função hash.
4. Criar uma máscara binária de 64 bits e com o valor $X'02CAFEBABACA2288'$. Depois o elemento deve ter blocos de 64 bits O primeiro bloco deve ser operado usando XOR com a máscara acima, gerando uma nova máscara. Essa deve ser usada na segundo bloco gerando a terceira máscara e assim por diante. Havendo necessidade de stuffing bytes ao final usar até 7 bytes valendo $X'FF'$. O último resultado é convertido em 2 inteiros de 32 bits. Estes valores são subtraídos, o maior menos o menor (ou o primeiro menos o segundo em módulo, tanto faz). O valor

obtido é operado com mod 20000 mais um para obter o valor da função hash.

5. Os bytes do elemento das posições 1,2,4,8,16,32,... são convertidos a inteiro e somados, devidamente ponderados por 1,2,4,8,16,... Verificar overflow para 1.000.000.000 como acima descrito. O resultado final da função hash é o mod 20000 mais um desse resultado.

Exemplo Consulte no lugar de sempre o arquivo F669EXE1.myd e verifique que para este arquivo, as respostas esperadas são:

0 0 1 1 1 1 1 1 1 0

Para você fazer

Você receberá o arquivo que foi colocado no repositório de nome

F669D001.myd

Este arquivo contém 2010 chaves, compostas cada uma de caracteres ASCII. Deve construir um filtro de BLOOM para as primeiras 2000 ocorrências desse arquivo. Depois disto, deve consultar as últimas 10 ocorrências para ver se elas pertencem ou não ao conjunto inicial. É óbvio que a pertinência deve ser verificada via o filtro de Bloom e não por força bruta, pois este é o objetivo de aprendizagem deste exercício.

Informe 1 (pertence) ou 0 (não pertence) para cada uma das 10 últimas ocorrências do arquivo.

1	2	3	4	5	6	7	8	9	10



Multiplicação de matrizes

Este é um algoritmo famoso na matemática. Dadas as matrizes A (de i linhas por j colunas) e B (de k linhas por m colunas), onde $j = k$, obter a matriz C, multiplicação de A por B, de i linhas por m colunas, a partir de $C[i][m] = \sum_{x=1}^j A[i][x] \times B[x][m]$. Para implementá-lo, tem-se o

```
1: função MM (A[i][j], B[j][m]:real) : (C[i][m]:real)
2: inteiro AUX
3: inteiro x1, x2, x3
4: para x1 = 1 até i
5:   para x2 = 1 até m
6:     AUX ← 0
7:     para x3 = 1 até j
8:       AUX ← AUX + A[x1][x3] × B[x3][x2]
9:     fim{para}
10:    C[x1][x2] ← AUX
11:  fim{para}
12: fim{para}
13: devolva C
14: fim{função}
```

Como se pode ver no algoritmo, o número de colunas de A precisa ser igual ao número de linhas de B, se esta igualdade não ocorrer a multiplicação não pode ser realizada. O tamanho da matriz resultado da multiplicação é o número de linhas de A e o número de colunas de B. Acompanhe no quadro

matriz A	matriz B	matriz A × B
i, j	j, k	i, k

Antes de continuar com este exercício, implemente a multiplicação entre estas duas matrizes: A, formada por (5 linhas por 6 colunas)

13	5	6	46	48	26
74	27	87	55	52	34
18	44	11	38	12	36
23	92	66	60	61	78
40	71	90	72	7	82

e a matriz B, formada por (6 linhas por 8 colunas)

88	94	43	75	90	89	28	68
32	56	8	25	64	12	35	83
41	48	29	23	78	50	11	44
27	61	16	3	2	69	79	6
99	55	30	17	81	39	15	71
62	47	82	22	98	38	33	52

A multiplicação dessas duas matrizes deve ser (5 linhas por 8 colunas)

9156	8458	5081	2764	8486	7551	5817	6599
19684	20457	11149	10023	22828	18375	10221	16891
7889	9354	5365	3813	9870	7138	6535	8312
20169	21163	12825	8476	25811	15934	12819	20851
17203	20687	12984	8984	23911	17269	13094	17766

Se definirmos uma operação elementar neste procedimento como sendo a multiplicação de um elemento de A por outro de B e depois a soma desse produto em uma variável de trabalho, pode-se afirmar que a quantidade de operações é $i \times j \times k$ ou mais resumidamente como sendo $O(ijk)$. No exemplo acima, foram $5 \times 6 \times 8$ operações ou 240 operações.

Múltiplas multiplicações matriciais

Agora suponha a existência de uma série de multiplicações, como por exemplo

$M_{1[10,100]} \times M_{2[100,1000]} \times M_{3[1000,3]}$

Como a multiplicação matricial é associativa, tanto faz, realizar a multiplicação como

$(M_{1[10,100]} \times M_{2[100,1000]}) \times M_{3[1000,3]}$

ou como

$M_{1[10,100]} \times (M_{2[100,1000]} \times M_{3[1000,3]})$

entretanto, a quantidade de trabalho, nem de longe é a mesma. Em $(M_{1[10,100]} \times M_{2[100,1000]}) \times M_{3[1000,3]}$ são necessárias $10 \times 100 \times 1000 = 1.000.000$ de operações dentro do parênteses e depois mais 300.000 operações fora do parênteses, totalizando 1.300.000 operações.

Entretanto, na segunda parentização, as operações são 300.000 dentro do parênteses e mais 3.000 operações totalizando 303.000 operações, que é muito menos do que 1.300.000.

Veja um segundo exemplo em

$M = M_{1[10,20]} \times M_{2[20,50]} \times M_{3[50,1]} \times M_{4[1,100]}$

onde as dimensões de cada matriz estão entre colchetes.

Calculando a expressão como $M_1 \times (M_2 \times (M_3 \times M_4))$ são necessárias 125.000 operações, enquanto calculando $(M_1 \times (M_2 \times (M_3))) \times M_4$ são requeridas apenas 2.200 operações.

A avaliação de todas as possibilidades de operações entre as matrizes é um processo exponencial, como se pode ver em

$$X_{n+1} = \frac{1}{n+1} \binom{2n}{n}$$

onde $X(n)$ é o número de maneiras que se pode parentizar um conjunto de n símbolos. Este número cresce e se torna impraticável quando n é grande, veja na tabela a seguir:

1 matriz	não há o que calcular
2 matrizes	1 possibilidade
3 matrizes	2 possibilidades
4 matrizes	5
5	14
6	42
7	132
8	429
9	1.430
10	4.862
15	2.674.440
20	1.767.263.190
30	1.002.242.216.651.368
40	680.425.371.729.975.700.000

Usando a programação dinâmica pode-se estudar o comportamento do problema através de um algoritmo $O(n^3)$. Seja m_{ij} o custo mínimo de computar $M_i \times M_{i+1} \times \dots \times M_j$ para $1 \leq i \leq j \leq n$. Claramente $m_{ij} = 0$ se $i = j$ e $m_{ij} = 0$ se menor dos valores de $m_{ik} + m_{k+1,j} + r_{i-1} \times r_k \times r_j$ se $j > i$ variando k nos limites $i \leq k < j$. Aqui r_k é o k -ésimo elemento no vetor de dimensões das matrizes.

O termo m_{ik} é o custo mínimo de avaliar $M' = M_i \times M_{i+1} \times \dots \times M_k$.

O segundo termo $m_{k+1,j}$ é o custo mínimo de avaliar $M'' = M_{k+1} \times M_{k+2} \times \dots \times M_j$.

O terceiro termo é o custo de multiplicar M' por M'' . Note que M' é uma matriz $r_{i-1} \times r_k$ e M'' é uma matriz $r_k \times r_j$.

A equação acima determina que m_{ij} , com $j > i$ é o valor mínimo obtido de todos os valores possíveis de k entre i e $j - 1$ da soma desses 3 termos.

A estratégia da programação dinâmica calcula os m_{ij} s na ordem em que a diferença dos subscripts vai sendo incrementada. Começa-se calculando m_{ii} para todo i . Depois $m_{i,i+1}$ para todo i , depois $m_{i,i+2}$ e assim por diante. Desta forma, os termos m_{ik} e $m_{k+1,j}$ da fórmula acima já estarão disponíveis quando m_{ij} for ser calculado. Veja-se o algoritmo

```
1: função avaliamats v
2: n ← tamanho(v)-1
3: m ← matriz n linhas × n colunas ← 0
4: q ← matriz n linhas × n colunas ← 0
5: para h ← 1 até n - 1
6:   para i ← 1 até n - h
7:     j ← i + h
8:     m[i, j] ← ∞
9:     para k ← i até j - 1
10:      temp ← m[i, k] + m[k + 1, j] + (v[i] × v[1 + k] × v[1 + j])
11:      se temp < m[i, j]
12:        m[i, j] ← temp
13:      q[i, j] ← k
14:    fim{se}
15:  fim{para}
16: fim{para}
17: fim{para}
18: imprima m,q
```

A matriz m contém as quantidades de operações mínimas enquanto a matriz q contém qual operação deve ser feita a cada etapa. A análise de q pode ser feita pelo algoritmo

```
1: função explicaq (q,i,j)
2: se i=j
3:   imprima 'M',i
4: senão
5:   imprima '('
6:   explicaq (q,i,q[i,j])
7:   explicaq (q,(q[i,j]+1),j)
8:   imprima ')'
9: fim{se}
```

Para obter a parentização correta, deve-se chamar **explicaq(q,1,6)** (o último número é a quantidade de matrizes).

Eis os resultados do algoritmo para o exemplo acima: (a ← 10 20 50 1 100):

0	10000	1200	2200	0	1	1	3
0	0	1000	3000	0	0	2	3
0	0	0	5000	0	0	0	3

((M1(M2M3))(M4)
2200 1200 1000

Veja mais um exemplo: (c ← 13 25 4 80 1)

0	1300	5460	745	0	1	2	1
0	0	8000	420	0	0	2	2
0	0	0	320	0	0	0	3

(M1(M2(M3M4)))
745 420 320

Veja-se agora um exemplo maior:
f← 41 90 38 73 3 36 39

0	140220	253954	29652	34080	38661	0	1	2	1	4	4
0	0	249660	18582	28302	33324	0	0	2	2	4	4
0	0	0	8322	12426	16980	0	0	0	3	4	4
0	0	0	0	7884	12753	0	0	0	0	4	4
0	0	0	0	0	4212	0	0	0	0	0	5

((M1(M2(M3M4)))(M5M6))
38661 33324 16980 12426 4212

Para você fazer

1. Suponha a multiplicação matricial da seguinte sequência de matrizes:

$M[65 \times 74] \times M[74 \times 96] \times M[96 \times 49] \times M[49 \times 24] \times M[24 \times 37] \times M[37 \times 32] \times M[32 \times 66] \times M[66 \times 27]$

O objetivo é calcular a sequência de multiplicações que MINIMIZA a quantidade de operações elementares. Assim, implemente os algoritmos acima e responda:

- 1a. Qual o número de multiplicações elementares do primeiro par de matrizes a ser calculado ?
- 1b. Qual o número total de operações elementares ao calcular a expressão completa ?

2. Mais um caso,

$M[1 \times 23] \times M[23 \times 26] \times M[26 \times 76] \times M[76 \times 56] \times M[56 \times 11] \times M[11 \times 34] \times M[34 \times 84] \times M[84 \times 6]$

- 2a. Operações do primeiro par ?
- 2b. Número total de operações ?

1a	1b	2a	2b

Para saber mais:
1. AHO, HOPCROFT, ULLMAN. Alfred, John e Jeffrey **The design and Analysis of Computer Algorithms**. 1974. Addison Wesley, Reading Massachusetts. (UP: 005.12 A286d, pág. 65).
2. ZIVIANI, Nivio. **Projeto de Algoritmos**. 2012. Thomson. (UP: 005.12 Z82p, 3.ed., pág. 51).
3. CORMEN, Thomas, et alli. **Algoritmos**. 2002. Campus. (UP: 005.12 A396alg, pág. 266 a 271).



Algoritmos memoizáveis

Lembrando a sequência de Fibonacci, cuja definição é

$$\begin{cases} F(1) = 1 \\ F(2) = 1 \\ F(n) = F(n-1) + F(n-2) \quad \text{se } n \geq 3 \end{cases}$$

Uma possibilidade de implementar este programa é imediatamente reproduzi-lo em um linguagem de programação dando como resultado

```
1: função int FIBO(int n)
2: se n < 3
3:   retorne 1
4: senão
5:   retorne FIBO(n-1) + FIBO(n-2)
6: fim{se}
```

Esta implementação apresenta um problema sério: o recálculo do mesmo valor da função muitas vezes. Acompanhe o raciocínio. Para calcular FIBO(3) fazem-se 2 chamadas a FIBO (FIBO(1) e FIBO(2)). Para calcular FIBO(4), fazem-se 2 chamadas a FIBO(3) e a FIBO(2). Como FIBO(3) faz 2 chamadas, no total são 4. Para calcular FIBO(5), chama-se FIBO(4) e FIBO(3), o que dá 6 chamadas... Acompanhe na tabela

FIBO(1)	imediata
FIBO(2)	imediata
FIBO(3)	2 chamadas
FIBO(4)	4 chamadas
FIBO(5)	6 chamadas
FIBO(6)	10 chamadas
FIBO(7)	16 chamadas
FIBO(8)	26 chamadas
FIBO(9)	42 chamadas
FIBO(10)	68 chamadas
FIBO(20)	8362
FIBO(30)	1.028.458
FIBO(40)	126.491.972
FIBO(50)	15.557.484.098
FIBO(60)	1.913.444.052.082
FIBO(70)	235.338.060.921.988
FIBO(80)	28.944.668.049.352.442
FIBO(90)	3.559.958.832.009.428.378
FIBO(100)	437.845.991.669.110.338.052

Uma estratégia alternativa para este tipo de problema é o chamado algoritmo memoizável, a cada chamada recursiva verifica se essa instância já não foi resolvida anteriormente. Para que isto funcione adequadamente deve-se criar uma tabela, na qual vão ser guardados os valores calculados. Veja o algoritmo:

```
1: global MFIB ← 1,1,-1,-1,... 998 vezes... -1
2: função int FIBOMEMO(int n)
3: se mfb[n] = -1
4:   mfb[n] ← FIBOMEMO(n-1) + FIBOMEMO(n-2)
5: fim{se}
6: retorne mfb[n]
```

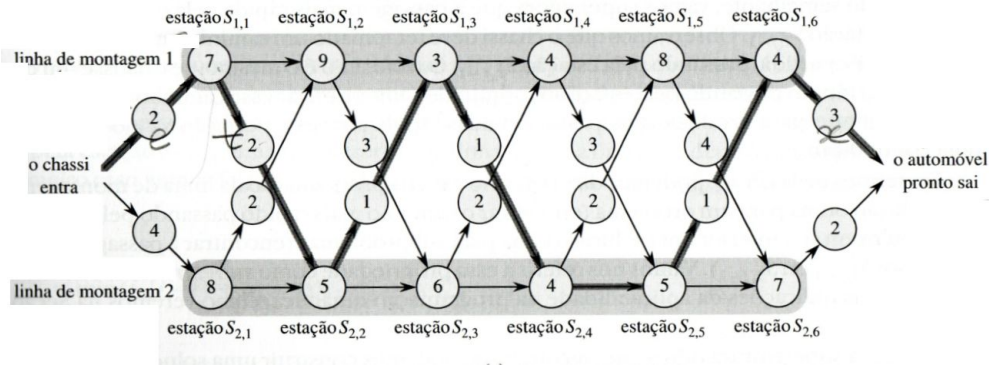
Acompanhe no quadro a seguir os desempenhos de ambos os programas em um computador real (Notebook DELL, equipado com Windows 7 e APL2). Obviamente ambos os programas dão o mesmo resultado numérico.

n	fibmemoiz(n)	fibrecu(n)
10	32 miliseg	78 miliseg
20	62 miliseg	127 miliseg
30	78 miliseg	6 seg
31	80 miliseg	10 seg
32	80 miliseg	16 seg
33	80 miliseg	25 seg
34	80 miliseg	41 seg
40	80 miliseg	12 min 16 seg
45	90 miliseg	2 h 16 min
50	97 miliseg	1 d 1 h
51	103 miliseg	1 d 16 h
52	103 miliseg	2 d 17 h

Programação Dinâmica

Suponha uma indústria montadora de automóveis. Originalmente, a fábrica tinha uma única linha de produção, e há pouco tempo, numa ampliação,

criou-se uma segunda linha de produção. O aspecto geral das duas linhas pode ser visto na figura



Um chassi entra em cada linha de montagem tem as peças adicionadas a ele em uma série de estações e um automóvel pronto sai no final da linha. Cada linha de montagem tem n estações, numeradas com j = 1, 2, ..., n. Denota-se a j-ésima estação da linha i (onde i = 1 ou 2) por Si,j. A S1,j executa a mesma operação que a estação S2,j, mas como as estações foram criadas em tempos (e com tecnologias) diferentes, explica-se as diferenças de tempos. Denota-se ai,j o tempo exigido para montagem da operação na estação Sj na linha i. Como se pode ver na figura, há também um tempo ei gasto para o chassi entrar na linha e um tempo de saída xi para o automóvel concluído sair da linha de montagem i.

Normalmente, os carros que entram na linha 1 seguem por ela, assim como os que entram na 2 seguem na linha 2, demorando em cada linha o valor especificado para cada tarefa. Para passar da tarefa j para a tarefa j + 1 NA MESMA LINHA não há nenhuma demora adicional. Entretanto, existe a possibilidade de um chassi mudar de linha. Se isto acontecer, há uma demora para a troca de linha estipulada em ti,j. Este tempo deve ser acrescentado se o automóvel for transferido para a outra linha após a tarefa Si,j. Finalmente, chegamos à pergunta objetivo deste exercício. Se vier uma encomenda de um grande cliente, que quer um carro com a máxima urgência, qual a sequência de operações em ambas as linhas que gera o carro mais rápido na fábrica ?

O algoritmo

```
1: função CARRORAP (a, t, e, x, n)
2: f1[1] ← e[1] + a1,1
3: f2[1] ← e[2] + a2,1
4: para j ← 2 até n
5:   se ((f1[j-1]+a[1][j]) ≤ (f2[j-1]+t[2][j]-1)+a[1][j]))
6:     f1[j] ← f1[j-1] + a[1][j]
7:     l1[j] ← 1
8:   senão
9:     f1[j] ← f2[j-1] + t[2][j-1] + a[1][j]
10:    l1[j] ← 2
11:   fim{se}
12:   se ((f2[j-1] + a[2][j]) ≤ (f1[j-1] + t[1][j-1] + a[2][j]))
13:     f2[j] ← f2[j-1] + a[2][j]
14:     l2[j] ← 2
15:   senão
16:     f2[j] ← f1[j-1] + t[1][j-1] + a[2][j]
17:     l2[j] ← 1
18:   fim{se}
19: fim{para}
20: se ((f1[n] + x[1]) ≤ (f2[n] + x[2]))
21:   fstar ← f1[n] + x[1]
22:   lstar ← 1
23: senão
24:   fstar ← f2[n] + x[2]
25:   lstar ← 2
26: fim{se}
27: print ('Duracao total ',fstar)
28: i ← lstar
29: j ← n
30: print('linha ',(i),' estação ',n)
31: para (j ← n até 2 passo -1)
32:   i ← l[i][j]
33:   print('linha ',(i),' estação ',j - 1)
34: fim{para}
```

Exemplo 1

Seja a instância:
número de estações: 6
tempos na linha 1: 7 9 3 4 8 4
linha 2: 8 5 6 4 5 7
swap da linha 1 para linha 2: 2 3 1 3 4
swap da linha 2 para linha 1: 2 1 2 2 1
set up: 2 4
set down: 3 2

Para este conjunto de dados, a resposta do algoritmo é 38 unidades de tempo, correspondendo à seguinte sequência de estações nas linhas: 1 2 1 2 2 1.

Este exemplo está explicado e resolvido no livro do Cormen (algoritmos) páginas 261 a 265.

Exemplo 2

Numero de estações em cada linha: 10
Tempos linha 1: 88 47 14 85 82 68 18 10 46 9
Tempos linha 2: 44 77 53 70 29 57 3 32 66 79
Swap 1 para 2: 5 11 28 19 4 1 24 6 9
Swap 2 para 1: 16 26 17 27 18 15 23 29 12
Set up: 11 20
Set down: 4 13

420 unidades de tempo, com: 2 1 1 2 2 2 2 1 1 1

Exemplo 3

Numero de estações em cada linha: 10
Tempos linha 1: 30 17 1 50 79 86 14 27 10 94
Tempos linha 2: 75 1 50 25 16 70 67 91 44 97
Swap 1 para 2: 11 14 5 15 10 13 30 2 22
Swap 2 para 1: 17 1 24 25 23 20 18 7 26
Set up: 2 1
Set down: 12 6

381 u.t. com: 1 1 1 2 2 2 1 1 1 2

Para você fazer

Resolva a seguinte instância:

Numero de estações em cada linha: 10
Tempos linha 1: 56 99 33 53 7 35 68 72 78 66
Tempos linha 2: 75 1 50 25 16 70 67 91 44 97
Swap 1 para 2: 25 7 21 17 3 8 13 30 5
Swap 2 para 1: 19 16 4 12 9 22 6 20 1
Set up: 14 12
Set down: 11 15

u.t.	1	2	3	4	5	6	7	8	9	10



- 1 - /

Padrão PDF

O padrão *Portable Document Format (PDF)* é o padrão de fato em descrição de páginas impressas, e corre a passos largos para se tornar a maneira pela qual vastos documentos são e serão produzidos de agora para o futuro. É comum hoje, ao comprar algum artefato tecnológico que exija documentação para sua operação, que o mesmo venha acompanhado de um CD ao invés de uma caixa de manuais. É óbvio que o CD conterá os mesmos (ou mais) manuais escritos em formato PDF.

Um arquivo PDF descreve como uma página ficará depois de impressa e ele é melhor do que imagem dessa página pelas seguintes razões

- * A imagem como tal sempre é muito maior do que sua descrição. A razão para isso é a evidente redundância de qualquer imagem.
- * A imagem como tal é fixa e só pode ser rotacionada, redimensionada ou de alguma maneira modificada à custa de processamento e perda de qualidade.
- * Sobre uma imagem não há como manipular o conteúdo, por exemplo conduzindo uma busca textual (~F alguma coisa ou então ~C e ~V).
- * Alterar uma única palavra sobre uma imagem exige habilidades de pintura. Sobre a descrição de uma página é tarefa simples: basta trocar uma palavra pela outra.
- * Imagine uma imagem enviada para um monitor (72 DPP) ou a mesma imagem enviada para uma impressora (300 DPP). Não podem ser a mesma imagem.

A versão 1.0 do padrão PDF nasceu em 1993 quando a Adobe lançou 2 programas: o Distiller (para gerar PDF) e o Reader (para ler): ambos pagos. A coisa começou a mudar quando o departamento de rendas americano comprou uma licença que permitia aos contribuintes baixar o Reader. Rapidamente a Adobe deu-se conta de que seria melhor para ela distribuir gratuitamente o Reader. Nos dez anos seguintes o PDF acabou se tornando o padrão de fato de descrição de imagens.

Apresenta as seguintes vantagens:

- * acesso aleatório: qualquer página pode ser montada independente das demais.
- * linearização: todos os elementos necessários para montar uma página estão juntos.
- * atualização incremental: mudanças ficam registradas no fim do arquivo. Vantagem: opção de desfazer ilimitada e rápida atualização. Desvantagem: o arquivo não pára de crescer.
- * fontes incorporadas: o destino sempre será montado corretamente. A fonte é desbastada de tudo que não é necessário.
- * padronização ISO: em 2008 a ISO abraçou o padrão Adobe PDF 1.7, o que o transformou em padrão aberto.
- * compatibilidade para trás e para a frente: Qualquer leitor lê as versões antigas e deve ler as futuras, já que todos os leitores ignoram o que não conhecem.
- * integrado a todos os principais browsers do protocolo HTTP.

Um arquivo PDF

Cabeçalho Formado por 2 linhas: a primeira identifica o arquivo como um PDF e informa a versão PDF em que ele foi gerado. A segunda linha inclui caracteres que não podem ser impressos (para avisar eventuais leitores deste fato). Exemplo:

```
%PDF-1.0
%ã
```

Corpo Um conjunto de nodos de um grafo, contendo as páginas, conteúdo gráfico, textos, sempre na forma de objetos. Cada objeto começa com um número de objeto (iniciando em 1), um número de geração (sempre 0) e a palavra chave obj. O objeto termina pela palavra chave endobj. Exemplo:

```
1 0 obj    -- descrevendo o objeto 1
<<        -- começa um dicionário
  /Kids [2 0 R] -- /Kids tem o valor 2 0 R
  /Count 1   -- só um
  /Type /Pages -- do tipo : página
>>        -- fim do dicionário
endobj     -- fim do objeto
```

Tabela de Referência Cruzada Lista o deslocamento em bytes de cada objeto no corpo do arquivo. Exemplo

```
0 6        -- seis entradas na tabela
0000000000 65535 f -- entrada especial
0000000015 00000 n -- obj1 no desloc 15
0000000074 00000 n -- obj2 no desloc 74
...
```

Rodapé Informações e metadados para aumentar a performance de acesso. A primeira entrada é trailer. Depois o dicionário de trailer que deve apresentar ao menos a entrada /Size dizendo quantos itens há na tabela de referência cruzada e /Root dizendo qual objeto é o catálogo do documento. Depois vem uma linha que só contém startxref seguida por uma única linha indicando o deslocamento em bytes utilizado no início da tabela de referência cruzada do arquivo. Termina tudo a linha %%EOF. Exemplo:

```
trailer -- palavra chave
<< -- começa um dicionário
  /Root 5 0 R -- o catálogo é o objeto 5
  /Size 6 -- tem o tamanho 6
>> -- fim do dicionário
startxref
459 -- deslocamento da tabela de ref cruzada
%%EOF -- fim de arquivo
```

Componentes

Uma descrição PDF suporta 5 componentes básicos: i. Números inteiros e reais (mas não exponenciais); ii. strings que são escritos entre parênteses; iii. Nomes, que sempre começam por uma barra normal; iv. Os valores booleanos true e false e v. O componente nulo (null). Aceita também 3 componentes compostos: i. Arrays, que são uma coleção ordenada de outros componentes, escritos entre colchetes; ii. Dicionários que são uma lista não ordenada de duplas: nome e componente. Começam por << e terminam por >>. Finalmente iii. Fluxos, que armazenam dados binários, sempre acompanhados de um dicionário que descreve seus atributos, como comprimento e parâmetros de compactação, por exemplo.

Criando um PDF na unha

A seguir deve-se criar o arquivo seguinte usando um editor ASCII plano (notepad, scintilla,...). Depois de salvo deve-se rodar o pdftk com a finalidade de acertar alguns detalhes mais difíceis de fazer à mão. As mudanças serão poucas. Aliás é bem interessante você comparar ambos os arquivos e olhar o que o pdftk alterou.

Digite o arquivo 01.PDF

```
%PDF-1.0
1 0 obj
<< /Type /Pages
  /Count 1
  /Kids [2 0 R]
>>
endobj
```

```
2 0 obj
<< /Type /Page
  /MediaBox [0 0 612 792]
  /Resources 3 0 R
  /Parent 1 0 R
  /Contents [4 0 R]
>>
endobj
```

```
3 0 obj
```

```
<< /Font
<< /F0
<< /Type /Font
  /BaseFont /Times-Italic
  /Subtype /Type1
>>
>>
endobj

4 0 obj
<< >>
stream
1. 0. 0. 1. 50. 700. cm
BT
/F0 72. Tf
(01a Mundo!) Tj
ET
endstream
endobj

5 0 obj
<< /Type /Catalog
  /Pages 1 0 R
>>
endobj
```

```
6 0 obj
<<
  /Author (...)
  /Keywords (teste de um pdf feito a mao)
>>
endobj
```

```
xref
0 7
trailer
<<
  /Size 7
  /Root 5 0 R
  /Info 6 0 R
>>
startxref
0
%%EOF
```

Atualize o arquivo PDF Chame o utilitário pdftk passando a ele o comando pdftk oi.pdf output ja.pdf. Veja o que ele alterou: i. Incluiu caracteres não imprimíveis após o header; ii. Acertou o tamanho (/Length) do objeto 4; iii. Criou a xref (entre as palavras xref e trailer); iv. Criou um tamanho correto (depois da palavra startxref). v. Renumerou os objetos.

Além desta funcionalidade (consertar um arquivo pdf mambembe, o pdftk possui as seguintes funcionalidades: mesclar vários pdf em um só; comprimir e descomprimir arquivos; quebrar um arquivo em vários; incluir marcas d'água, criptografar e descriptografar entre outras funcionalidades.

Referência para esta aula:
www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf

Entregue o arquivo Renomeie o arquivo ja.pdf para fulano.pdf (fulano é o seu nome) e entregue-o para o professor.

Para você fazer

Construa um arquivo PDF com 2 páginas: Na primeira deve escrever a frase

Deixai-me viver!

com um tamanho de 36 pts e na segunda folha a frase

Na noite sombria,

com tamanho 48 pts. Finalmente, deve criar um metadado (/Author) com o seu sobrenome. Ambas as frases acima são do canto IV de I-Juca-Pirama de Gonçalves Dias.

A avaliação desta folha será conduzida abrindo o arquivo produzido por você nas duas modalidades: com o Acrobat reader (para ver as frases) e com um editor ASCII para ver o metadado /Author.



Compressão JPG

Ao contrário da compressão de texto ou de programas, quando não é aceitável a perda de qualquer informação, no caso de imagens (e sobretudo no caso de vídeo) é perfeitamente aceitável algum nível de degradação da imagem. Isso ocorre devido a dois fatores:

- ★ Nossos olhos (e ouvidos) não são capazes de distinguir mudanças sutis e principalmente
- ★ Imagens (e vídeos) são objetos muito grandes que praticamente exigem serem comprimidos para um manuseio adequado.

As etapas pelas quais uma imagem passa antes de ser guardada em memória como um objeto JPG são as seguintes:

1. A imagem é dividida em blocos de 8 × 8 pixels. A razão é diminuir a quantidade de operações aritméticas necessárias durante a compressão (ainda que pagando para isso ter uma compressão ligeiramente ineficiente).
2. Cada bloco de imagem sofre a ação da transformada discreta do cosseno. Este procedimento (reversível) gera outra matriz 8 × 8, mas com uma propriedade interessante. Apenas os valores mais à esquerda e acima é que têm valores significativos. Embaixo e à direita, o valor mais comum é zero.
3. Os dados são quantizados visando diminuir o comprimento dos valores e aumentar a redundância. Aqui ocorre a degradação na compressão, já que este processo é irreversível.
4. Na compressão, os dados quantizados são percorridos em zig-zag. A seguir processos de compressão, tais como run-length, Huffman, QM (esta última protegida por patente) são aplicados gerando o arquivo final.

É de se notar que a maioria das câmeras digitais de fotografia (ou como dizem os franceses: *photo numérique*) já fazem todo este processamento, a fim de economizar memória.

Ao longo do processo, inúmeras otimizações são feitas para torná-lo mais eficiente. A primeira já citada é trabalhar com blocos de 64 valores ao invés de trabalhar com a imagem inteira. Uma segunda otimização ocorre quando os cossenos (uma função transcendental) são previamente calculados e armazenados, sendo depois apenas consultados. Uma terceira otimização ocorre quando o processo é manuseado algebricamente a fim de que as iterações sejam convertidas em um produto matricial de matrizes. Finalmente, usa-se aritmética de ponto fixo (fazendo os ajustes necessários) já que esta sempre é muito mais eficiente que a aritmética de ponto flutuante.

Transformada discreta do cosseno

Baseada no conceito de localidade de referência espacial (probabilidade alta de dois pixels vizinhos terem valores próximos ou iguais) esta transformação atua muito eficientemente na preparação de dados de fotografias para compressão. A fórmula da transformada bidimensional é

$$G_{ij} = \frac{1}{\sqrt{2n}} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} P_{xy} \cos\left(\frac{(2y+1)j\pi}{2n}\right) \cos\left(\frac{(2x+1)i\pi}{2n}\right)$$

Onde *n* é igual a 8, *C_f* = 1 quando *f* > 0 e *C_f* = $\frac{1}{\sqrt{2}}$ quando *f* = 0. *G_{ij}* é o valor na matriz transformada, *P_{xy}* é o valor do pixel correspondente na imagem, e *i*, *j*, *x* e *y* variam entre 0 e 7.

A transformada inversa discreta do cosseno bidirecional (conhecida como IDCT) apresenta a seguinte formulação:

$$P_{xy} = \frac{1}{4} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C_i C_j G_{ij} \cos\left(\frac{(2x+1)i\pi}{2n}\right) \cos\left(\frac{(2y+1)j\pi}{2n}\right)$$

Onde *n* é igual a 8, *C_f* = 1 quando *f* > 0 e *C_f* = $\frac{1}{\sqrt{2}}$ quando *f* = 0. *P_{xy}* é o valor do pixel correspondente na imagem, *G_{ij}* é o valor na matriz transformada, e *i*, *j*, *x* e *y* variam entre 0 e 7.

A fórmula da transformada direta pode ser facilmente traduzida a um programa em qualquer linguagem (neste caso em C)

```
#include<stdio.h>
#include<math.h>
double c(double x){
    if (x==0) {
        return(1/sqrt(2.0));
    }
    else {
        return(1.0);
    }
}
void dct (double ima[8][8], double r[8][8]) {
```

```
double PI = 3.14159265;
int i,j,x,y;
double s,t;
for (i=0;i<8;i++){
    for(j=0;j<8;j++){
        s=0.0;
        for(x=0;x<8;x++){
            for(y=0;y<8;y++){
                t=ima[x][y];
                s=s+t*(cos((1+2.0*y)*j*PI/16.0))*
                    (cos((1+2.0*x)*i*PI/16.0));
            }
        }
        t=0.25*(c(i))*(c(j))*s;
        r[i][j]=t;
    }
}
}
main() {
    int i,j;
    double im1[8][8]={
        1,19,...,
        ...,
    };
    double in1[8][8];
    dct(im1, in1);
    for (i=0;i<8;i++){
        for (j=0;j<8;j++){
            printf("%.2f",in1[i][j]);
        }
        printf("\n");
    }
}
```

Exemplo

Suponha um bloco de imagem com os seguintes valores

1	19	37	55	73	91	109	127
19	37	55	73	91	109	127	145
37	55	73	91	109	127	145	163
55	73	91	109	127	145	163	181
73	91	109	127	145	163	181	199
91	109	127	145	163	181	199	217
109	127	145	163	181	199	217	235
127	145	163	181	199	217	235	253

A transformada discreta do cosseno para este bloco é

1016.0	-328.0	0.0	-34.3	0.0	-10.2	0.0	-2.6
-328.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-34.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-10.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-2.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Para você fazer

Suponha um bloco de 8 × 8 com os seguintes valores de cinza.

46	46	58	34	46	22	46	46
34	34	34	22	46	58	46	22
34	46	34	34	46	34	58	34
22	34	58	34	58	34	58	46
58	22	22	46	34	22	58	22
22	34	58	58	58	46	58	46
58	34	58	34	58	22	58	58
58	34	58	22	22	22	58	22

Aplice a transformada discreta do cosseno a este bloco e informe:

1.Valor da transformada em [0,0]	Valor da transformada em [0,2]



- 1 - /

Compressão JPG

Ao contrário da compressão de texto ou de programas, quando não é aceitável a perda de qualquer informação, no caso de imagens (e sobretudo no caso de vídeo) é perfeitamente aceitável algum nível de degradação da imagem. Isso ocorre devido a dois fatores:

- ★ Nossos olhos (e ouvidos) não são capazes de distinguir mudanças sutis e principalmente
- ★ Imagens (e vídeos) são objetos muito grandes que praticamente exigem serem comprimidos para um manuseio adequado.

As etapas pelas quais uma imagem passa antes de ser guardada em memória como um objeto JPG são as seguintes:

1. A imagem é dividida em blocos de 8 × 8 pixels. A razão é diminuir a quantidade de operações aritméticas necessárias durante a compressão (ainda que pagando para isso ter uma compressão ligeiramente ineficiente).
2. Cada bloco de imagem sofre a ação da transformada discreta do cosseno. Este procedimento (reversível) gera outra matriz 8 × 8, mas com uma propriedade interessante. Apenas os valores mais à esquerda e acima é que têm valores significativos. Embaixo e à direita, o valor mais comum é zero.
3. Os dados são quantizados visando diminuir o comprimento dos valores e aumentar a redundância. Aqui ocorre a degradação na compressão, já que este processo é irreversível.
4. Na compressão, os dados quantizados são percorridos em zig-zag. A seguir processos de compressão, tais como run-length, Huffman, QM (esta última protegida por patente) são aplicados gerando o arquivo final.

É de se notar que a maioria das câmeras digitais de fotografia (ou como dizem os franceses: *photo numérique*) já fazem todo este processamento, a fim de economizar memória.

Ao longo do processo, inúmeras otimizações são feitas para torná-lo mais eficiente. A primeira já citada é trabalhar com blocos de 64 valores ao invés de trabalhar com a imagem inteira. Uma segunda otimização ocorre quando os cossenos (uma função transcendental) são previamente calculados e armazenados, sendo depois apenas consultados. Uma terceira otimização ocorre quando o processo é manuseado algebricamente a fim de que as iterações sejam convertidas em um produto matricial de matrizes. Finalmente, usa-se aritmética de ponto fixo (fazendo os ajustes necessários) já que esta sempre é muito mais eficiente que a aritmética de ponto flutuante.

Transformada discreta do cosseno

Baseada no conceito de localidade de referência espacial (probabilidade alta de dois pixels vizinhos terem valores próximos ou iguais) esta transformação atua muito eficientemente na preparação de dados de fotografias para compressão. A fórmula da transformada bidimensional é

$$G_{ij} = \frac{1}{\sqrt{2n}} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} P_{xy} \cos\left(\frac{(2y+1)j\pi}{2n}\right) \cos\left(\frac{(2x+1)i\pi}{2n}\right)$$

Onde *n* é igual a 8, *C_f* = 1 quando *f* > 0 e *C_f* = 1/√2 quando *f* = 0. *G_{ij}* é o valor na matriz transformada, *P_{xy}* é o valor do pixel correspondente na imagem, e *i*, *j*, *x* e *y* variam entre 0 e 7.

A transformada inversa discreta do cosseno bidirecional (conhecida como IDCT) apresenta a seguinte formulação:

$$P_{xy} = \frac{1}{4} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C_i C_j G_{ij} \cos\left(\frac{(2x+1)i\pi}{2n}\right) \cos\left(\frac{(2y+1)j\pi}{2n}\right)$$

Onde *n* é igual a 8, *C_f* = 1 quando *f* > 0 e *C_f* = 1/√2 quando *f* = 0. *P_{xy}* é o valor do pixel correspondente na imagem, *G_{ij}* é o valor na matriz transformada, e *i*, *j*, *x* e *y* variam entre 0 e 7.

A fórmula da transformada direta pode ser facilmente traduzida a um programa em qualquer linguagem (neste caso em Python)

```
import math
import numpy as np
def c(x):
    if x==0:
        return 1/math.sqrt(2.0)
    else:
        return 1.0
def dct(ima):
    r=np.zeros((8,8),float)
    i=0
    while (i<8):
```

```
        j=0
        while(j<8):
            s=0.0
            x=0
            while x<8:
                y=0
                while y<8:
                    t=ima[x][y]
                    s=s+t*(math.cos((1+2.0*y)*j*math.pi/16.0))*
                        (math.cos((1+2.0*x)*i*math.pi/16.0))
                    y=y+1
                x=x+1
                t=0.25*(c(i))*(c(j))*s
                r[i][j]=t
                j=j+1
            i=i+1
        return r
im=np.array([[1, 19, 37, 55, 73, 91, 109, 127],
[19, 37, 55, 73, 91, 109, 127, 145],
[37, 55, 73, 91, 109, 127, 145, 163],
[55, 73, 91, 109, 127, 145, 163, 181],
[73, 91, 109, 127, 145, 163, 181, 199],
[91, 109, 127, 145, 163, 181, 199, 217],
[109, 127, 145, 163, 181, 199, 217, 235],
[127, 145, 163, 181, 199, 217, 235, 253]],float)
for linha in dct(im):
    for val in linha:
        print ('%7.1f '% val,end='')
    print ()
```

Exemplo

Suponha um bloco de imagem com os seguintes valores

1	19	37	55	73	91	109	127
19	37	55	73	91	109	127	145
37	55	73	91	109	127	145	163
55	73	91	109	127	145	163	181
73	91	109	127	145	163	181	199
91	109	127	145	163	181	199	217
109	127	145	163	181	199	217	235
127	145	163	181	199	217	235	253

A transformada discreta do cosseno para este bloco é

1016.0	-328.0	0.0	-34.3	0.0	-10.2	0.0	-2.6
-328.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-34.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-10.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-2.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0

🔗 Para você fazer

Suponha um bloco de 8 × 8 com os seguintes valores de cinza.

29	48	86	67	67	29	67	29
67	29	29	86	86	29	48	67
48	29	29	29	29	48	29	29
86	86	48	48	67	29	48	86
48	86	67	29	48	29	48	29
29	86	29	67	48	29	48	48
86	48	67	48	48	86	29	67
29	86	86	67	86	48	29	67

Aplique a transformada discreta do cosseno a este bloco e informe:

1.Valor da transformada em [0,0]	Valor da transformada em [0,2]



Algoritmo Reed-Muller - detecção e Correção de erros

Na década de 70, os EUA mandaram a Marte 2 sondas, denominadas VIKING. Elas enviaram milhares de fotografias do planeta vizinho (inclusive uma que deu o que falar: parecia uma face humanóide). Após capturas e codificados, os pixels viajavam cerca de 15 minutos à velocidade da luz (300.000 Km/s) e eram recebidos, decodificados e gravados na Terra, no ritmo de 500 pixels/seg. A distância entre Marte e a Terra era de 268.200.000 Km (o dobro da Terra/Sol).

Cydonia Cydonia é uma região de Marte, no hemisfério do norte do planeta, em uma zona transitiva entre as regiões fortemente cobertas de crateras do sul e planícies relativamente lisas ao norte. Algum planetologista acreditam que as planícies do norte podem ter sido fundos do oceano e Cydonia pode ter sido uma zona litorânea (embora isto ainda seja incerto). Cydonia está coberta em numerosos planaltos escarpados alguns dos quais atraíram a atenção de cientistas e curiosos.

A Face A Face em Marte é uma grande característica da superfície do planeta Marte localizada na região de Cydonia, 10 graus a norte do equador marciano. Mede aproximadamente 3 km de comprimento e 1,5 km de largura. Foi fotografada a 25 de Julho de 1976 pela sonda Viking 1 que orbitava o planeta na altura.

A maioria das interpretações da fotografia sugeria que seria uma formação natural, uma das muitas "mesas" da região de Cydonia. Nesta visão, a aparência da face tem origem numa combinação do ângulo de luz (com o Sol baixo no horizonte marciano na altura em que a fotografia foi tirada), a baixa resolução da fotografia que suavizou as irregularidades da superfície e a tendência do cérebro humano em reconhecer padrões familiares, especialmente caras (pareidolia). Finalmente, um buraco nos dados enviados pela Viking 1 criaram um ponto negro no local exato onde se localizaria uma narina na face humana, muitos outros destes pontos negros são visíveis na imagem.

Outra interpretação da foto é que representaria um monumento artificial criado por antigos marcianos ou outros extraterrestres visitantes do sistema solar num passado muito antigo. O livro "Message of Cydonia" (Mensagem de Cydonia) de Richard Hoagland vai mais longe e interpreta o local como sendo uma cidade arruinada com pirâmides construídas artificialmente. Uma destas pirâmides perto da face é trilateral, lisa e com uma cratera perto da base que a maioria dos cientistas crêem que são de origem natural e produzidas por milhões de anos de erosão causada pelas tempestades de areia. O local seria uma cidade e um forte em ruínas, e que a Face estava alinhada apontando para o local em que o Sol se levantava há meio milhão de anos atrás, época em que se acreditava que a Face tinha sido construída.

A interpretação científica ganhou fôlego com as imagens da Mars Global Surveyor em 1998 e a Mars Odyssey em 2002, que mostraram a região com uma luminosidade diferente e com uma melhor resolução e o aspecto de face quase que desaparece, o que levou a que os que suportam teorias da conspiração afirmassem que as imagens foram propositadamente alteradas.

Em 21 de Setembro de 2006 a Agência Espacial Europeia publicou novas fotografias da região de Cydonia tiradas pela Mars Express. As novas imagens têm uma resolução de menos de 14 m/pixel.

O processo de fotografar Qualquer erro introduzido na viagem era fatal: não havia como repetir a visada. Tendo em vista esta restrição, usou-se um método forte de detecção e correção de erros, conhecido como Código Reed-Muller. Aqui, cada pixel era codificado em 32 níveis de cinza e, ao invés de serem enviados 5 bits, eram enviados 32. Isto permitia detectar e corrigir até 7 erros cometidos no envio dos 32 bits. Ou seja, o sistema se agüentava firme com até 21% (7/32) de taxa de erro, o que é altíssimo sob qualquer ponto de vista, mesmo sob um ponto de vista marciano.

O Código Reed-Muller usa as matrizes de Hadamard que por sua vez codificam os bits usando os códigos Gray, que implementam o conceito de distância de Hamming entre eles.

As matrizes de Hadamard são obtidas a partir da seguinte equação:

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \text{ e } H_{n+1} = H_n \otimes H_1$$

onde $\otimes$ = produto cartesiano. O produto cartesiano é simples: troque cada 1 em H_n por H_1 e cada -1 por $-H_1$. Na missão Mariner, construiu-se e usou-se a matriz H_5 (32,5). Esta matriz tinha 32 x 32 bits (a distância de Hamming entre linhas vizinhas sempre era 16). Por característica matemática, nesta codificação, pode-se detectar e corrigir até $\lfloor ((d-1)/2) \rfloor$ erros, onde d é igual a distância de Hamming entre o código recebido e o existente na tabela.

Quando a Mariner queria enviar um certo nível de cinza (digamos 28) ela enviava os 32 bits da linha (no caso a 28. linha) Na recuperação na Terra o string recebido era comparado com as 32 linhas possíveis. Se nenhum erro houvesse haveria um batimento completo com alguma linha: pronto, o número da linha seria o código esperado. Se 1 erro aparecesse, não haveria mais o batimento completo, mas daí o receptor deveria procurar qual a linha que tinha a menor distância de Hamming em relação ao string recebido.

Como 1 erro ocorreu, haveria uma linha com $d=1$. O número desta linha era o código correto. Isto funciona com $d = 0, 1, 2, 3, 4, 5, 6$ e até 7. Com 8 erros já apareceria uma ambigüidade (2 linhas seriam candidatas...)

Acompanhe o exemplo

Nossa espaçonave PÉNATABUA vai enviar pixels com 16 níveis de cinza, contendo imagens do planeta VAIQUEÉMOLÉ. Usará portanto 16 bits para cada pixel, usando a matriz (16,4) dos códigos Reed-Muller.

Vamos construir as matrizes de Hadamard para (16,4). Acompanhe na louza o processo desde H_1, H_2, H_3 e H_4 .

H_4 ficará como (0 ao invés de -1 para facilidade):

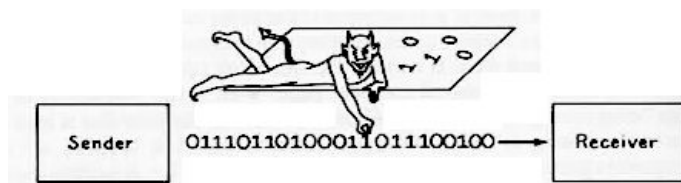
```
0= 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1= 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
2= 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
3= 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
4= 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
5= 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1
6= 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1
7= 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
8= 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
9= 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1
A= 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1
B= 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0
C= 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
D= 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0
E= 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0
F= 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1
```

Antes de continuar, procure responder. Até quantos erros o esquema acima pode corrigir ?

2. Suponhamos que a nossa espaçonave queira mandar os pixels: 0, 7, A deve enviar portanto:

```
0= 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
7= 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
A= 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1
```

Marcando agora com um "!", os lugares onde o "Space Devil" atuou:



```
?= 1 1 1 !0! 1 1 1 1 !1! 1 1 1 !0! 1 1
?= 1 0 0 1 0 1 !0! 0 1 0 0 1 0 1 1 0
?= 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1
```

Pesquisando na matriz de Hadamard, verifica-se que a primeira mensagem tem uma distância de hamming de 2 para o código 0 (é a menor distancia possível).

A segunda mensagem tem $d=1$ para o código 7 e a terceira tem $d=0$ para o código A. Logo a mensagem enviada foi 0,7,A. Fácil não? ;-)

Para você fazer

Nesta instância do problema serão enviados pixels com 16 níveis de cinza. A matriz é a H_4 acima.

Você recebeu uma mensagem composta dos seguintes códigos em hexadecimal:

C1= 97B6 C2= AA51 C3= E433

sabendo que foram introduzidos de 0 ate 3 erros em cada código, descubra qual a mensagem original que foi enviada e responda cada código em hexadecimal.

RESPOSTAS:

C1 = _____ (a resposta deve ser um número entre 0 e F)

C2 = _____ (a resposta deve ser um número entre 0 e F)

C3 = _____ (a resposta deve ser um número entre 0 e F)



Certificação Digital

Como se viu, grande parte da segurança de sistemas distribuídos reside na utilização adequada das ferramentas de criptografia lá estudadas. Entretanto, o primeiro ponto chave aqui é certificar-se da identidade do parceiro, sem a necessidade de ter um colóquio vis-a-vis com ele. A solução que a humanidade encontrou foi o desenvolvimento de uma estrutura de certificação digital, com confiança mútua e herdada. Do correto entendimento daqueles conceitos deriva a certeza da segurança obtida no esquema. Assim, a atividade de hoje é uma prática – tanto quanto possível real – dos conceitos lá vistos.

Na aula de hoje cada um participantes será ator em um cenário real de negócios on-line.

O exercício vai se desenvolver em diversos instantes e em cada um dele, haverá tarefas para todos, a depender do papel desempenhado.

1. Para efeito de mensagens e senhas, deve-se usar apenas o alfabeto maiúsculo, mais o espaço, aqui representado como /. Para converter letras em números e vice-versa, usar a equivalência A=1, B=2, C=3, ... X=24, Y=25, Z=26 e /=27. Não usar acentos, nem til nem cedilha.
2. O professor se auto-nomeia autoridade raiz. Gera um par de senhas (P=pública e S=secreta), esconde a S, e publica no quadro negro o seu certificado, o qual inclui a senha P da autoridade raiz.
3. Autoridades certificadoras. _____ alunos são convidados a se instituírem em autoridades certificadoras. Aproximam-se do professor, escolhem o nome de sua CA e ganham seu par de chaves. A chave S devem escondê-la e a chave P é publicada dentro do seu certificado que é publicado no quadro.
4. As chaves geradas são formadas por 3 números, a saber: n_1, n_2, n_3 . As chaves são: n_1 e n_3 e a outra chave do par é n_2 e n_3 . Note que n_3 é compartilhada pelo par de chaves P e S.
5. Cada certificado publicado (e distribuído), deve ter: a) nome de quem é emitido; b) a chave P desse certificado; c) o nome da autoridade que o emitiu; d) a assinatura dessa autoridade. Essa assinatura é a palavra "CERTIF" criptografada pela chave S da autoridade que emitiu esse certificado.
6. Agora, todos os alunos participantes devem comprar seus certificados de alguma CA. A escolha é livre. Os certificados devem ser preenchidos pela CA e publicados no quadro por esta. Muito cuidado ao repassar aos alunos as suas chaves S que precisam ficar invioláveis. (se houver furo aqui, a CA está fora do mercado...)
7. Cada aluno deve escolher seu par para efeito de certificação cruzada (desafio-resposta). A dupla vai ser batizada de Alice (Alício) e Bob (Bela). Se o número de alunos na sala for ímpar, caberá a este certificar o professor.
8. O aluno A deve gerar uma palavra contendo entre cinco e oito letras. Esta palavra deve ser enviada ao aluno B, em texto plano (sem codificação).
9. Para codificar e decodificar mensagens vai-se sempre passar de letras a números e vice-versa usando a tabela acima. O objetivo é sempre trabalhar com números.
10. Para aplicar o algoritmo, cada número (letra) vai ser codificado usando a fórmula:

$$L_c = (L_o)^{C_1} \text{ mod } C_2$$

L_c	Letra (número) codificada
L_o	Letra (número) original
C_1	primeiro número da chave (P ou S, tanto faz)
C_2	segundo número da chave (P ou S, tanto faz)

11. Para fazer estas contas, busque o utilitário=XYN, disponível na rede local, para achar os resultados corretos.
12. O aluno B, recebe esta palavra, codifica-a usando a sua senha S e devolve o resultado a A.
13. A recebe a palavra codificada, busca no quadro a senha P do aluno B e decodifica o que recebeu.
14. Se o resultado da decodificação acima é igual à palavra original que A enviou a B, então B é quem diz ser. Ele está certificado. A deve informar a B que ele está certificado e deve pedir que B gere uma palavra e lhe envie em texto plano.
15. A recebe a mensagem, criptografa-a com a sua chave S e devolve o resultado a B.
16. B recebe a mensagem, busca no quadro a chave P de A, decriptografa a mensagem, compara o resultado e certifica (ou não) o aluno A.
17. Estando tudo certo A pode gerar uma chave *ad hoc* para uso em comunicação simétrica entre ambos, usando os algoritmos que ainda iremos aprender nas próximas aulas.
18. Uma autoridade certificadora nova, chamada CARLOS ALBERTO FICTICIO (sigla CAF) se instala no mercado e se certifica junto a uma das CAS que já existem na sala.

19. Vários participantes, a saber Uran, Viol, Wald, Xana, Yori, e Zwin compram certificados da CAF e têm seus certificados publicados no quadro
20. Cada aluno, deve escolher um destes personagens que acabaram de entrar e comprovar a identidade deles, percorrendo a cadeia de certificados até o raiz.
21. A última atividade é pensar e relatar os riscos que cada participante corre se deixar vaziar a sua senha S.

Para você fazer

Preencha o questionário a seguir, e ao final, devolva-o ao professor:

1. Comprei o meu certificado digital da CA

2. Minhas chaves foram:
Pública: _____ , _____
Secreta: _____ , _____
3. Meu parceiro na certificação cruzada foi o

Eu fui o/a _____ ele/a foi _____.
4. A palavra que escolhi para certificá-lo/a foi:

5. Na segunda parte, depois da entrada da certificadora CAF, escolhi a seguinte pessoa, para seguir até a autoridade raiz
pessoa nova escolhida: _____
cuja chave pública foi _____
6. A CA certificadora dessa pessoa foi a CA certificadora:

cuja chave pública foi _____
Sua assinatura foi: _____
que devidamente decodificada deu origem à: _____
7. A CA certificadora dessa CA foi a CA certificadora:

cuja chave pública foi _____
Sua assinatura foi: _____
que devidamente decodificada deu origem à: _____
8. A CA certificadora dessa CA foi a CA certificadora:

cuja chave pública foi _____
Sua assinatura foi: _____
que devidamente decodificada deu origem à: _____
e se tudo deu certo, você acabou de chegar na CA raiz.
9. O que de errado pode acontecer se os participantes perderem suas chaves secretas?
participante _____ riscos que ele corre



Cifras de Playfair Uma das últimas cifras usadas antes de inventarem-se os computadores, chegou a ser operada ainda na Segunda Grande Guerra. Proposta por volta de 1850, fez sua estréia na Guerra dos Boers. Idealizada por Charles Wheatstone, quem convenceu o Governo Inglês a usar foi o Barão de Playfair, daí o nome com que ficou conhecida.

Possui as fraquezas de sempre: sucumbe a uma análise de frequência, mas por trabalhar com dígrafos (2 letras), a análise tem que ser feita sobre tais grupos. Em contrapartida, tem algumas vantagens: não precisa de tabelas, usa chaves fáceis de memorizar e trocar quando necessário, é simples de operar e consequentemente é pouco sujeita a erros. Por isso esta cifra é ótima para ser usada como “cifra de campo”.

Na cifra Playfair as letras do texto plano são usadas duas a duas (o que caracteriza um bloco dígrafo). O alfabeto cifrante é colocado em uma grade 5×5 , o que implica em matar uma letra. Pode-se excluir o K (substituindo por C), o W (por V) ou o Y (por I). Há ainda quem mate o J e deixe I e J valerem I. Nesta folha, vai-se excluir a letra Y substituindo-a quando necessário por I.

O alfabeto começa com as letras da chave, sem repetições. As demais casas são preenchidas com o alfabeto (menos as letras da chave) em ordem alfabética. Por exemplo, se a chave for CURITIBAPARANA, o início do alfabeto cifrante é CURITBAPN. Eis como fica o alfabeto cifrante:

C	U	R	I	T
B	A	P	N	D
E	F	G	H	J
K	L	M	O	Q
S	V	W	X	Z

A seguir, a mensagem a criptografar deve ser dividida em grupos de 2 caracteres. Esses dois caracteres formarão a linha e a coluna da resposta. Em outras palavras, vai-se consultar cada bloco destes no alfabeto cifrante: 3 casos podem ocorrer:

- As duas letras estão na mesma linha: cada letra é substituída pela sua vizinha da direita
- As duas letras estão na mesma coluna: cada letra é substituída pela sua vizinha de baixo Nestes dois casos, a linha ou a coluna formam um anel, ligando o último ao primeiro.
- As duas letras formam os vértices de um retângulo. O grupo é substituído pelos outros dois vértices do retângulo (cada letra é substituída pela letra que está na mesma linha)

Precisa-se designar uma letra CORINGA para auxiliar nesta codificação. Embora a mensagem possa incluir este coringa, ao fazer isto, é necessária uma melhor especificação para este caso. Assim, para facilitar nossa vida, vai-se EXCLUIR este coringa da mensagem. Portanto, aqui, já são 2 letras as excluídas: a 26ª letra e o coringa.

Escolheu-se aqui o coringa W e para sua aplicação, vão ser usadas duas regras:

- Letras dobradas (SS, RR ou mesmo AA ou OO) impedem a aplicação da cifra. Se ocorrer um grupo assim, colocar o coringa entre o par. Nesta folha, esta regra vai ser ligeiramente modificada a fim de auxiliar na implementação em computador. Quando uma letra estiver dobrada, estas duas letras vão ser substituídas por letra+coringa e letra+coringa.
- Se o ultimo grupo tiver uma única letra, colocar um coringa também para completar o grupo

Continuando o exemplo: seja criptografar a mensagem MINHA TERRA TEM PALMEIRAS ONDE CANTA O SABIA. Formando os blocos fica:

MI NH AT ER RA TE MP AL ME IR AS ON DE
CA NT AO SA BI A. O último bloco como se viu deve ser AW.

Veja-se o primeiro grupo que é MI. Localizando este grupo na tabela de cifração ve-se que as duas letras formam um retângulo. Devem ser substituídas por OR. O próximo grupo é NH que estão na mesma coluna. O N deve ser substituído po H e o H por O. Depois vem o grupo AT que deve ser substituído por DU. Depois ER, que vira GC. Depois RA, que vira UP. Depois TE que vira CJ. depois MP que vira WG. Depois AL que vira FV. Depois ME que se torna KG. E, assim por diante.

A decryptografia segue os mesmos passos, agora em ordem inversa.

Cifra Playfair de 4 grades É uma variação da cifra estudada, na qual usam-se 4 cifras, dispostas assim $\frac{12}{34}$. No método devem ser fornecidas 4 senhas (que podem ser iguais, ou vazias) mas que permitem a construção das 4 cifras mencionadas no método. Agora cada grupo (2 letras) da mensagem clara deve ser buscado nas cifras 1 e 4. A mensagem cifrada correspondente será encontrado nas cifras 2 e 3. A grande vantagem desta variante é que:

- Não há porque tratar duplas de letras iguais. O problema desaparece.
- As 3 regras (mesma linha, mesma coluna e retângulo) se convertem em apenas 1: em todos os casos o resultado é um retângulo.

Veja-se o exemplo: Seja criptografar a mensagem OUVIRAM DO IPIRANGA AS MARGENS PLACIDAS. Manuseado o texto fica OU VI RA MD OI PI RA NG AA SM AR GE NS PL AC ID AS Se usarmos as senhas ('OLIVIAPALITO' 'RECRUTAZERO' 'CARMENSANDIEGO' e 'TIGREHAROLDO'), as cifras ficarão

OLIVA|RECUT|
PTBCD|AZOBD|
EFGHJ|FGHIJ|
KMNQR|KLMNP|
SUWXZ|QSVWX|

CARME|TIGRE|
NSDIG|HAOLD|
OBFHJ|BCFJK|
KLPQT|MNPQS|
UVWXZ|UVWXZ|

e a mensagem cifrada será RUEMLGPSECZ-CLGMREGQKUEJRPPBNEJTDTT

Exemplo Completo Fazendo um exemplo completo, usando estrofes da maravilhosa poesia de Manoel Bandeira (Vou me embora para Passárgada), eis algumas criptografias:

- A criptografia Playfair da frase
MAS TRISTE DE NAO TER JEITO, com a senha CELULASTRONCO gerou
PUTRSKTRTJATUNDTTKCJRN.
- A decryptografia de
FGTJAGNWMBGWEGCGJX com a senha GRADEESPERTA, gerou
SUBIREINOPAUDESEBO.
- A descompressão usando 4 grades senhas:
LIVROVERDE,
GRAMPEADOR,
OITONOVEDEZ e
SACOLACHEIA de
LWEMBIDDSIPLCCCEBK deu
QUANDODENOITEMEDER.

Apenas para conferir, eis as 4 grades do exemplo acima

LIVRO|GRAMP|
EDABC|EDOBC|
FGHJK|FHIJK|
MNPQS|LNQST|
TUWXZ|UVWXZ|

OITNV|SACOL|
EDZAB|HEITD|
CFGHJ|FGJKM|
KLMPQ|NPQRT|
RSUWX|UVWXZ|

🔑 Para você fazer

1. Playfair com 1 senha Criptografe a frase, também obtida da poesia de Manoel Bandeira, Vou me embora para Passárgada:

DEITONABEIRADORIO

com a senha

PINGUIMFAMOSO

e responda abaixo com o sexto, décimo e décimo quarto caracteres da resposta.

6	10	14

2. Playfair com 1 senha Decryptografe a frase

BVJVQIMZLZLUVFWFDGUNSU

com a senha

VISITANTELEGAL

Responda a frase encontrada aqui

3. Playfair com 4 senhas Use as senhas

NEZINHOXAROPE
TVACABOLA
MELANCIAGRANDE
e
MENSAGEMRUI

Para decryptografar a frase

HLTALNJGFAIPTBCPKEDU

Responda a frase encontrada aqui



- 1 - /

A cifra de Viginère

Durante séculos, a cifra de substituição monoalfabética simples foi suficiente para guardar segredos. Neste caso, a mensagem 'ivoviuauvaeamelancia' criptografada com a chave

alfabeto = ABCDEFGHIJKLMNOPQRSTUVWXYZ
chave = EWTKIYVPXGOZNLMRHDCABQFSU

Daria origem à mensagem: XBLBXAEABEIENIZEJTXE. O desenvolvimento subsequente da análise de frequência, primeiro no mundo árabe e depois na Europa (entre os séculos XII e XVII),¹ destruiu sua segurança. Quem primeiro propôs uma melhoria no método foi Leon Alberti (1404-1472) que sugeriu usar 2 alfabetos e alternar as letras da mensagem entre eles. Algo como

alfabeto = ABCDEFGHIJKLMNOPQRSTUVWXYZ
chave 1 = EWTKIYVPXGOZNLMRHDCABQFSU
chave 2 = DXYLMKJBZOTPECUGFANQRWIVSH

Agora, a mesma mensagem ("ivoviuauvaeamelancia"), alternando os dois alfabetos, ficaria: XWLW... Embora houvesse descoberto o avanço mais significativos dos últimos 1000 anos, Alberti não desenvolveu sua idéia, coisa que acabou sendo feita pelo diplomata francês Blaise de Vigenère (1523-1596). A cifra de Viginère não usa 2, mas sim 26 alfabetos diferentes. O primeiro passo consiste em montar o Quadrado de Viginere:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
26	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Trata-se de um alfabeto normal seguido de 26 alfabetos cifrados, cada um deslocado uma letra em relação ao alfabeto anterior. A idéia é utilizar uma linha diferente para cifrar cada letra da mensagem original, de maneira a inviabilizar o ataque por análise de frequência. Para obter esta variação, utiliza-se uma senha. Para ver como isso é feito, vai-se usar a senha 'UNPOSITIVO' e criptografar a mensagem = 'amanhaanoite', que dará como resultado = UZPBZITVJWNR.

- Escreve-se a senha sobre a mensagem e repete-se-a até o final da mensagem. Fica

senha = UNPOSITIVOUN
mensagem = amanhaanoite
- Para cifrar a primeira letra, (a no exemplo), identifica-se qual a letra da senha que lhe corresponde (U no exemplo).
- Busca-se no quadrado de Viginère, qual o alfabeto que começa pela letra U (linha 20 no exemplo).
- Usa-se este alfabeto para converter a letra da mensagem. Neste exemplo (A → U)
- Repete-se o ciclo acima para todas as letras da mensagem. No exemplo, a segunda letra, começa identificando a letra N na senha, o que leva ao alfabeto 13, e o m vira z.

¹A propósito, quem se interessar pelo método da análise da frequência, leia o conto de Conan Doyle “Os dançarinos”, onde Sherlock Holmes descreve o método de maneira magistral.

Exemplo

Seja uma instância em que se quer criptografar EXISTEMPROBLEMASQUETEMSOLUCAO, usando a senha GLORIOSO. A resposta é KIWJBSEDXZPCMASGWFSKMAKCRFQRW. Já a decriptografiação de JMCDEUTHITKSDLAFEIEOPUERVFDLHMS, usando a senha CERCADAS deu origem à frase HILBERTPRECISALIDARCOMUMPROBLEMA.

👉 Para você fazer

Criptografe a mensagem

EULERCOMECOUCOMUMAPLANTADACIDADE
com a senha
VOLTAREI

E, depois decriptografe a mensagem

TIEPUSHWEFXZAWMOGRZDTZOGPVGZZLA
com a senha
PREVISA0



- 1 - /

Criptografia Simétrica (DES)

Este exercício simula uma criptografia de chave simétrica com princípio (apenas o princípio !) similar ao *DES = Data Encryption Standard*, padrão do governo americano para criptografia simétrica.

Para converter uma mensagem, a primeira coisa que emissor e receptor devem combinar é a respeito de uma chave comum.

Neste exemplo, usaremos a *chave = AYAHOUSQ*. Note que a chave tem 8 caracteres, o que caracteriza a criptografia em bloco, neste caso, bloco de 8 caracteres.

A chave deve ser convertida em um vetor de 8 números, usando a convenção seguinte:

/	0	E	5	J	10	O	15	T	20	Y	25
A	1	F	6	K	11	P	16	U	21	Z	26
B	2	G	7	L	12	Q	17	V	22		
C	3	H	8	M	13	R	18	W	23		
D	4	I	9	N	14	S	19	X	24		

Neste caso, tem-se como resultado: 1 25 1 8 15 21 19 17

O texto plano que deve ser criptografado deve ser separado em blocos de 8 caracteres também. Seja a seguinte mensagem que deve ser criptografada

NAO/VOLTAR/JAMAI/OU/NUNCA (espaço é igual a /)

E, fica:

NAO/VOLT AR/JAMAI S/OU/NUN CA////////

(Note o preenchimento com espaços ao final)

Deve-se primeiro inverter os pares da mensagem, e fica:

AN/OOVTL RAJ/MAIA /SUON/NU AC////////

Agora, deve-se converter cada bloco em seu equivalente numérico:

1 14 0 15 15 22 20 12
0 19 21 15 14 0 14 21

18 1 10 0 13 1 9 1
1 3 0 0 0 0 0 0

Deve-se fazer 2 somas agora: primeiro deve-se somar a chave a cada grupo, e fica

2 39 1 23 30 43 39 29
1 44 22 23 29 21 33 38

19 26 11 8 28 22 28 18
2 28 1 8 15 21 19 17

A próxima soma é com o deslocamento de cada caracter na chave (somar 0 1 2 3 4 5 6 7 a cada grupo), e fica:

2 40 3 26 34 48 45 36
1 45 24 26 33 26 39 45

19 27 13 11 32 27 34 25
2 29 3 11 19 26 25 24

A seguir, calcule o resto da divisão de cada número por 27:

2 13 3 26 7 21 18 9
1 18 24 26 6 26 12 18

19 0 13 11 5 0 7 25
2 2 3 11 19 26 25 24

Lembre de desconverter os pares. Fica:

13 2 26 3 21 7 9 18
18 1 26 24 26 6 18 12

0 19 11 13 0 5 25 7
2 2 11 3 26 19 24 25

Finalmente, converta a mensagem numérica em caracteres usando a tabela original.

MBZCUGIR /SKM/EYG RAZXZFRL BBKCSXY

Que é a mensagem devidamente criptografada.

Note que para decryptografar uma mensagem cifrada o mesmo método deve ser empregado, só que em ordem inversa.



Para você fazer

- Use a chave: IDMCCLVDL
- Criptografe a mensagem: ENTRE/NOVE/HORAS/DA/MANHA////////
 - Transforme a chave em números:
 - Separe a mensagem em grupos de 8, completando com brancos:
 - Inverta os pares de caracteres
 - Transforme a mensagem em números
 - Some a chave aos caracteres da mensagem
 - Some o deslocamento na chave
 - Ache o resto da divisao por 27.
Regra: ate 26 inclusive não precisa fazer nada. Depois, use a tabela

27=0, 28=1, 29=2, 30=3, 31=4, 32=5, 33=6, 34=7, 35=8, 36=9, 37=10, 38=11, 39=12, 40=13, 41=14, 42=15, 43=16, 44=17, 45=18, 46=19, 47=20, 48=21, 49=22, 50=23, 51=24, 52=25, 53=26, 54=0, 55=1, 56=2, 57=3, 58=4, 59=5, 60=6, 61=7
 - Converta os números em caracteres
 - Desinverta os pares e junte tudo. O resultado é:
- Considere a mensagem que você acabou de cifrar como o vetor C

Responda agora:

1. C[4] _____ 2. C[11] _____ 3. C[23] _____

- Use a mesma chave e decryptografe
FAYTMRDOWIKODUSLFRCC/CAEJUFO/PSJ
 - Separe a mensagem em grupos de 8. Não é necessário completar brancos
 - Inverta os pares
 - Converta os caracteres em números. Subtraia de cada um o valor da chave e depois o deslocamento (0 1 2 3 4 5 6 7) na chave. Se o resultado for negativo, some 27 ao resultado
 - Converta em caracteres, usando a tabela
 - Desinverta os pares e responda. A mensagem plena é:
- Considere a mensagem que você acabou de decryptografar como o vetor P

Responda agora:

4. P[14] _____ 5. P[25] _____ 6. P[29] _____

1



- 1 - /

TEA

O algoritmo *Tiny Encryption Algorithm* foi desenvolvido na Universidade de Cambridge em 1994, pelos autores David Wheeler e Roger Needham. Foi apresentado no Fast Software Encryption Workshop. Não está sujeito a nenhum tipo de patente. A função de criptografia, programada na linguagem C aparece abaixo:

```
void encrypta(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z=text[1];
    unsigned long delta =0x9e3779b9, sum=0;
    int n;
    for (n=0; n<32; n++) {
        sum = sum + delta;
        y = y + ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
        z = z + ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
    }
    text[0] = y; text[1] = z;
}
```

1

O algoritmo TEA usa várias passagens de adição de inteiros, combinados com XOR e deslocamentos lógicos em nível de bit (<< e >>) para obter mistura e difusão dos padrões de bit no texto puro. O texto puro é um bloco de 64 bits representado como dois inteiros longos de 32 bits no vetor *Text[]*. A chave tem 128 bits de comprimento, representado como quatro inteiros de 32 bits.

Em cada uma das 32 passagens as duas metades do texto são repetidamente combinadas com as partes deslocadas da chave e uma com as outras. O uso da operação XOR e das partes deslocadas do texto proporciona a mistura e o deslocamento. A troca das duas partes do texto proporciona a difusão. A constante *delta* que não se repete, é combinada, em cada ciclo, com cada parte do texto para ocultar a chave. Isso para evitar o caso dela poder ser revelada por uma seção de texto que não varie. A função de decifração é a inversa da função de criptografia e tem o seu código abaixo:

```
void decrypt (unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z=text[1];
    unsigned long delta =0x9e3779b9, sum = delta << 5;
    int n;
    for (n=0; n<32; n++) {
        z = z - ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
        y = y - ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
        sum = sum - delta;
    }
    text[0] = y; text[1]=z;
}
```

2

Esses programas curtos oferecem criptografia de chave secreta de forma segura e razoavelmente rápida. Ele é bem mais rápido do que o algoritmo DES e o caráter conciso do programa se presta à otimização e implementação em hardware. A chave de 128 bits é segura contra ataques de força bruta. Estudos realizados por seus autores e por outros autores revelaram apenas duas deficiências muito pequenas.³ Na verdade, existem 3 outras chaves distintas, que geram a mesma mensagem cifrada. Nesse sentido, a longitude efetiva da chave é apenas de 126 bits (e não 128). Esta debilidade deu origem a um método que foi usado para quebrar a proteção da console Xbox da Microsoft, já que esta usava o algoritmo TEA como função *hash* para proteção. Em trabalhos posteriores, tais deficiências foram removidas no algoritmo XTEA.

A seguir, um programa C que utiliza o TEA para cifrar ou decifrar dois arquivos previamente abertos (usando *stdio.h*).

```
while (!feof(infile)) {
    i = fread(Text,1,8,infile);
    if (i<=0) break;
    while (i<8){Text[i++]=' ';}
    switch (tipo) {
        case 1: encrypt(k,(unsigned long *) Text); break;
        case 2: decrypt(k,(unsigned long *) Text); break;
    }
    j = fwrite(Text,1,8,outfile);
}
```

¹O símbolo << indica um *shift* à esquerda – note que não é uma rotação. O operando é deslocado à esquerda e entram 4 zeros à direita. Bits eventualmente perdidos, ficam perdidos. Exemplo: se x é 0 0 0 0 0 1 1 1, $x << 1$ é igual a 0 0 0 0 1 1 1 0. O símbolo $\otimes$, que no código C aparece como um acento circunflexo (é o seu sinal neste linguagem), significa o exclusive-or, operador binário que tem o seguinte resultado: $0 \otimes 0 = 0$, $1 \otimes 1 = 0$, $0 \otimes 1 = 1$ e $1 \otimes 0 = 1$. Finalmente, as somas acima indicadas são todas elas módulo 2^{32} .

²neste caso, as subtrações também são feitas em módulo 2^{32} .

³Wheeler, D. J. e Needham, R. M. (1997) *Tea Extensions*, Outubro de 1994, págs. 1-3.

Exemplo

Vamos criptografar a frase 'IVO VIU A UVA'. Convertendo-a em hexadecimal, fica x'49 56 4F 20 56 49 55 20 41 20 55 56 41'. Não se deve esquecer que o texto pleno deve ter tamanho múltiplo de 8, daí x'49 56 4F 20 56 49 55 20 41 20 55 56 41 20 20 20'.

Usando a chave de criptografia x'01 02 03 04 05 06 07 08 09 AA AB AC AD AE AF 00', criptografando fica x 36 25 27 D6 82 17 48 30 51 5E B4 E3 CB 4C 7B CE.

Obviamente, submetendo este string à decryptografia com a mesma chave, recupera-se a frase original, IVO VIU A UVA.

Apenas para constar, agora vai-se decryptografar o valor acima, com uma chave ligeiramente alterada (um '0' na 32ª posição da chave, virou 1, e a nova chave agora é x'01 02 03 04 05 06 07 08 09 AA AB AC AD AE AF 01').

O resultado da decryptografia ficou x'0E 2F C6 5C 0A 92 B1 10 2F CC 87 B5 D0 9F 59 F5', que obviamente nada tem a ver com a frase original (IVO VIU A UVA).

Mais dois exemplos

Bom para quem implementar o código acima.

```
senha=x'4546485254505649474A47435854524C' (EFHRTPVIGJGCTRL)
frase=PL I NUNCA FOI USADA
deu=x'FE954195FCAA47758FE544229D9EE4D5FC590E757F6B026E'
```

e também

```
senha=x'4D4A5855494642494E57535654594258' (MJXUIFBINWSVTYBX)
frase=SQL E PARA ACESSAR
deu=x'B41E5545CBDC3BC94A55F59CFEBA2D2F87E327F345BA26B2'
```



Para você fazer

Utilize a senha (dados em hexadecimal):

544E4B4744544B4841575544554F4B4F

Decryptografe a frase (também em hexadecimal):

FDE8C8E66419071159E4718E76E725F430145F310A2982E2

Resposta em ASCII:

Atente para o fato de que no retorno, obrigatoriamente deverão aparecer letras e não outros caracteres. Eis o subconjunto da tabela ASCII de que se fala

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	5	
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A

Uma observação final: Note que tanto no algoritmo de criptografia quanto no de decryptografia existe uma constante numérica que obrigatoriamente tem que ter o mesmo valor em ambos os algoritmos. No original o valor é de $\delta = 0x9e3779b9$. Se você quiser, pode alterar tal valor para aquele que você quiser (desde que seja o mesmo em ambos os algoritmos). Se você mantiver tal valor escondido, ele passa a ser um nível adicional de criptografia. Obviamente, se a criptografia for um com valor e a decryptografia com outro, o resultado será ininteligível.

Uma curiosidade Quanto tempo demoraria uma tentativa de quebra do algoritmo por “força bruta”? Suponhamos um supercomputador capaz de examinar 1.000.000.000 de chaves por segundo. Quanto tempo este computador gastaria na tarefa?

As chaves são $2^{126} = 8.5070 \times 10^{37}$. Se o computador examina 1 bilhão de combinações por segundo, gastará 8.5070×10^{28} segundos. Em dias, serão 9.8461×10^{23} dias. Em anos serão 2697570767123287000000 anos.



- 1 - /

Criptografia Assimétrica: RSA

RSA são as iniciais dos autores deste algoritmo (Rivest, Shameir e Adleman). A segurança deste método está baseada em dois princípios:

- A dificuldade computacional esperada (mais ainda não provada) em fatorar grandes números. Este fato vai ser melhor estudado nas próximas aulas, quando vocês serão desafiados a “quebrar” o método RSA.
- A garantia de que quem diz ser alguém de fato o é. Esta garantia é obtida pela certificação digital.

Trocando em miúdos, para o RSA funcionar, a chave pública de todo mundo deve ser sempre conhecida e espera-se ter certeza de que quem diz ser o proprietário do certificado é de fato quem diz ser.

Algoritmo RSA A seguir, as etapas necessárias para calcular as chaves pública e privada e depois como criptografar e decryptografar mensagens.

Para o cálculo de CP e CS há que se escolher 2 números primos p e q	Por exemplo: p = 3 e q = 7
Calcula-se n = p x q	n = 21
Calcula-se f = (p-1) x (q-1)	f = 2 x 6 = 12
Escolhe-se c, tal que o mdc entre c e f seja 1	c = 5 já que mdc(5,12) = 1
Escolhe-se d tal que (c x d) mod f = 1	5 x d mod 12 tem que ser 1, d = 17 já que 5 x 17 = 85 e 85 mod 12 = 1
A CP do indivíduo é (c,n)	CP = (5,21)
A CS do indivíduo é (d,n)	CS = (17,21)
Quando alguém quer mandar uma mensagem para o indivíduo, quebra a mensagem m em blocos (números)	digamos que m = 3, 14, 9
Para cifrar m, esse alguém faz $CP(m) = m^c \bmod n$	$3^5 \bmod 21 = 12$, $14^5 \bmod 21 = 14$, $9^5 \bmod 21 = 18$
Quando a mensagem chega, há que se fazer $m' = [CP(m)]^d \bmod n$	$12^{17} \bmod 21 = 3$, $14^{17} \bmod 21 = 14$, $18^{17} \bmod 21 = 9$

Para fazer este exercício você deverá ter acesso a um computador, e nele deverá programar a seguinte função para auxiliar na tarefa de elevar grandes números a grandes expoentes:

```
1: inteirolongo função XELEVADOAY (inteirolongo X, Y, N) {algoritmo
  ADDITION CHAINING, SCH96, pag. 244}
2: inteirolongo TMP
3: se Y = 1 então
4:   devolva (X mod N) {em C ou Python: X % N}
5: senão
6:   se (Y mod 2) = 0 então
7:     TMP ← XELEVADOAY (X, (Y/2), N)
8:     devolva (TMP * TMP) mod N
9:   senão
10:    TMP ← XELEVADOAY (X, ((Y-1)/2), N)
11:    TMP ← (TMP * TMP) mod N
12:    TMP ← (TMP * X) mod N
13:    devolva TMP
14: fim se
15: fim se
16: fim função
```

Depois de digitar, use os seguintes exemplos para testar o programa

$(2^3) \bmod 20 = 8$
 $(5387^{2189}) \bmod 3311 = 1668$
 $(7752^{4156}) \bmod 12981 = 4812$

Para criptografar uma mensagem qualquer, você deve:

- Converter a mensagem (caracteres) em números
- Escolher o par de números que usará como chave (a,b)
- Para cada número, calcular $(numero^a) \bmod b$ e enviar o resultado

Para decryptografar uma mensagem qualquer, supostamente já em forma de números, você deve:

- Recuperar a chave oposta usada na criptografia, na forma (c,d)
- Para cada número recebido, calcular $(numero^c) \bmod d$
- Recuperar o caractere usando a tabela

		D	5	H	9	L	13	P	17	T	21	X	25
A	2	E	6	I	10	M	14	Q	18	U	22	Y	26
B	3	F	7	J	11	N	15	R	19	V	23	Z	27
C	4	G	8	K	12	O	16	S	20	W	24	/	28

Exemplo

Erasmus (cuja chave publica é [1153,4757] mandou a mensagem 3568, 386. O que ela significa ?
Resposta: O / (a letra O e um espaço)

Agora você quer mandar a mensagem "AS"para a Frida (chave pública da Frida é [763,3953]). Como fica ?
Resposta: 453, 867

A mensagem "447, 686" só deve ser lida por você (sua chave secreta é [193,1829]. O que é ?
Resposta: J A (as letras J e A)

Você quer assinar a mensagem LH, usando a sua chave secreta. Como fica ?
Resposta: 1375 e 390

Para você fazer

- Eis as chaves públicas conhecidas

Alice=(337,2479), Bento=(623,3239)
Carlos=(841,6059), Dante=(977,7031)
Erasmus=(1267,6497), Frida=(803,3337)

- Sua chave pública é (265,1961)
e sua chave secreta é (1321)

Obs: Note que a sua CS e CP compartilham o segundo número. Ele só não é escrito na CS, já que não dá para chamá-lo de secreto.

- Você recebeu a seguinte mensagem (alfabética) de dante
Mensagem:

247 354 1609 2501 354 1213 1609 4749 3832 3832
6105 6411 1609 99 4428 1609 6411 1213 354 1609
6411 4749 4376 1609 247 1213 1609 4584 4428 6411
4428 5032 1213 3832 247 1213

Qual é a mensagem ?

A=_____.

Dica: decodifique usando a CP do remetente

- Você quer mandar a mensagem abaixo para alice
Mensagem:

CUIDOSOS/SE/INCUBEM/DO/VASO/DAS/CORES

B=_____.

Dica: codifique com a CP do destinatário

- A mensagem abaixo só deve ser lida por você. Qual é?
Mensagem:

23 1569 1710 1608 1893 1897 950 146 1893 611
1569 1556 950 1556 1893 893 893 1569 1556 950
1641 342 1556 950 23 1897 1556 950 1569 1897
828 342 893 1893 1897 1556

C=_____.

Dica: decodifique usando a sua CS

- Esta mensagem vai assinada por você
Mensagem:

POIS/QUE/FRACO/E/SEM/TRIBO/E/SEM/FAMILIA

D=_____.

Dica: codifique com a sua CS

Agora responda as questões, usando A, B, C e D calculados acima

A[1]	A[36]	B[19]	B[37]
C[11]	C[36]	D[32]	D[40]



- 1 - /

Esteganografia

Trata-se da ciência (arte ?) de esconder mensagens em locais inesperados. Leia-se a propósito na notícia divulgada no sitio www.no.com.br (já não existe mais) cerca de 6 meses antes do ataque de 11/09/2001 a Nova Iorque. Note que o autor escreve o nome da técnica de maneira errada (esteganografia e não estenografia).

Osama bin Laden não é o típico líder terrorista árabe assim como não são típicas as suas táticas. Agora, garante o FBI, ele está fazendo uso da Internet para enviar instruções aos seus soldados santos mundo afora de uma maneira inusitada. Mapas, mensagens, tudo fica escondido de maneira invisível em imagens pornográficas. Em imagens de mulheres em poses luxuriantes pode estar um detalhada plano para explodir uma embaixada americana.

O processo se chama estenografia e embora guarde algum parentesco com a criptografia, dela não tem nada. As mensagens de bin Laden conclamando à Jihad não estão codificadas – apenas escondidas. Qualquer arquivo de computador é, no fim, uma série interminável de uns e zeros. Os hackers muçulmanos de bin Laden incluem no arquivo – uma fotografia em formato jpeg, por exemplo – mais meia dúzia de bytes. Aumenta o tamanho da imagem, por certo, mas passa em branco para o usuário que for vê-la na tela do Netscape ou de um Photoshop.

O princípio é até bem simples. Dados criptografados, como os passados em emails entre grandes empresários ou espões ou o que for, partem do princípio que serão interceptados. Boa é a criptografia que codifica de tal maneira a mensagem que, fora seu destinatário, ninguém mais será capaz de entendê-la. A estenografia, não. Estenografia, por exemplo, foi usada por um tal Histieu, tirano de Mileto, primeiro aliado de Dario e mais tarde brigado com os persas, no envio de uma carta para casa. Escreveu-a na cabeça de um escravo – que teve tempo de deixar o cabelo crescer naqueles tempos pré-grande rede (século 5 aC). O escravo passou incólume para revista de dos soldados persas. Na casa de Histieu, bastou que lhe raspassem a cabeça.

O desespero dos agentes americanos com o uso da técnica, deste criativo contrabando de mensagens, tende a ficar maior que com a criptografia. Na rede, começam a enumerar-se os programas que escondem o que se quer em imagens, sons, arquivos de todo o tipo. E a imagem não deixa pista – como dá para ver nos retratos de Shakespeare exibidos no relatório do professor Neil Johnson para a George Manson University. O programa sendo usado pelos homens de bin Laden, fácilmo de ser encontrado online, é provavelmente o simples White noise storm, que roda em MS-DOS.

Exercício proposto

Você está recebendo um arquivo BMP contendo uma imagem. Este arquivo não é de uma imagem mapeada, assim, cada pixel na imagem estará sendo construído a partir das quantidades de azul (B), verde (G) e vermelho (R). O nome do arquivo é XMAHE001.BMP A única informação referente à localização das letras da mensagem no arquivo é um conjunto de endereços de pontos:

2832 4580 4986 5528 8213 10635 11575 12116 12925 14810 16420
20714 21398 26767 28112 28793 29600 30003 31216

Cada um destes números endereça o byte B (azul) de um determinado ponto da imagem. A primeira dificuldade, é localizar este pixel no arquivo (na imagem não é necessário localizá-lo). O primeiro pixel do arquivo é o de número 1, e não se pode esquecer a eventual necessidade de alinhamento das linhas em múltiplos de 4 bytes. Quando ele for encontrado, deve-se atentar para os 3 valores desse pixel: B, G e R, nesta ordem. Para identificar a letra da mensagem, use o seguinte algoritmo:

1: se B é par **então**
2: deve-se usar a dupla B e G para achar a letra
3: se B > 127 **então**
4: letra = B - G
5: **senão**
6: letra = G - B
7: **fim se**
8: **senão**
9: deve-se usar a dupla B e R para achar a letra
10: se B > 127 **então**
11: letra = B - R
12: **senão**
13: letra = R - B
14: **fim se**
15: **fim se**

Para calcular o valor da letra, use a seguinte tabela

let	dec	hex
A	1	1
B	2	2
C	3	3
D	4	4
E	5	5
F	6	6
G	7	7
H	8	8
I	9	9
J	10	A
L	11	B
M	12	C

let	dec	hex
N	13	D
O	14	E
P	15	F
Q	16	10
R	17	11
S	18	12
T	19	13
U	20	14
V	21	15
X	22	16
Z	23	17
0	24	18

let	dec	hex
1	25	19
2	26	1A
3	27	1B
4	28	1C
5	29	1D
6	30	1E
7	31	1F
8	32	20
9	33	21
esp	34	22

Relembrando o formato BMP *true color*

No início do arquivo, existe um bloco de controle formado por

Tipo	Tam	Desl	Nome	Descricao	exemplo
char	2	0	type	constante igual BM	424D
lint	4	2	size	tamanho do arquivo	C6 04 00 00
lint	4	A	off	desloc ate a imagem	36 00 00 00
lint	4	E	size	tamanho bloco	28 00 00 00
lint	4	12	width	n. de colunas	0A 00 00 00
lint	4	16	height	n. de linhas	0C 00 00 00
int	2	1A	planes	planos	01 00
int	2	1C	bitct	bits / pixel	18 00
lint	4	22	sizeim	tamanho da imagem	90 00 00 00
imagem	**	36	BGR	blue/green/red	

** = nnn × 3

Obs₁ As linhas estão alinhadas em múltiplos de 32 bits.

Obs₂ FF = branco e 00 = preto

Procedimentos

- Trabalhe apenas em decimal (ou em hexa, mas não misture)
- Ache TC (tamanho em colunas da imagem)
- Ache TB (tamanho em bytes de cada linha). $TB = 3 \times TC$
- Ajuste TBA ao próximo múltiplo de 4. $TBA = TB \text{ arred } \uparrow 4$
- Achar L,C do pixel objetivo.
 - Valor decimal do pixel objetivo. $EP = \text{pegar da folha}$
 - Linha do ponto objetivo. $LP = 1 + \lfloor (EP \div TC) \rfloor$
 - Coluna do ponto objetivo. $CP = EP - ((LP - 1) \times TC)$
- Calcular o deslocamento do byte BBP (azul) do pixel objetivo
 - $BBP = ((LP - 1) \times TBA) + (CP \times 3) + 54 - 3$
 - GGP = o seguinte a BBP
 - RRP = o seguinte a GGP
- Encontrados BBP, GGP e RRP use o algoritmo acima para decifrar a mensagem.

Obs₁ o 54 acima na fórmula corresponde a X'36'.

Obs₂ o 3 acima na fórmula se deve a que a numeração começa em zero.

Para você fazer

Informe a mensagem encontrada: O _____

1



- 1 - /

Desafio: Quebremos o RSA ?

Após ter estudado o algoritmo RSA, você está sendo convidado a quebrar uma comunicação que foi feita usando RSA. Para “facilitar” as coisas, estarão sendo usados primos entre 70 e 100 que são valores bem mais manuseáveis do que os normalmente usados nas comunicações entre computadores (e que usualmente são números de 1024 bits). Entretanto, a lógica do trabalho é a mesma.

Relembrando o RSA

Algoritmo RSA

- 1. Para um dado usuário haverá 2 chaves escolhidas por ele: *CP* e *CS*
- 2. Para o cálculo de CP e CS há que se escolher 2 números primos *p* e *q*
- 3. Calcula-se *n* = *p* × *q*
- 4. Calcula-se *f* = (*p* − 1) × (*q* − 1)
- 5. Escolhe-se *c*, tal que o mdc entre *c* e *f* seja 1
- 6. Escolhe-se *d* tal que (*c* × *d*) mod *f* = 1
- 7. A *CP* do indivíduo é (*c*, *n*)
- 8. A *CS* do indivíduo é (*d*, *n*)
- 9. Quando alguém quer mandar uma mensagem para o indivíduo, quebra a mensagem *m* em blocos (números)
- 10. Para cifrar *m*, esse alguém faz *CP*(*m*) = *m*^{*c*} mod *n*
- 11. Quando a mensagem chega, há que se fazer *m'* = [*CP*(*m*)]^{*d*} mod *n*
- 12. obviamente *m* = *m'*

Exemplo

Sejam *p* = 79 e *q* = 89; *n* = 7031, *f* = 6864 e *c* = 977
CP = (977, 7031)
CS = (3569, 7031)
Seja a mensagem *x* = 769, 500, 759, 453, 201
*x*⁹⁷⁷ mod 7031 = 3825, 6285, 4174, 744, 758
Na deciptação, tem-se:
*y*³⁵⁶⁹ mod 7031 = 769, 500, 759, 453, 201

Desafio

Há uma mensagem composta de diversos números que foram codificados usando a criptografia assimétrica RSA. Cada número corresponde a uma letra, obtida segundo a equivalência:

	A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8	9
I	J	K	L	M	N	O	P	Q
10	11	12	13	14	15	16	17	18
R	S	T	U	V	W	X	Y	Z
19	20	21	22	23	24	25	26	27

O remetente da mensagem usou sua chave pública (1433 , 5893) para codificar a mesma. O seu objetivo é:

- Descobrir qual a chave secreta do remetente
- Com ela, decodificar a mensagem

Algoritmo de potência de grandes números

```
Calcula xy mod n
1: inteirolongo função ADCHA(inteirolongo x, y, n)
2: se (y = 1)
3:   devolva x mod n
4: senão
5:   se y é par
6:     AUX ← ADCHA(x, (y ÷ 2), n)
7:     devolva (AUX × AUX) mod n
8:   senão
9:     AUX ← ADCHA(x, ((y − 1) ÷ 2), n)
10:    AUX ← (AUX × AUX) mod n
11:    devolva (AUX × x) mod n
12: fim{se}
13: fim{se}
```

Como proceder

Da teoria, se conhecem as seguintes variáveis:

n o segundo número da chave pública. É um produtos dos primos *p* e *q*

c o primeiro número da chave pública.

Precisamos conhecer (descobrir):

p e *q* os fatores de *n*

f facilmente calculado através de (*p* − 1) × (*q* − 1)

d o primeiro número da chave secreta. É o único número entre 1..f para o qual a fórmula (*c* × *d*) mod *f* = 1 é verdadeira.

Fatoração

Existem diversos métodos para fatorar um número grande. Veja-se em CORMEM, et alli, ALGORITMOS, pág 710, diversos métodos, por exemplo, a heurística *ρ* de Pollard (que tem custo *O*($\sqrt[3]{n}$), ou uma lista completa de abordagens em ATALLAH, M, Handbook of..., página 42-8, entre outros.

Entretanto, como os números envolvidos são pequenos (para o computador, claro!), sugere-se usar a estratégia de força bruta:

```
Calcula se n tem fatores:
1: inteiro função FATORA(inteiro n)
2: inteiro I ← 2
3: inteiro RESP ← 0
4: inteiro LIMITE ←  $\sqrt{n}$ 
5: enquanto I ≤ LIMITE
6:   se n mod I = 0
7:     devolva I
8:   fim{se}
9:   I ++
10: fim{enquanto}
11: devolva RESP
```

Uma vez encontrado o fator *p* de *n*, basta dividir *n* por *p*, para obter *q*, já que pela teoria, *n* é um produto de dois primos. Logo, encontrado um, o outro sai imediatamente.

Agora pode-se calcular *f* que vem a ser *f* = (*p* − 1) × (*q* − 1).

Calculo de *d*

Como se viu acima, *d* é o único número entre 1 e *f* que torna verdadeira a relação (*c* × *d*) mod *f* = 1. O algoritmo a usar pode ser:

```
1: inteiro função PROCUD(inteiro c, f)
2: inteiro D ← 1
3: enquanto D ≤ f
4:   se ((c × D) mod f) = 1
5:     devolva D
6:   fim{se}
7:   D ++
8: fim{enquanto}
```

A mensagem

Que você interceptou é:
4120, 5827, 2534, 2728, 265, 4120, 5442, 1, 5339, 1929, 1174, 3274, 802, 4120, 5442, 1722, 2728, 1, 5696, 2496, 4120, 1, 4195, 1722, 1174, 1, 265, 4120, 5442, 1, 2728, 1174, 802, 2496, 1105, 1722, 1174, 1

A frase procurada é _____

Esta frase foi extraída do (excelente) livro "O Último Teorema de Fermat", de Simon Singh.

um exemplo completo

Seja um caso onde a chave conhecida é (229,1927) e a mensagem é 850,635,304,635,254,635,1,482,635, 1674,482,... .

Conforme se viu a primeira coisa a fazer é fatorar o *n*=1927, obtendo-se a informação 1927 = 41 × 47. Depois, calcula-se *f* = 40 × 46 = 1840.

A seguir, usando-se o *c* (chave pública, ou 229 no exemplo) procura-se o *d* que é o único número entre 1..f que torna verdadeira a relação (229 × *d*) mod 1840 = 1. O valor de *d* = 1149.

Agora as letras são: 850¹¹⁴⁹ mod 1927 = 12, que corresponde à letra K. As demais letras são: KANADAbCALC..., onde *b* é um espaço em branco.

Por que não usar estes algoritmos na prática ?

Porque no pior caso, estamos falando de números de 1024 bits, o que daria algo como 1.797693135 × 10³⁰⁸. Considerando, que há 6,02 × 10²³ átomos de carbono-12 em 12 g dessa substância (0,012 kg ou 1 mol de C) constante conhecida como o número de Avogadro, e que o número total de elétrons do universo está estimado em 10⁸⁰ fica fácil olhar para a magnitude do número acima citado. O conjunto de todos os computadores da terra trabalhando no mesmo problema (este) demoraria bilhões de anos para rodar estes algoritmos em um único caso, pelo menos enquanto alguém não achar algoritmos mais eficientes para tanto. Uma esperança aqui é a computação quântica (como no filme *corra lola, corra*), mas provavelmente estaremos todos mortos antes que haja alguma coisa prática aí.



Exemplos Históricos

Criptografia One Way

Há um caso histórico famoso, que usa este método e que nunca foi quebrado. Ele é conhecido como a "cifra de Beale". Em janeiro de 1820, um forasteiro de nome Thomas Beale, chegou à cidade de Lynchburg e se hospedou no principal hotel da cidade. Embora se relacionasse bem com todos não disse a que veio e lá ficou até o final de março, desaparecendo repentinamente. Em 1822, retornou, ficou mais alguns meses e foi embora novamente. Ao sair, deixou com o hotelheiro uma caixa de ferro contendo "documentos importantes". Suas orientações eram de que ele guardasse a caixa por 10 anos, e se nesse período ninguém a procurasse, o hotelheiro deveria abri-la e seguir as instruções dentro dela. Além disso haveria na caixa 3 documentos indecifráveis sem o auxílio de uma chave. A chave teria sido deixada com um amigo de uma cidade vizinha. O amigo forneceria a chave ao hotelheiro em 1832. O hotelheiro esperou até 1845, sem que ninguém o procurasse, quando a curiosidade venceu: abriu a caixa. Nas orientações Beale dizia que ele e 4 amigos enquanto conduziam búfalos a norte de Santa Fé haviam encontrado um veio de ouro, que garimparam por 4 anos. Os 3 documentos cifrados continham a descrição do local onde o tesouro fora escondido (o primeiro), a descrição do tesouro (o segundo) e as pessoas que deveriam herdar o tesouro (o terceiro).

Até 1862, o hotelheiro conseguiu decifrar apenas o segundo documento. Ele usava como chave a Declaração da Independência americana. O texto dizia que Beale escondera em um local próprio, cercado de pedras, a não mais de 12 km do hotel, cerca de 3000 libras de ouro (aproximadamente 1400 kilos). O ouro estava acondicionado em potes de ferro. A descrição exata estava no texto 1 e os destinatários do tesouro no anexo 3.

Até hoje ninguém conseguiu decifrar os outros 2 textos. Os originais das cartas, bem como mais detalhes desta história, estão no livro de Simon Singh, "O livro dos códigos".

Este método é muito adequado para criptografar um único texto. Usa como chave um outro texto, que pode ser famoso e bem conhecido (por exemplo, o Hino Nacional Brasileiro) ou então um texto especialmente preparado para ser chave.

A mensagem cifrada usa números como elementos da mensagem. Cada número destes remete à letra inicial da palavra que tem este número no chave original.

Por exemplo, usando como chave o texto
Ouviram¹ do² Ipiranga³ as⁴ margens⁵ plácidas⁶
De⁷ um⁸ povo⁹ heróico¹⁰ o¹¹ brado¹² retumbante¹³,
E¹⁴ o¹⁵ sol¹⁶ da¹⁷ Liberdade¹⁸, em¹⁹ raios²⁰ fúlgidos²¹,
Brilhou²² no²³ céu²⁴ da²⁵ Pátria²⁶ nesse²⁷ instante²⁸.
Se²⁹ o³⁰ penhor³¹ dessa³² igualdade³³
Conseguimos³⁴ conquistar³⁵ com³⁶ braço³⁷ forte³⁸,
Em³⁹ teu⁴⁰ seio⁴¹, ó⁴² Liberdade⁴³,
Desafia⁴⁴ o⁴⁵ nosso⁴⁶ peito⁴⁷ a⁴⁸ própria⁴⁹ morte⁵⁰!
Ó⁵¹ Pátria⁵² amada⁵³,
Idolatrada⁵⁴,
Salve⁵⁵! Salve⁵⁶!

Poderia se criptografar a seguinte frase: 55,30,34,42,13,20,45
Que seria traduzida por "SOCORRO". Note-se que letras repetidas, vêm escritas com números diferentes, já que existem diversos locais no texto original onde a mesma letra pode ser pinçada. Este fato inviabiliza o ataque por frequência de letras.

A cifra ADFGVX

A cifra ADFGVX foi introduzida no dia 5 de março de 1918, um pouco antes da ofensiva alemã de 21 de março. Tal cifra fora selecionada por um comitê de criptógrafos alemães especialmente para dar garantia a este avanço. Nessa época a artilharia alemã estava a menos de 100Km de Paris e se preparava para o assalto final. A esperança dos franceses era descobrir o ponto de suas defesas que os alemães pretendiam quebrar. Felizmente (para eles) havia uma arma secreta: um tenente francês, magro, de pele bronzeada e de mente penetrante chamado Georges Painvin. Este só descobrira seu talento para a criptoanálise, após um encontro casual com um membro do *Bureau du Chiffre*, logo após o início da guerra. Seu esforço contra a cifra ADFGVX o fez perder 15 kilos de peso. Na noite do dia 2 de junho de 1918, ele conseguiu decifrar uma mensagem em ADFGVX. Essa mensagem o levou a decifrar outra que dizia *Corram com as munições, mesmo durante o dia, se não forem vistos*. O início da mensagem indicava que fora enviada de um local entre Montdidier e Compiègne, cerca de 80Km ao norte de Paris. A necessidade urgente de munição sugeriu que ali se daria o ataque. A inteligência confirmou uma movimentação suspeita. Daí, soldados franceses foram enviados para reforçar a defesa. Uma semana depois começou a ofensiva. Tendo perdido o elemento surpresa, após 5 dias de batalhas infernais, o exército alemão foi obrigado a recuar.

A descrição do código ADFGVX

É uma mescla entre substituição e transposição. A cifragem começa com o desenho de uma grade 6 × 6, contendo 36 células. Nelas se escrevem aleatoriamente as 26 letras e os 10 dígitos. Cada uma das linhas e das colunas

assim produzidas é identificada pelas letras **ADFGVX**. Naturalmente, esta distribuição já faz parte da chave da mensagem. Suponhamos a grade

	A	D	F	G	V	X
A	u	n	i	c	e	p
D	f	d	y	2	s	m
F	o	0	a	q	x	j
G	k	8	7	v	t	4
V	9	b	6	h	5	z
X	3	l	1	r	w	g

o primeiro estágio, pega-se cada letra da mensagem e esta é substituída pelas letras que rotulam a linha e a coluna. Assim, "a" será substituída por "FF", "3" seria substituído por "XA" e assim por diante.

Seja como exemplo a mensagem "amanhã, prova de topicos". Ficaria

mensagem: amanha prova de topicos
original: a m a n h a p r o v a d e t o p i c o s
estagio 1: FF DX FF AD VG FF AX XG FA GG FF DD AV GV FA AX AF AG FA DV

No segundo estágio entra a figura de uma chave, na forma de uma palavra qualquer, por exemplo **HORRIVEL**. Primeiro, os caracteres repetidos são retirados, ficando: **HORIVEL**. A seguir, as letras do estágio 1 são escritas em uma matriz cujas colunas são as letras da chave, assim:

H	O	R	I	V	E	L
F	F	D	X	F	F	A
D	V	G	F	F	A	X
X	G	F	A	G	G	F
F	D	D	A	V	G	V
F	A	A	X	A	F	A
G	F	A	D	V		

No terceiro estágio, a chave (e as colunas da matriz) são colocadas em ordem alfabética:

E	H	I	L	O	R	V
F	F	X	A	F	D	F
A	D	F	X	V	G	F
G	X	A	F	G	F	G
G	F	A	V	D	D	V
F	F	X	A	A	A	A
	G	D		F	A	V

Finalmente, a mensagem é aquela formada pelo corpo da tabela acima, a saber:

FF XA FD FA DF XV GF GX AF GF GG FA VD DV FF XA AA AG DF AV

Algumas observações

- A mensagem precisa estar alinhada em um tamanho múltiplo ao da chave. Não pode ocorrer *padding* de um determinado caractere (zeros ?), pois isto geraria um padrão, e um padrão é tudo o que um criptógrafo não quer. Assim, neste exercício, ao final da mensagem, agregam-se tantos caracteres aleatórios quantos necessários.
- Em particular, para facilitar, aqui se usaram apenas números para este preenchimento. Na prática, deveriam ser os 36 caracteres possíveis.
- Cabe ao emissor, encerrar a mensagem a fim de que o preenchimento final não induza o leitor a erro.
- As letras A,D,F,G,V e X foram escolhidas por terem padrões bem distintos no código Morse. Na verdade, podiam ter sido escolhidas quaisquer 6 letras.

Para você fazer

Decodifique a mensagem

VA AV FF XA VA AG AF AG FX AF XV FA VV XG DX VD GV
FX FV AA VV XA AG XF DA FV DF VA AA DF XG FV VX FV
FX

Usando a chave **LANCANDOAS** e a tabela de codificação

	A	D	F	G	V	X
A	X	9	M	T	6	G
D	Z	U	5	F	V	N
F	S	Y	I	0	D	8
G	H	W	2	Q	3	R
V	E	J	K	4	O	7
X	A	1	L	P	C	B

1



Teste de Primalidade

Quero saber se p é ou não é primo. Então:

1. Usando a teoria da matemática básica => complexidade $O(n)$
1: funcao primo(x)
2: d ← 2
3: numero ← 1
4: enquanto (d < ⌈ sqrt(x) ⌋) ∧ numero=1
5: se x mod d = 0
6: numero ← 0
7: fim{se}
8: d ← d + 1
9: fim{enquanto}
10: retorne numero
11: fimfuncao
2. Usando o pequeno teorema de Fermat => complexidade $O(\log n)$
1: funcao primofermat (p)
2: a ← aleatorio(p)
3: enquanto mdc (p,a) ≠ 1
4: a ← aleatorio(p)
5: fim{enquanto}
6: se $1=(a^{p-1}) \bmod p$
7: retorne 1 {alguma chance de erro}
8: senão
9: retorne 0
10: fim{se}
11: fimfuncao

Em resumo, estes são os passos:

- (a) Chuto um número menor do que p e que seja coprimo com p (a e b são coprimos se o mdc(a,b) = 1)
- (b) Calculo $x = [a^{(p-1)}] \bmod p$, usando o algoritmo addition chaining (visto em aulas passadas) ou similar
- (c) Se $x=1$ provavelmente* é primo **
- (d) Se $x\neq 1$ é certamente composto
- (e) * Esta probabilidade pode ser abaixada, rodando várias vezes este algoritmo, com diversos "a"
- (f) ** A menos dos números de Carmichael

- 1: função mdc (a,b)
2: se b=0
3: retorne a
4: senão
5: retorne mdc (b, a mod b)
6: fim{se}
7: fimfuncao

Criptografia El Gamal

Vimos que a segurança do método RSA reside na dificuldade (esperada, mas ainda não provada) de fatorar números grandes. Outros problemas difíceis de resolver também podem dar origem a outros métodos de criptografia. Um desses métodos, que recebeu o nome de seu descobridor (Taher El Gamal), está na dificuldade esperada para resolver o problema conhecido como *logaritmo discreto*. Quando se trabalha com números reais, o $\log_b a$ é o número x para o qual a é igual a b^x . Lembrando através de uma fórmula:

$\log_a b = c \Rightarrow a^c = b$

Usando-se a aritmética dos ponteiros do relógio, também é possível definir um logaritmo (agora chamado *discreto*). Note que para este tipo de problema só intervem números inteiros, como convém à aritmética discreta. Dados os inteiro b e n com $b < n$, o logaritmo discreto de um inteiro a na base b é o inteiro x tal que $b^x \equiv a \bmod n$ usando uma fórmula:

$\uparrow \log_{b,n} a = x \Rightarrow b^x \equiv a \bmod n$

Por falta de símbolo ou até nome para este logaritmo, ele costuma ser chamado de *índice* e escreve-se

$x = \text{ind}_{b,n} a$

El Gamal

O esquema El Gamal foi originalmente proposto para implementar assinaturas digitais, mas ele também pode ser usado para encriptar e desencriptar mensagens. O El Gamal tem uma vantagem (dois de seus parâmetros públicos podem ser compartilhados por grupos de usuários) e uma desvantagem (a mensagem cifrada tem o dobro de tamanho do que a mensagem plena). A criação de chaves começa pela escolha de um primo p (possivelmente bem grande, com mais de 100 dígitos decimais). Depois escolhem-se outros dois aleatórios g e x , ambos menores do que p . A seguir, calcula-se

$y = g^x \bmod p$

A chave pública é y, g, p e a chave privada é x . Havendo um grupo de usuários, estes podem compartilhar o par g, p . Cada um terá apenas que escolher um x próprio e a seguir gerar o seu y .

Encriptando

Suponha-se o desejo de encriptar o valor m . O remetente começa escolhendo um aleatório *ad hoc* (ou seja: nunca deve ser reaproveitado) k , tal que $1 \leq k \leq (p - 2)$ e ainda tal que k e $p - 1$ sejam primos entre si, ou seja que estes dois números não tenham submúltiplo comum exceto a unidade. Finalmente, o texto cifrado é composto pelo par $[a, b]$, que por sua vez é

$a = g^k \bmod p$

$b = (m \times y^k) \bmod p$

O cálculo de b fica facilitado já que

$b = (m \times y^k) \bmod p = [m \times (y^k \bmod p)] \bmod p$

Desencriptando

O destinatário, ao receber o par $[a, b]$, calcula

$\xi = a^{p-(1+x)} \bmod p$

$m = (b \times \xi) \bmod p$

Exemplo

Vamos criptografar a palavra "CABELO"usando o esquema El Gamal e enviar a mensagem para um destinatário qualquer. Primeiro, precisamos transformar o texto em números. Vai-se usar aqui uma tabela bem simples:

	1	G	8	N	15	U	22
A	2	H	9	O	16	V	23
B	3	I	10	P	17	W	24
C	4	J	11	Q	18	X	25
D	5	K	12	R	19	Y	26
E	6	L	13	S	20	Z	27
F	7	M	14	T	21		

Com isso, a mensagem a criptografar passa a ser 4,2,3,6,13,16. O destinatário, sabendo que deveria receber mensagens criptografadas, tratou de gerar suas chaves. Primeiro, gerou um primo (inferior a 10.000, que ninguém está muito a fim de fazer contas muito grandes). Obteve o número $p = 9293$. A seguir, gerou dois números, ambos inferiores a p . Foram eles $g = 8579$ e $x = 8656$. O próximo passo foi calcular $y = g^x \bmod p = 8579^{8656} \bmod 9293$. O resultado encontrado foi $y = 1625$. Com isto, a chave pública do destinatário, que ele está publicando é $[y, g, p]$ ou seja, neste caso $[1625, 8579, 9293]$. A chave secreta do destinatário, que só ele sabe é $[8656]$. Para enviar a mensagem CABELO (4,2,3,6,13,16) para ele, precisamos construir o texto cifrado. Cada um dos números dará origem a outros 2 números (não esquecer que no método El Gamal as mensagens do-
bram de tamanho). Usando as fórmulas

$a = g^k \bmod p$

$b = [m \times (y^k \bmod p)] \bmod p$

tem-se

	C	A	B	E	L	O
a	8939	4555	899	6507	6685	2585
b	9151	8355	5218	1901	4232	4465
k	7069	8243	5435	7441	8321	6431

Note que o que é enviado é apenas o par $[a, b]$ para cada caracter a enviar. A constante k não é enviada. Ela só aparece aqui caso se queira refazer os cálculos envolvidos. Do lado do receptor, como vimos, chega o par $[a, b]$. Agora, usando as fórmulas

$\xi = a^{p-(1+x)} \bmod p$

$m = (b \times \xi) \bmod p$

o destinatário consegue recuperar a mensagem "CABELO".

Algoritmo de $x^y \bmod n$

Calcula $x^y \bmod n$

- 1: inteirolongo função ADCHA(inteirolongo x, y, n)
2: se (y = 1)
3: devolva x mod n
4: senão
5: se y é par
6: AUX ← ADCHA(x, (y ÷ 2), n)
7: devolva (AUX × AUX) mod n
8: senão
9: AUX ← ADCHA(x, ((y - 1) ÷ 2), n)
10: AUX ← (AUX × AUX) mod n
11: devolva (AUX × x) mod n
12: fim{se}
13: fim{se}
- Para testar o algoritmo: $3^4 \bmod 5 = 1$
 $2^3 \bmod 4 = 0$
 $2^3 \bmod 30 = 8$
 $1127^{1135} \bmod 7641 = 1964$
 $6901^{7541} \bmod 11005 = 9031$

Para você fazer

Suponha que você quer receber mensagens cifradas. Sua chave pública é $[y, g, p]$ é $[496, 8183, 9413]$, e sua chave privada x é $[6634]$. Você recebeu a seguinte mensagem, composta por 16 caracteres. Eis a primeira parte:

- a= 1549 5014 1198 4407 6295 4621 7893 3687
b= 3350 8430 1844 4218 4218 5222 809 6561

E a segunda parte

- a= 6295 3335 699 7821 3172 1759 8620 398
b= 6482 246 9159 1156 6050 5235 7758 6154

Descriptografe e informe o diz a mensagem:

1



- 1 - /

Esteganografia

Em países onde a censura é abundante, os dissidentes tendem a usar a tecnologia para burlar sua rigidez. A criptografia permite que mensagens secretas sejam enviadas mas se o governo está de olho sobre os dissidentes (os "de sempre", segundo o chefe de polícia de Getúlio Vargas, ou o ator Claude Rains que faz o chefe de polícia francês em Casablanca), o simples fato de enviar uma mensagem secreta já pode ser visto como delito reprimível.

A idéia é comunicar mensagens que não sejam entendidas como tal. Usa-se a esteganografia, das palavras gregas que significam "escrita cifrada". Heródoto escreveu sobre um general que raspou a cabeça de um escravo, tatuou lá uma mensagem, esperou o cabelo crescer e o enviou atravessando as linhas inimigas, que não perceberam a passagem do mensageiro. As técnicas modernas são conceitualmente as mesmas, só com uma largura de banda mais alta e com latência mais baixa.

A seguir duas mensagens inocentes de um gato:



A segunda imagem contém além da imagem do gato, a íntegra do Hino Nacional Brasileiro com um total de 3060 caracteres.

O truque está em usar uma imagem BMP em true-color, que como vimos admite 16.777.216 (2²⁴) cores. Se usurpamos um único bit de cada byte, haverá uma diminuição nas cores, mas garantidamente nenhum olho humano será capaz de percebê-la. A idéia é deixar apenas 7 bits para codificar a cor e usar o bit menos significativo de cada byte para codificar a mensagem. A quantidade de cores agora é de apenas 2.097.152 (2²¹), mas como está se usando o bit menos importante, a diferença é imperceptível.

Não fiz isto neste exercício, mas a boa prática manda que a mensagem seja enviada criptografada, quando então um degrau significativo na segurança da mensagem será adicionado.

Outra observação bem importante é que esta técnica só se aplica a imagens NOVAS. Se o autor pegar uma imagem manjada na Internet e codificar nela uma mensagem, qualquer desconfiança poderá ser confirmada simplesmente rodando um FILE COMPARE (utilitário FC) sobre a imagem enviada e a que existe por aí na Internet. O FC vai apontar diferenças de conteúdo, e vai escancarar o uso da esteganografia.

Finalmente, para encerrar o nosso curso de resistência às ditaduras, segue a recomendação de criar um site dedicado ao ditador cheio de imagens laudatórias dele, de suas pretensas conquistas e essas coisas. Cada imagem dessa pode carregar mensagens secretas e tão virulentas (do ponto de vista do ditador) quanto se queira.

Padrão BMP O bloco de controle tem a seguinte configuração

Tipo	Tam	Desl	Nome	Descrição	exemplo
char	2	0	type	constante igual a BM	424D
long int	4	2	size	tamanho do arquivo	00 01 45 16
int	2	6	reserved	zeros	00 00
int	2	8	reserved	zeros	00 00
long int	4	A	off	desloc até a imagem	00 00 00 36
long int	4	E	size	resto do bloco controle	28 00 00 00
long int	4	12	width	n. de colunas	00 00 00 96
long int	4	16	height	n. de linhas	00 00 00 B8
int	2	1A	planes	planos	00 01
int	2	1C	bitcount	bits / pixel	00 18
long int	4	1E	compress	compressao	00 00 00 00
long int	4	22	sizeimage	tamanho da imagem	00 01 44 E0
long int	4	26	pix/m H	pixels / m na horizontal	00 00 00 00
long int	4	2A	pix/m V	na vertical	00 00 00 00

Note que

- ★ Neste arquivo não há tabela de cores; ele é true color, já que o campo *bitcount* vale 18 (24 em decimal).
- ★ Portanto, no endereço 36 está a cor azul do primeiro pixel da imagem.
- ★ Cada linha, no seu final pode ter *stuffing bits* de maneira a alinhar a próxima linha em múltiplo de 4 bytes.
- ★ Se um determinado pixel tiver R = G = B, pode-se concluir que este pixel é monocromático, ou como se diz “branco e preto”.
- ★ a ordem das cores no arquivo é BGR ao contrário do nome do esquema que é RGB.

Como fazer A seguir o começo do primeiro arquivo do gato (sem nada)

ende	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ascii
0000	42	4D	16	45	01	00	00	00	00	00	36	00	00	00	28	00	BM.E.....6...(.0010
0010	00	00	96	00	00	00	B8	00	00	00	01	00	18	00	00	00	D.....0020
0020	00	00	E0	44	01	00	00	00	00	00	00	00	00	00	00	00	...D.....0030
0030	00	00	00	00	00	00	B6	CD	C5	B7	CE	C6	B8	CF	CB	B7	c~.A_d4Q0040
0040	D0	CC	AD	C5	C5	8D	A9	AA	63	7E	82	41	5F	64	34	51	A_d4Q0050
0050	5A	26	45	4E	20	3E	49	2B	49	54	3C	5A	65	41	5F	6A	Z&EN >I+IT<ZeA_j0060
0060	31	4F	5A	20	3E	49	2E	4C	57	3D	5B	66	44	60	6B	3B	10Z >I.LW=[fD`k;0070
0070	57	62	3B	57	62	4F	6B	76	68	83	91	74	8F	9D	4B	66	Wb;Wb0kvh...t..Kf0080
0080	74	51	6C	7A	4E	6B	79	4C	69	77	4C	69	77	47	64	72	tQlZnKyLiwLiwGdr0090
0090	49	69	76	57	79	86	94	B6	C3	69	8C	99	84	A9	B7	66	IivWy....i....f00A0
00A0	8B	99	69	90	9E	87	B1	BE	5B	84	93	6A	95	A4	89	B6	..i.....[.j....00B0
00B0	C4	85	B4	C2	8A	B8	C9	87	B8	C8	70	A3	B3	5B	8E	9E	p..[.00C0
00C0	65	99	A9	80	B5	C2	9E	D5	E2	8C	C4	CF	B5	EA	F7	9E	e.....b..g.00D0
00D0	D3	E0	8A	BF	CC	A7	DC	E9	86	BD	CC	62	9B	AA	67	9F	b..g.00E0
00E0	B0	65	A0	B0	6C	A4	B7	72	A7	BB	67	99	AF	55	84	9A	.e..l..r..g..U..00F0
00F0	50	7C	93	57	83	9A	66	92	A9	65	94	AA	72	9C	B3	6A	Pl .W .f..e..r..j0100
0100	94	AB	57	7E	94	57	7E	94	5E	86	99	56	7E	91	4F	7A	..W~.W~.~.V~.0z0110
0110	8B	5E	89	9A	47	72	83	51	7C	8D	58	81	90	44	6D	7C	^..Gr.Q .X..Dm ...

Agora, o começo do segundo arquivo (com o hino codificado)

ende	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ascii
0000	42	4D	16	45	01	00	00	00	00	00	36	00	00	00	28	00	BM.E.....6...(.0010
0010	00	00	96	00	00	00	B8	00	00	00	01	00	18	00	00	00	D.....0020
0020	00	00	E0	44	01	00	00	00	00	00	00	00	00	00	00	00	...D.....0030
0030	00	00	00	00	00	00	B6	CC	C4	B6	CF	C6	B9	CF	CB	B7	b~.A_e4Q0040
0040	D1	CD	AC	C5	C4	8C	A8	AB	62	7E	83	41	5F	65	34	51	b~.A_e4Q0050
0050	5B	27	44	4F	20	3F	48	2B	49	55	3C	5B	65	40	5E	6B	['DO ?H+IU<[e@`k0060
0060	31	4E	5B	20	3E	49	2E	4D	57	3D	5A	66	45	60	6A	3B	1N[>I.MW=ZfE`j;0070
0070	57	62	3A	56	62	4F	6A	77	69	82	91	75	8E	9D	4A	66	Wb;Vb0jwi..u..Jf0080
0080	75	50	6C	7A	4E	6A	78	4D	69	76	4C	69	76	46	64	73	uPlZnJxMivLivFds0090
0090	49	68	77	57	79	87	94	B6	C3	68	8C	98	84	A8	B6	67	IhwWy....h....g00A0
00A0	8A	98	69	90	9E	87	B0	BF	5B	85	92	6A	94	A4	88	B7	..i.....[.j....00B0
00B0	C5	84	B5	C2	8A	B9	C8	87	B9	C9	70	A2	B3	5A	8E	9F	p..Z..00C0
00C0	65	98	A8	80	B4	C3	9E	D5	E3	8C	C5	CF	B5	EA	F6	9F	e.....b..f.00D0
00D0	D3	E0	8A	BF	CD	A7	DC	E9	87	BC	CC	62	9A	AB	66	9E	b..f.00E0
00E0	B1	64	A0	B0	6C	A4	B6	73	A7	BA	66	98	AE	55	84	9B	.d..l..s..f..U..00F0
00F0	51	7D	92	56	83	9B	66	92	A9	64	94	AA	72	9C	B2	6B	Q .V..f..d..r..k0100
0100	95	AA	57	7F	94	57	7E	95	5F	86	98	56	7E	91	4E	7B	..W~.W~.~.V~.N{0110
0110	8B	5F	88	9A	47	72	82	51	7D	8C	58	81	91	45	6C	7D	^..Gr.Q .X..E }

Nele está codificada o Hino Nacional Brasileiro. Nos bytes 55, 56, ..., 70 (em decimal) ou 36, 37, 38, 39, 3A, 3B, 3C, 3D, 3E, 3F, 40, 41, 42, 43, 44 e 45 está codificado um número binário contendo o comprimento da mensagem, neste caso 3060. Que começa no byte 71 (decimal) ou 46 (hexadecimal).

A solução do problema pode ser obtida de duas maneiras:

- ★ Escrevendo um programa que leia a imagem e recupere os bits montando conjuntos de 8 em 8 bits e consultando uma tabela ASCII ou
- ★ Olhando o arquivo através de um utilitário binário (por exemplo wxHexEditor ou então o site <http://www.onlinehexeditor.com/>) e investigando os bytes em questão. Se o byte tiver conteúdo PAR, é porque foi codificado um ZERO nele. Se o conteúdo for ÍMPAR, codificou-se um UM nele. Deve-se recuperar os bits de 8 em 8 e depois consultar manualmente uma tabela ASCII.

Para você fazer

Você recebeu um arquivo de nome

XM01.BMP

Nele está codificada uma pequena mensagem. Os dois primeiros bytes da mensagem codificam seu tamanho, como visto acima.

A mensagem achada é:



31-2.1.3.	Impostos e Taxas	.00	720.72
32-2.1.3.1.	IPTU	.00	317.56
34-2.1.3.3.	ICMS	.00	403.16
35-2.2.	Não Exigível	.00	191.39
36-2.2.1.	Capital	.00	191.39

As perguntas a responder são duas:

1. Qual o valor da conta “Exigível” ?
2. Qual o saldo deste balancete ?

As respostas a estas perguntas são 1899.72 e (879.67)

Para você fazer

Plano de contas

1-1.	ATIVO
2-1.1.	Disponível
3-1.1.1.	Caixa
4-1.1.2.	Bancos
7-1.1.2.3.	CEF
8-1.2.	Imobilizado
9-1.2.1.	Imóveis
11-1.2.3.	Móveis e Utensílios
12-1.3.	Realizável
15-1.3.3.	Contas a Receber
16-1.3.4.	Créditos em aplicações financeiras
17-2.	PASSIVO
18-2.1.	Exigível
19-2.1.1.	Contas a Pagar
22-2.1.1.3.	Embalagens
23-2.1.1.4.	Despesas Administrativas
24-2.1.1.4.1.	Impressos e Material Escritório
25-2.1.1.4.2.	Café
26-2.1.1.4.3.	Material de limpeza
27-2.1.2.	Fornecedores
28-2.1.2.1.	Fornecedor A
29-2.1.2.2.	Fornecedor B
30-2.1.2.3.	Fornecedor C
31-2.1.3.	Impostos e Taxas
32-2.1.3.1.	IPTU
33-2.1.3.2.	IR
35-2.2.	Não Exigível
36-2.2.1.	Capital

E o seguinte arquivo de lançamentos

01 28 T	175.89	0907	100912
02 36 M	109.86	0807	101115
03 32 U	111.44	0810	100917
04 28 B	58.00	1008	100706
05 28 V	187.90	1012	080726
06 22 I	94.00	1009	091103
07 24 D	175.59	0812	091210
08 11 H	49.00	1007	100722
09 33 K	10.00	1007	091223
10 22 G	161.08	1012	081213
11 33 L	34.00	1007	090910
12 36 O	186.59	0809	091102
13 29 U	17.00	0810	090816
14 15 F	157.79	0911	101206
15 29 P	118.78	1009	091005
16 25 O	136.05	0808	081106
17 07 G	35.00	1010	091227
18 26 N	1.00	1008	081017
19 15 J	145.31	1007	101004
20 32 J	33.00	0808	081013
21 33 I	184.72	1009	091223
22 07 T	6.00	1011	101017
23 33 X	148.41	1010	080817
24 36 K	66.00	0809	101224
25 33 C	187.18	0809	101024
26 15 L	7.00	0910	100926
27 22 F	178.85	0911	100801
28 26 H	104.93	0911	080904
29 07 D	4.00	0911	100903
30 25 Q	134.92	1007	080702

Pergunta-se qual o valor da conta “Exigível” e qual o saldo deste balancete ?



- 1 - /

Exemplo

Seja o seguinte Plano de Contas:

1-1.	ATIVO
2-1.1.	Disponível
3-1.1.1.	Caixa
4-1.1.2.	Bancos
6-1.1.2.2.	Banco do Brasil
7-1.1.2.3.	CEF
8-1.2.	Imobilizado
10-1.2.2.	Veículos
12-1.3.	Realizável
13-1.3.1.	Estoque de produtos
14-1.3.2.	Estoque de matérias primas
16-1.3.4.	Créditos em aplicações financeiras
17-2.	PASSIVO
18-2.1.	Exigível
19-2.1.1.	Contas a Pagar
20-2.1.1.1.	Matéria Prima
23-2.1.1.4.	Despesas Administrativas
24-2.1.1.4.1.	Impressos e Material Escritório
25-2.1.1.4.2.	Café
27-2.1.2.	Fornecedores
28-2.1.2.1.	Fornecedor A
30-2.1.2.3.	Fornecedor C
31-2.1.3.	Impostos e Taxas
32-2.1.3.1.	IPTU
34-2.1.3.3.	ICMS
35-2.2.	Não Exigível
36-2.2.1.	Capital
38-2.2.3.	Provisões

E o seguinte arquivo de lançamentos

01 24 E	175.17	0909	081222
02 10 I	125.09	0910	101121
03 14 A	24.00	1007	091203
04 34 O	25.00	0911	091117
05 13 B	70.00	0908	101211
06 30 Z	122.25	1010	080813
07 28 L	143.96	1009	100912
08 07 P	89.00	1009	080908
09 30 F	196.34	1007	081102
10 30 C	181.94	1010	100904
11 28 R	32.00	0809	081210
12 36 W	103.39	0812	090927
13 32 M	199.44	0907	091015
14 32 V	118.12	0909	090922
15 16 D	150.68	0909	080926
16 16 Y	144.07	1009	091012
17 13 T	94.00	0812	090708
18 07 H	46.00	1011	100820
19 06 G	71.00	1010	091018
20 30 J	179.87	0807	100725
21 36 X	88.00	1008	081212
22 24 K	105.47	0812	100717
23 34 S	128.11	1009	101221
24 06 K	66.00	1007	080912
25 25 B	42.00	1008	090704
26 34 V	107.36	0911	081218
27 34 Q	142.69	0807	091101
28 16 H	59.00	0809	080807
29 10 G	194.60	0808	090717
30 10 L	78.00	0912	091211

Este processamento deverá gerar o seguinte balancete

1-1.	ATIVO	1211.44	.00
2-1.1.	Disponível	272.00	.00
4-1.1.2.	Bancos	272.00	.00
6-1.1.2.2.	Banco do Brasil	137.00	.00
7-1.1.2.3.	CEF	135.00	.00
8-1.2.	Imobilizado	397.69	.00
10-1.2.2.	Veículos	397.69	.00
12-1.3.	Realizável	541.75	.00
13-1.3.1.	Estoque de produtos	164.00	.00
14-1.3.2.	Estoque de matérias primas	24.00	.00
16-1.3.4.	Créditos em aplicações financeiras	353.75	.00
17-2.	PASSIVO	.00	2091.11
18-2.1.	Exigível	.00	1899.72
19-2.1.1.	Contas a Pagar	.00	322.64
23-2.1.1.4.	Despesas Administrativas	.00	322.64
24-2.1.1.4.1.	Impressos e Material Escritório	.00	280.64
25-2.1.1.4.2.	Café	.00	42.00
27-2.1.2.	Fornecedores	.00	856.36
28-2.1.2.1.	Fornecedor A	.00	175.96
30-2.1.2.3.	Fornecedor C	.00	680.40

Contabilidade

Há sempre pelo menos duas maneiras de enxergar a contabilidade: como coisa de contadores e do fisco e como ferramenta de gestão da organização. Esta visão reflete a importância que a organização atribui à contabilidade e a maneira como ela ocorre.

Mantida atualizada, a contabilidade pode informar em tempo real a quantas anda seu lucro, quanto vai gastar de impostos, qual a matéria prima e inúmeras outras questões fundamentais sobre a vida financeira da empresa.

Descrição do aplicativo

Um conjunto de arquivos e programas que registram a vida financeira da companhia, servindo como registro legal junto às autoridades: Junta Comercial, Fisco e Acionistas (se companhia aberta). Toda a transação efetuada gera um ou mais lançamentos contábeis. Os balanços são as principais saídas do sistema.

Principais idéias apresentadas

Processamento Legal A novidade é gerar documentos, relatórios e arquivos que são entregues à autoridade. E, por causa disto, são por ela regulados e padronizados.

Congelamento de arquivos Como consequência da regulação, os arquivos ao final do ciclo são congelados e não podem mais ser modificados.

Ambiente onde se insere

Qualquer empresa, salvo as incluídas em esquemas simplificados de tributação, precisam ter e manter suas contabilidades.

Principais funcionalidades

Balancetes diários, mensais e anuais. Relatórios submetidos ao Fisco.

Entidades e atributos principais

Plano de contas O plano de contas é a estrutura sobre a qual a contabilidade ocorre. Tudo começa com ele. Ele descreve quais contas vão ser controladas e como vão ser somadas e subtraídas a fim de mostrar os principais números da organização.

Eis o início de um plano de contas bem simplificado:

- 1. ATIVO
 - 1.1. Disponível
 - 1.1.1. Caixa
 - 1.1.2. Bancos
 - 1.1.2.1. Itaú
 - 1.1.2.2. Banco do Brasil
 - 1.1.2.3. ...
 - 1.2. Imobilizado
 - 1.2.1. Imóveis
 - 1.2.2. Veículos
 - 1.2.3. Móveis e Utensílios
 - 1.3. Realizável
 - 1.3.1. Estoque de produtos
 - 1.3.2. Estoque de matérias primas
 - 1.3.3. Contas a Receber
 - 1.3.4. Créditos em aplicações financeiras
 - 2. PASSIVO
 - 2.1. Exigível
 - 2.1.1. Contas a Pagar
 - 2.1.1.1. Matéria Prima
 - 2.1.1.2. Salários Produção
 - 2.1.1.3. Embalagens
 - 2.1.1.4. Despesas Administrativas
 - 2.1.1.4.1. Impressos e Material Escritório
 - 2.1.2. Fornecedores
 - 2.1.3. Impostos e Taxas
 - 2.2. Não Exigível
 - 2.2.1. Capital
 - 2.2.2. Reservas
 - 2.2.3. Provisões

Esta é apenas uma parte de um plano de contas. Elas podem ainda se subdividir quanto for necessário (é comum, planos de grandes empresas terem até 10 níveis). Com os lançamentos apropriados e desde que tudo esteja atualizado, pode-se conhecer qualquer conta desde as detalhadas até as acumuladas no nível desejado.

A estrutura de um arquivo simples para conter o plano de contas pode

ser:

CAMPO	TIPO	TAM	OBSERVAÇÃO
número-conta	N	6	identifica os registros para permitir a manutenção
codigo-conta	C	12	Pode ser 6 × 2, com 6 níveis e 2 caracteres por nível
descrição	C	30	tamanho a gosto
mini-código	N	4	atalho para o código-conta
debito	N	20,2	20 casas com 2 decimais
credito	N	20,2	20 casas com 2 decimais
saldo	N	20,2	20 casas com 2 decimais

Lançamentos Cada transação efetuada (tanto de crédito como de débito) gera um lançamento. O arquivo tem a seguinte estrutura

CAMPO	TIPO	TAM	OBSERVAÇÃO
número-lançamento	N	6	identifica os registros para permitir a manutenção
mini-código	N	4	atalho para o código-conta
descrição	C	30	tamanho a gosto
valor	N	20,2	20 casas com 2 decimais
data-referência	D		
data-pagamento	D		

Balancete Finalmente, o terceiro arquivo envolvido é o do balancete, que vem a ser a consolidação dos lançamentos em um determinado período. Este arquivo, uma vez processado é congelado e não pode mais sofrer modificações. Parte importante deste arquivo é o transporte para trás e para frente.

CAMPO	TIPO	TAM	OBSERVAÇÃO
periodo-referência	C	6	período do balancete
código-conta	C	12	código-conta de acordo com o plano de contas
data	D		
descrição	C	30	tamanho a gosto
valor	N	20,2	20 casas com 2 decimais
<i>Débito</i>	N	22,2	valor
<i>Crédito</i>	N	22,2	valor
<i>Saldo</i>	N	22,2	valor

Conjunto de ciclos, processos e/ou casos de uso, algoritmos

As contas são alimentadas em lotes diários, semanais ou mensais. Os lotes são processados em busca de erros de consistência e de compatibilidade e à medida em que eles vão sendo aprovados, passam a integrar o arquivo de lançamentos.

Quando todos os lotes são aprovados, eles são integrados em um único arquivo de lançamentos e uma vez este aprovado, é executado o ciclo mensal da contabilidade. Considerando que esta atividade atende a uma necessidade fiscal (e legal), depois de processada a contabilidade mensal, o arquivo de lançamentos (e o balancete mensal, daí resultante) são congelados, isto é, não podem mais ser modificados.

Após o processamento bem sucedido do ciclo mensal da contabilidade, os valores das contas são transferidos de mês, e o ciclo recomeça no novo mês.

Finalmente, no final do ano este procedimento se repete e são encerrados os ciclos anuais envolvidos.

Sistemas correlatos

Contas a pagar Lista das dívidas da empresa. Contém informações sobre cada uma delas, bem como as políticas implementadas para atraso ou adiantamento. Permite consolidações e previsões.

Contas a receber Praticamente a mesma coisa de cima, mas em sentido inverso

Dívida ativa ou Cobrança Judicial Para este sistema vem as contas que não foram pagas em tempo hábil

Ativo Fixo Lista dos bens da empresa, com suas políticas de depreciação

Investimentos Bens e investimentos em geral

Folha de Pagamento

A folha de pagamento é um documento de emissão obrigatória para efeito de fiscalização trabalhista e previdenciária. A empresa é obrigada a preparar a folha de pagamento da remuneração paga, devida ou creditada a todos os



empregados a seu serviço.

Para sua elaboração não existe modelo oficial, ou seja, podem ser adotados critérios que melhor atendam as necessidades de cada empresa. Uma folha de pagamento, por mais simples que seja, apresenta pelo menos os seguintes elementos:

- Discriminação do nome dos empregados (segurados), indicando cargo, função ou serviço prestado;
- Valor bruto dos salários;
- Valor da contribuição de Previdência, descontado dos salários;
- Valor líquido que os empregados receberão.

Da folha de pagamento origina-se o recibo de pagamento, também conhecido como Hollerith (cuja origem vem de Hermann Hollerith) que indica os dados que constaram da folha relativamente a cada um dos empregados e a estes é entregue.

Valor bruto dos salários É o valor, considerado, para a empresa como despesa total de salários, além dos salários é também despesa para a empresa a contribuição de Previdência parte empresa e o FGTS (Fundo de Garantia do Tempo de Serviço)

Previdência Social - INSS De acordo com a legislação atual, todo empregado assalariado, regido pela Consolidação das Leis do Trabalho (CLT), está obrigado a contribuir com a Previdência Social. Essa contribuição é descontada do empregado em folha de pagamento, ela varia de acordo com a faixa salarial de cada empregado e, é calculada mediante aplicação de um percentual sobre o salário de contribuição.¹

Atualmente, o cálculo é feito com base na seguinte tabela:

Tabela de Contribuição da Previdência Social	
Salário de Contribuição R\$	Alíquota para fins de Recolhimento ao INSS (%)
Até 1556.94	8
De 1556.95 até 2594.92	9
De 2594.93 até 5189.82	11

O valor limite para contribuição é de R\$ 570.88 (teto de contribuição).

A Contribuição de Previdência parte referente à empresa, corresponde a 26,8% sobre o valor bruto da folha de pagamento.

FGTS (Fundo de Garantia do Tempo de Serviço) Corresponde a 8% sobre o valor da folha de pagamento, o qual será recolhido na Caixa Econômica Federal, em nome dos empregados. Constitui uma despesa paga pela empresa aos empregados.

IRF (Imposto de Renda na Fonte) É um imposto variável de acordo com o valor do salário. Sobre o valor devido do salário aplica-se a tabela progressiva a seguir:

Base de cálculo mensal em R\$	Alíquota (%)	Parcela a deduzir do imposto em R\$
Até 1903.98	Isento	-
Acima de 1903.98 até 2826.65	7,5%	R\$142.80
De 2826.66 a 3751.05	15,0%	R\$354.80
De 3751.06 a 4664.68	22,5%	R\$636.13
Acima de R\$ 4664.68	27,5%	R\$869.36

Salário Família É um auxílio da Previdência Social aos trabalhadores de baixa renda que possuem filhos menores de 14 anos de idade. A empresa paga este valor para o empregado e é reembolsada pela Previdência Social quando efetua os recolhimentos correspondentes a folha de pagamento. Corresponde a um valor fixo por filho. Antigamente era um valor fixo por filho. Mas, nos últimos anos o Governo Brasileiro mudou a fórmula de cálculo do S.F. lançando duas parcelas: 41.37 (por filho) para os salários menores que 806.81 e de 29.16 (por filho) para os maiores que 806.80.

¹Salário contribuição é um valor fixado pela Previdência que serve de base para o cálculo das contribuições previdenciárias.

Processo de cálculo

Dados de entrada Simplificadamente, para o conjunto de funcionários deve ser dito:

Nome	Sal. Bruto	Atrasos	Núm. Filhos
João	1.000	-	1
Maria	2.000	-	2
Paulo	2.200	100	-

Com estes dados, deve-se calcular a folha, assim

Nome	Bruto	Atr	PDev	INSS	IR	SF	Liq	FGTS
N	B	A	P	I	R	F	L	G

N,B,A: dados de entrada

P: calcular: B - A

I: calcular conforme tabela a partir de P

R: conforme tabela a partir de P

F: dado de entrada (filhos) × constante

L: calcular: P - I - R + F

G: calcular: 8% de P

Além da folha é preciso calcular o RESUMO DA FOLHA

INSS empregados: somatória vertical da coluna I da folha

FGTS a recolher: somatória vertical da coluna G da folha

IR a recolher: somatória vertical da coluna R da folha

Salário Família a pagar: somatório vertical da coluna F da folha

Líquido a creditar aos empregados: somatório vertical da coluna L da folha

Valor bruto da folha: igual a soma da soma de duas colunas: P + G

INSS da empresa: corresponde a 26,8% do valor bruto da folha

INSS a recolher: o resultado de INSS da Empresa menos a soma da coluna F da folha

Despesa total da empresa: Soma do valor bruto da folha + INSS a recolher

Observação importante: se tiver alguma dúvida no entendimento das regras, observe o exemplo abaixo. Ele foi gerado pelo programa que vai corrigir a sua resposta, logo ele deve ser a fonte final de consulta para resolver dúvidas.

Para você fazer

Suponha a seguinte empresa

NOME	BRUTO	ATRASOS	FILHOS
jose	2761.95	80.60	0
andre	9480.77	.00	2
pedro	1300.00	.00	0
joaquim	1200.00	80.80	2
andre	6030.02	8.31	0
sueli	1100.00	.00	0
zelia	9080.26	.00	1
sueli	1400.00	.00	1
bianca	2109.72	.00	0
paulo	3738.42	.00	0

Calcule a Folha de Pagamento e informe:

1. Imposto de Renda a Recolher
2. Salário Família a Pagar
3. Despesa Total da Empresa

IRR	SFP	DTE



- 1 - /

Controle de Estoque

Este exercício trata genericamente do abastecimento de materiais. Estes podem ser de qualquer tipo (matéria prima, produtos acabados, insumos,...) e seu uso futuro pode ser

- alimentação de um processo produtivo
- vendas no atacado e varejo
- manutenção de máquinas e equipamentos
- abastecimento de itens de consumo

O controle de estoque provê um ponto de equilíbrio entre duas tendências opostas: grandes estoques (minimização dos custos diretos de aquisição e nunca parar por falta de estoque, melhores preços por quantidade) versus pequenos estoques (economia de capital investido no estoque, economia de espaço de armazenamento). Além disso, pense-se nos custos de obsolescência, deterioração, seguros, vigilância, etc. Fica talvez mais fácil entender isto se o mesmo fenômeno for levado para as compras de uma dona de casa.

Ao balancear estas duas visões opostas, um bom sistema de estoque sugere o que comprar, quando e quanto.

Determinação da curva ABC

Esta ferramenta, originalmente descrita pelo economista Wilfredo Pareto, sugere estudar o estoque e tratar diferentemente os poucos itens caros (classe A) dos muitos itens baratos (classe C). Este procedimento tem os seguintes passos:

1. Obter a listagem de todos os itens do estoque e seus preços unitários
2. Obter ou estimar os consumos mensais ou anuais de cada item
3. Multiplicar os 2 valores acima para cada item
4. Ordenar a lista em ordem decrescente pelo item anterior
5. Separar os pontos referentes a 80% e 90% (ou mais) do item anterior
6. Estão separadas as zonas A, B e C do estoque

Observação importante: Os valores 80% e 90% surgem aqui apenas como referência. Cada ramo industrial, e mesmo cada indústria costuma estabelecer os seus parâmetros que melhor individualizam o meio onde a indústria opera. Então é comum achar curvas ABC de 60% e 85%; de 50% e 75% ou na verdade, de quaisquer outros valores.

Política de estoque

Como se viu, há duas estratégias opostas para manter o estoque. O ponto ideal é um ponto de equilíbrio que traga uma solução de compromisso entre ambas. Tal ponto é determinado pela teoria dos estoques.

Antes, veja-se um exemplo: Pretende-se a aquisição mais barata de um produto cuja demanda anual é de 8.400 unidades e cujo preço unitário de compra é de R\$ 3,00. Estima-se que cada compra custe R\$50,00 e que o custo de armazenamento (segurança, espaço, perdas...) é de 30% ao ano sobre o valor médio do estoque.

Preencha-se a seguinte tabela:

modalidade	frequencia	lote	C _{armaz}	C _{compra}	C _{total}
anual	1	8.400	3.780	50	3.830
semestral	2	4.200	1.890	100	1.990
quatrimestral	3	2.800	1.263.45	150	1.410
trimestral	4	2.100	945	200	1.145
bimestral	6	1.400	630	300	930
a cada 45 dias	8	1.050	472.50	400	872,50
mensal	12	700	315	600	915
quinzenal	24	350	157.50	1.200	1.357,50
semanal	52	161	72.45	2.600	2.672,45

Note-se que na tabela acima não se considera o custo do material adquirido (8.400 ao ano), já que ele é o mesmo para todas as alternativas. O custo do armazenamento se obtém multiplicando a quantidade média do estoque (adquirido ÷ 2) pelo preço do item pela taxa diária (0.3 ÷ 360) pelo número de dias do estoque pela quantidade de compras no ano.

Assim, por exemplo, o custo da compra modalidade quadrimestral é

$$q_{econom} = \frac{1}{2} \times q \times b \times i \times t_1 \times n$$

, onde

q_{econom} tamanho do lote econômico

q quantidade de itens a adquirir no ano

b preço unitário de cada item

i taxa diária de armazenamento (0.3 ÷ 360)

t₁ período considerado, 120 dias

n Número de períodos ao ano, no caso 3

$$C_{armaz} = \frac{1}{2} \times 2800 \times 3 \times 0.00083 \times 120 \times 3 = 1260.00$$

Se se desenhar em um gráfico as diversas possibilidades ver-se-ão as diversas possibilidades e salta que a área abaixo da curva descreve os custos de armazenamento.

Existe uma fórmula analítica que descreve o lote de compra ótimo sem que o usuário tenha que fazer todas as composições. É ela

$$q_e = \sqrt{\frac{2 \times K \times D}{b \times P}}$$

, onde *q_e* é o lote econômico; *K* custo unitário da aquisição, *D* consumo; *b* é o custo unitário do item e *P* é a Taxa anual de armazenamento. No exemplo

$$q_e = \sqrt{\frac{2 \times 50 \times 8400}{3 \times 0.3}} = 965$$

e daí

$$n = \frac{D}{q_e} = \frac{8400}{965} = 8,7 \sim 42 \text{ dias}$$

Para você fazer

Suponha o seguinte estoque

produto	unid	preco	ultimos 3 consumos				estoque
condensador	um	29414.00	4	2	14	14	
prego 20mm	milhar	43.22	94	76	36	114	
grampos ind	milhar	283.00	677	996	201	1166	
parafina	kg	8.20	821	162	129	1158	
misturador	um	12001.00	6	2	7	5	
aco 12mm	m2	885.00	327	453	411	561	
aco 8mm	m2	337.00	504	502	501	156	
aramé 5mm	metro	.41	93	78	82	48	
cano cobre 10p	metro	70.95	417	945	749	226	
tinta preta	galao	35.00	8	14	9	9	
cal	ton	81.00	115	143	124	144	
papel A4-90mg	resma	2538.00	6	3	8	12	
cano cobre 1p	metro	18.12	290	560	304	1457	
ripa fina	m	263.00	25	30	38	38	
cano cobre 2p	metro	22.00	45	324	8	46	
querosene	litro	1.39	6508	1052	5415	3010	
prego 50mm	milhar	60.42	120	106	110	288	
gesso	ton	69.00	135	144	105	119	
aramé 1mm	metro	.40	6	81	46	95	
papel A4-75mg	resma	678.00	3	1	5	1	

Calcule os cortes A e B usando como fator de corte

.52 .74

E informe quantos produtos caem na categoria A e quantos na B

Calcule a reposicao de estoque do produto

aco 12mm

e calcule o custo total do estoque caso queira fazer compras

a cada 2d

E finalmente, calcule a quantidade de aquisições no ano para um lote econômico.

Observação: caso seu produto seja um condensador ou um misturador, em tese, eles teriam que ter lotes inteiros. Teoricamente, não seria possível comprar 1.5 misturadores. Entretanto, neste exemplo, esta restrição não será considerada, e você PODE comprar 1.5 misturadores. Faz-se isto para não complicar e não criar exceções ao modelo.

prods A	prods B	custo pedido	qtd aquisições

Algoritmo Lempel-Ziv Welch (LZW)

Acr nimo dos nomes dos seus inventores Lempel, Ziv e Welch. Lempel e Ziv publicaram suas id ias em 1977 (ZIV, Jacob, LEMPEL, Abraham; Compression of Individual Sequences Via Variable-Rate Coding, IEEE Transactions on Information Theory, September 1978) e Terry Welch refinou o algoritmo em 1984. (WELCH, T. A. (June 1984). "A technique for high-performance data compression."Computer. Vol. 17, pp. 8-19.)

Em s ntese, o algoritmo troca conjuntos de caracteres por c digos simples. N o h  an lise pr via do conte do e a compress o ocorre quando um conjunto grande de s mbolos   trocado por um  nico c digo. Os c digos LZW podem ter qualquer n mero de bits, mas precisam ser maiores do que a representa  o de 1 caracter (ou seja, maiores do que 8 bits). Os primeiros 256 c digos s o assinalados para o conjunto padr o inicial. Em um exemplo com c digos de 12 bits, de 0 a 255 s o caracteres simples e de 256 a 4095 s o para seq ncias.

Este algoritmo foi o escolhido para comprimir imagens do tipo GIF.

Sejam os algoritmos

1: **fun  o** COMPRIME {le caracteres e imprime c digos num ricos}
2: ST   primeiro caracter
3: **enquanto** h  caracteres na entrada **fa a**
4: CA   pr ximo caracter
5: **se** ST + CA est  no dicion rio **ent o**
6: ST   ST + CA
7: **sen o**
8: imprima o n mero da linha de ST {vem a ser o codigo de ST}
9: adicione ST + CA no dicion rio {na primeira linha disponivel}
10: ST   CA
11: **fim se**
12: **fim enquanto**
13: imprima o n mero da linha de ST {vem a ser o c digo}
14: **fim** {fun  o}

1: **fun  o** DESCOMPRIME {le c digos num ricos e imprime caracteres}
2: OL   primeiro c digo
3: CA   caracter que est  na linha OL
4: imprima CA
5: **enquanto** h  c digos na entrada **fa a**
6: NE   pr ximo c digo
7: **se** NE aponta para linha vazia em dic **ent o**
8: ST   caracter (es) da linha OL de dic
9: ST   ST + CA
10: **sen o**
11: ST   caracter (es) da linha NE de dic
12: **fim se**
13: imprima ST
14: CA   primeiro caractere de ST
15: inclua o caracter "OL"+ CA na primeira linha livre de dic
16: OL   NE
17: **fim enquanto**
18: **fim** {fun  o}

Problemas

O cl ssico problema dos algoritmos baseados em dicion rios   a mem ria necess ria. Quanto mais dados do arquivo s o lidos, mais seq ncias precisam ser armazenadas no dicion rio, e com isso, necessita-se de mais mem ria. O outro problema que isso causa   de espa o de endere amento: os c digos das entradas no dicion rio crescem junto com ele (em geral ocupam $\log_2(N)$ bits, onde N   o tamanho, ou n mero de entradas, do dicion rio). V rias abordagens podem ser adotadas para lidar com esse problema, as mais simples s o:

- Congelamento do dicion rio (quando o dicion rio atinge o tamanho limite, ele fica "congelado"e mais nenhuma entrada pode ser adicionada nele).
- Esvaziamento (o dicion rio   esvaziado e a compress o come a toda de novo). Essa t cnica pressup e que   mais vantajoso usar a redund ncia local que a redund ncia global para a compress o. Corresponde a separar o arquivo em "blocos"comprimidos separadamente.
- Uso de uma pol tica de esvaziamento que seja comum tanto ao compressor quanto ao descompressor (apagar entradas mais antigas, por exemplo).

Exemplo

Para este exemplo:

- Considere textos construidos apenas com as 10 primeiras letras
- O dicion rio ter  40 entradas. As 10 primeiras s o ABCDEFGHIJ
- As mensagens ter o 30 caracteres de comprimento

Seja comprimir a mensagem:

BDEFFFFFCEGGAFFHCFJDICEBHJCE

Ap s a aplica  o do algoritmo acima sobre esta mensagem o dicion rio ficar 

1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	BD
12	DE
13	EF
14	FF
15	FFF
16	FFFF
17	FE
18	EC
19	CE
20	EG
21	GG
22	GA
23	AF
24	FH
25	HC
26	CF
27	FJ
28	JD
29	DI
30	IC
31	CEB
32	BH
33	HJ
34	JC
35	
36	
37	
38	
39	
40	

E o resultado ser 

2 4 5 6 14 15 6 5 3 5 7 7 1 6 8 3 6 10 4 9 19 2 8 10 19

Para voc  fazer

Coloque em C os codigos resultados da compress o de

D E I D D D D D D I I I I I I
I I E D F C G A G H A D G A F

C[4]	C[10]	C[13]	C[15]	C[22]

(1 n mero)

Coloque em D os caracteres resultados da descompress o de

7 3 6 13 13 7 10 7 8 19 8
10 22 19 2 4 3 16 21 6 8 30

D[7]	D[15]	D[22]	D[23]	D[30]

(1 letra)

1

||||| ||||| ||||| - 1 - /

Fractal

Fractal, em matemática, é uma forma geométrica que é complexa e detalhada em qualquer nível de ampliação com que seja olhada. Frequentemente fractais são auto-selhantes, isto é, eles têm a propriedade de que cada porção pequena do fractal pode ser vista como uma réplica reduzida do todo. Um exemplo de um fractal é a "curva de floco de neve" construída levando um triângulo equilátero e triângulos equiláteros menores repetidamente erguendo-se no terço mediano dos lados progressivamente menores. Teoricamente, o resultado seria uma figura de área finita mas com um perímetro de comprimento infinito, consistindo em um número infinito de vértices. Em condições matemáticas, não pode ser diferenciada tal curva. Há muitas estruturas matemáticas que são fractais; por exemplo o triângulo de Sierpinski, o floco de neve de Koch, a curva de Peano, o conjunto de Mandelbrot, e o atrator de Lorenz.

Um ponto decisivo no estudo de fractais veio com a descoberta de geometria fractal pelo matemático francês Benoit B. Mandelbrot (nascido na Polônia) nos anos setenta. Mandelbrot adotou uma definição muito mais abstrata de dimensão do que a que é usada na geometria Euclidiana. O resultado é que um fractal não pode ser tratado como existindo estritamente em uma, duas, ou qualquer outra dimensão de número inteiro. Ao invés, deve ser alguma dimensão fracionária. A "curva de floco de neve" tem uma dimensão de 1.2618.

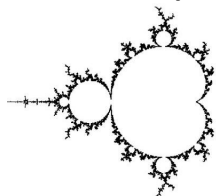
De acordo com Mandelbrot que inventou a palavra: "Eu cunhei fractal do adjetivo latino fractus. O verbo latino correspondente frangere significa quebrar: criar fragmentos irregulares, além de fragmentou (como em fração ou refração), fractus também deveria querer dizer "irregular", ambos significados que são preservados em fragmento.

A natureza é fractal. Construções teóricas é que são linhas retas, superfícies planas e assim por diante. Ou seja, não é o fractal que é artificial. É a geometria Euclidiana que o é... São exemplos de fractais: um litoral, uma samambaia, um perfil montanhoso, etc etc.

Também foram usados fractais para comprimir imagens em computadores. Em 1987, um matemático inglês descobriu a Transformada fractal que automaticamente descobre códigos de fractal em imagens do mundo real. A descoberta gerou o conceito de compressão fractal, usada em uma variedade de sistemas multimedia e outras aplicações de computador baseadas em imagem.

Benoit Mandelbrot

O fractal mais famoso que existe é o chamado "conjunto de Mandelbrot". Já se disse que este conjunto é o mais complexo elemento da matemática que os olhos humanos já viram.



Assim como os números reais podem ser dispostos na forma de uma reta numerada, os números complexos podem ser dispostos ao longo de um plano, que contem dois eixos (representados por a e b). Este plano denomina-se "plano de Argand-Gauss"

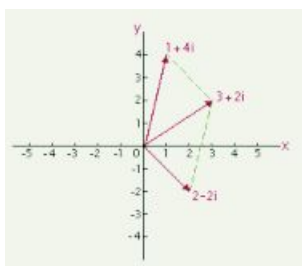


Figura 1: O plano de Argand Gauss

Embora sendo x e y dois números complexos, não se possa estabelecer uma relação de ordem (ou seja, é impossível dizer se $x > y$ ou vice-versa), podem-se operar dois complexos usando a adição, subtração, multiplicação e divisão.

Todas elas são realizadas como se os números x e y (complexos) fossem polinômios na variável i. Assim temos: $(3 + 4i) + (2 + i) = 5 + 5i$
 $i + (2 + i) = 2 + 2i$
 $2 + 3 = 5$
 $(3 + i) \times (2 - i) = 7 - i$

Isto posto, o CM, é obtido através da análise de todos os pontos de um conjunto que está no plano de Argand Gauss. O conjunto de Mandelbrot é o conjunto de todos os números complexos c tais que após um certo número de iterações $z = z^2 + c$, z não tende para infinito. O valor inicial de z é zero.

Mediante uma análise matemática simples, determina-se se para cada ponto, se ele pertence ou não ao CM. Pintando-se de uma cor a uns e de outra a outros, obtem-se uma visualização do CM, que tem este aspecto:

Para cada ponto do plano complexo (ou melhor dizendo para cada ponto de uma grade sobre o plano complexo), far-se-á uma sequência de iterações. Se ao final dessa sequência, o resultado final estiver fora do círculo de raio = 2, o ponto está fora do CM. Se o resultado for menor que 2, o ponto está dentro do CM. Embora o cálculo correto fosse testar o ponto contra infinito, pode-se fazer o teste com 2. (Se o ponto ultrapassa 2, ele tenderá ao infinito).

A fórmula da iteração é:

$$z_{n+1} = z_n^2 + c$$

, onde $z_0 = 0 + 0i$ e c é o ponto que está sendo testado.

É claro que para decidir se o ponto está ou não dentro do círculo de raio = 2, há que se ter um ponto de parada. Não se pode testar um ponto ao infinito. Se, nas primeiras 1000 iterações o ponto não saiu do círculo, não há garantia de que na vez 1001 ele não vá sair.

Portanto, a definição do CM é: O conjunto de pontos do plano complexo, que após iterados k vezes, segundo a fórmula $z_{n+1} = z_n^2 + c$ não caem fora do círculo de raio = 2.

Esta definição estaria ótima para pintar o fractal em 2 cores (pertence ou não pertence), mas há um truque que torna os fractais muito mais bonitos: a cor.

É hora de fazer uma distinção entre Fractal de Mandelbrot e Conjunto de Mandelbrot. O conjunto é apenas a reunião dos pontos que satisfazem a iteração acima. O fractal é um desenho formado por esse conjunto e pelos pontos externos, todos eles pintados de alguma maneira especificada.

Uso da Cor

Para o exercício a seguir, você deve usar a seguinte tabela de cores fuga do raio = 2

em menos de 3 iterações	cor preto
em 4 iterações	cinza
em 5 iterações	marrom
em 6	vermelho
em 7	amarelo
em 8	rosa
em 9	verde claro
de 10 a 14 (inclusive)	roxo
de 15 a 19	laranja
de 20 a 29	verde escuro
de 30 a 49	azul claro
de 50 a 74	creme
de 75 a 99	azul escuro
não foge antes de 100	branco

O zoom da imagem é obtido diminuindo-se o tamanho da malha que é colocada sobre o plano complexo. Por exemplo, podemos começar com os limites $-3 + 2i$ (canto superior esquerdo) e $3 - 3i$ para o canto inferior direito, com uma malha de 400 x 600 pontos, e incremento de 0,01 a cada ponto e centro em 0,0.

Se escolhermos um zoom de 10 vezes, ainda centrado na origem, a variação passa a ser de $-0.3 + 0.2i$ e de $0.3 - 0.2i$ com incremento de 0,001 e centro ainda em 0,0.

Note-se que o limite de ampliação da imagem (zoom) está limitado unicamente pela aritmética de ponto flutuante do computador e não pela característica do Conjunto de Mandelbrot.

Note-se também que quanto maior k, mais refinada é a imagem, enquanto mais demorada o é.

Um exemplo

A seguinte grade de pontos, dará origem ao trecho colorido:

0.4505 0.2163	0.4600 0.2163	0.4694 0.2163	0.4789 0.2163
0.4505 0.2149	0.4600 0.2149	0.4694 0.2149	0.4789 0.2149
0.4505 0.2135	0.4600 0.2135	0.4694 0.2135	0.4789 0.2135

Resposta

amarelo	vermelho	marrom	marrom
amarelo	vermelho	marrom	marrom
amarelo	vermelho	marrom	marrom

Para você fazer

Programar o fractal de mandelbrot em um computador, forneça os seguintes pontos iniciais e descubra com qual cor (da tabela acima) este ponto do fractal deverá ser colorido:

.4785 .3388	.4871 .3388	.4956 .3388	.5042 .3388
.4785 .3372	.4871 .3372	.4956 .3372	.5042 .3372
.4785 .3355	.4871 .3355	.4956 .3355	.5042 .3355

Resposta

The need to solve linear systems arises in an estimated 75% of all scientific computed problems (Dahlquist & Björck in *Numerical Methods*, 1974).

Cálculo Numérico - Sistemas Lineares

Uma equação é linear se cada um de seus termos contém não mais do que uma variável e se cada variável aparece na primeira potência. Por exemplo $3x + 2y + 3z = 8$ é linear, mas $3x + 2yz + 3w = 27$ não é, pois o segundo termo tem yz e também $2x^2 + 4y + 5z = 90$ também não é linear, pois aparece x^2 .

Vamos considerar n equações lineares com n incógnitas e vamos nos referir a esse conjunto como um Sistema Linear de ordem n . Uma solução para tal sistema é um conjunto de n valores para as incógnitas, de maneira a que quando esses valores substituímos as variáveis nas equações, todas elas são simultaneamente satisfeitas. Por exemplo, o sistema

$$\begin{cases} x + y + z = 1 \\ x - y - z = 1 \\ 2x + 3y - 4z = 9 \end{cases}$$

tem a solução $x = 1$, $y = 1$ e $z = -1$. Você pode verificar e se certificar disso substituindo estas variáveis no sistema. O mesmo sistema também pode ser escrito na forma matricial

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 2 & 3 & -4 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 9 \end{pmatrix}$$

Para quem, como nós, conhece a multiplicação matricial, o sistema acima também pode ser escrito como

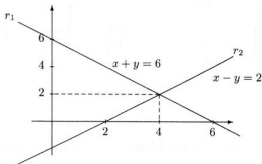
$$A \cdot x = b,$$

onde A é chamada Matriz de Coeficientes, b é o termo independente e x é o vetor solução. Dado um sistema linear qualquer o mesmo pode ser classificado em termos do número de soluções que ele admite:

- Sistema possível com solução única. Por exemplo, seja

$$\begin{cases} x + y = 6 \\ x - y = 2 \end{cases}$$

Este sistema, que pode ser representado na figura

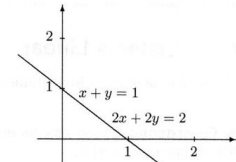


admite uma única solução, representada pelo ponto (4, 2).

- Sistema possível com infinitas soluções. Por exemplo, seja

$$\begin{cases} x + y = 1 \\ 2x + 2y = 2 \end{cases}$$

Como se pode ver na figura

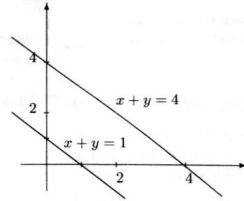


as soluções são infinitas (sobre toda a linha)

- Sistema impossível, sem solução. Por exemplo

$$\begin{cases} x + y = 1 \\ x + y = 4 \end{cases}$$

Como se pode ver na figura, as retas são paralelas e portanto não têm ponto comum.



Método do escalonamento de Gauss

Resolver um sistema linear $A \cdot x = b$ é achar um sistema equivalente $A' \cdot x = b'$ que tenha solução imediata, ou pelo menos mais simples do que o sistema original. Dois sistemas são ditos equivalentes se, embora diferentes, têm a mesma solução. As operações que são realizadas sobre os sistemas e que não lhes alteram as soluções, são:

- Trocar uma linha (equação) de lugar com outra.
- Multiplicar uma linha (equação) por uma constante
- Trocar uma linha por uma soma de 2 linhas

Seja por exemplo, resolver o sistema

$$\begin{cases} x + 4y + 4z = 23 \\ 3x + 3y + z = 27 \\ 2x + 5y + 2z = 25 \end{cases}$$

Passando para uma matriz mais simples

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 3 & 3 & 1 & 27 \\ 2 & 5 & 2 & 25 \end{array} \right)$$

aplicando à terceira linha a propriedade que diz $L_3 = L_3 - p \cdot L_1$ e escolhendo p de modo que surja um zero na primeira posição da linha, fica: $2, 5, 2, 25 - p \cdot 1, 4, 4, 23$ e escolhendo $p = 2$, a nova linha passa a ser $0, -3, -6, -21$ e a matriz fica:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 3 & 3 & 1 & 27 \\ 0 & -3 & -6 & -21 \end{array} \right)$$

aplicando à segunda linha a propriedade que diz $L_2 = L_2 - p \cdot L_1$ e fazendo $p = 3$ para que surja um zero na primeira posição e a linha fica: $3, 3, 1, 27 - p \cdot 1, 4, 4, 23$. Com $p = 3$, a nova linha fica sendo $0, -9, -11, -42$ e a matriz agora é:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 0 & -9 & -11 & -42 \\ 0 & -3 & -6 & -21 \end{array} \right)$$

Finalmente, elimina-se o -3 na matriz acima de maneira a deixar a matriz escalonada: $L_3 = L_3 - p \cdot L_2$ e fazendo $p = 0.333333$, a segunda linha fica sendo $0, 0, -2.33333, -7$ e a matriz escalonada agora é:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 0 & -9 & -11 & -42 \\ 0 & 0 & -2.33333 & -7 \end{array} \right)$$

Examinando a última linha, percebe-se que $z = \frac{-7}{-2.33333} = 3$. Levando este valor para a linha anterior, acha-se $y = 1$ e levando os dois valores na primeira linha, acha-se $x = 7$. Ou seja, o vetor $\{7, 1, 3\}$ é a solução procurada do sistema. O algoritmo em Freemat que resolve este sistema é

```
function trisup(a)
n=size(a,1);
for passo=1:n-1
for i=passo+1:n
pivo=a(i,passo)/a(passo,passo);
for j=1:n
a(i,j)=a(i,j)-pivo*a(passo,j);
end
a(i,n+1)=a(i,n+1)-pivo*a(passo,n+1);
end
end
x(n)=a(n,n+1)/a(n,n);
for i=n-1:-1:1
x(i)=a(i,n+1);
for j=i+1:n
x(i)=x(i)-a(i,j)*x(j);
end
x(i)=x(i)/a(i,i);
end
disp(a);
disp(x);
```

Método LU

No método LU, deve-se transformar a matriz A (dos coeficientes) em um produto matricial $L \cdot U$ onde a matriz U (de Upper) é a própria matriz A devidamente escalonada (como visto no método anterior) e a matriz L (de Lower) é uma matriz que contém unidades na diagonal principal, zeros acima destas posições e que contém os pivots do método anterior abaixo da diagonal principal.

No exemplo anterior, as matrizes L e U são

$$U = \begin{pmatrix} 1.00 & 4.00 & 4.00 \\ 0 & -9.00 & -11.00 \\ 0 & 0 & -2.33 \end{pmatrix}$$

$$L = \begin{pmatrix} 1.00 & 0 & 0 \\ 3.00 & 1.00 & 0 \\ 2.00 & 0.33 & 1 \end{pmatrix}$$

Note que a multiplicação matricial entre $L \cdot U$ recupera a matriz A . Obtendo estas duas matrizes, é fácil calcular y (o vetor de termos independentes que é calculado junto com a matriz escalonada no método anterior e depois o vetor x de soluções. Em termos algorítmicos, fica $[L]\{y\} = \{b\}$. Agora é praticamente imediato calcular o vetor $\{y\}$. Finalmente, o sistema se resolve calculando $[U]\{x\} = \{y\}$.

Acompanhe o algoritmo

```
function lua1(u,b)
n=size(u,2);
for passo=1:n-1
for i=passo+1:n
pivo=u(i,passo)/u(passo,passo);
for j=1:n
u(i,j)=u(i,j)-pivo*u(passo,j);
end
L(i,passo)=pivo;
end
end
for i=1:n-1
L(i,i)=1;
for j=i+1:n
L(i,j)=0;
end
end
y(1)=b(1);
for i=2:n
y(i)=b(i);
for j=1:i-1
y(i)=y(i)-L(i,j)*y(j);
end
end
x(n)=y(n)/u(n,n);
for i=n-1:-1:1
x(i)=y(i);
for j=i+1:n
x(i)=x(i)-u(i,j)*x(j);
end
x(i)=x(i)/u(i,i);
end
disp('L');
disp(L);
disp('U');
disp(U);
disp('b');
disp(b);
disp('solucao');
disp(x);
```

Para você fazer

Resolva pelos dois métodos o sistema

$$\begin{pmatrix} 6 & 1 & 5 & 7 & 2 & 96 \\ 7 & 7 & 1 & 5 & 2 & 88 \\ 4 & 2 & 2 & 1 & 6 & 63 \\ 4 & 1 & 5 & 1 & 6 & 86 \\ 2 & 1 & 6 & 5 & 6 & 87 \end{pmatrix}$$

e mostre no primeiro método a matriz escalonada e no segundo método as matrizes L e U (pode usar o verso desta folha, além de apresentar os 5 valores do vetor resposta x , que por óbvio devem ser os mesmos em ambos os métodos

x	y	z	w	t



- 1 - /

The need to solve linear systems arises in an estimated 75% of all scientific computed problems (Dahlquist & Björck in *Numerical Methods*, 1974).

Cálculo Numérico - Sistemas Lineares

Uma equação é linear se cada um de seus termos contém não mais do que uma variável e se cada variável aparece na primeira potência. Por exemplo $3x + 2y + 3z = 8$ é linear, mas $3x + 2yz + 3w = 27$ não é, pois o segundo termo tem yz e também $2x^2 + 4y + 5z = 90$ também não é linear, pois aparece x^2 .

Vamos considerar n equações lineares com n incógnitas e vamos nos referir a esse conjunto como um Sistema Linear de ordem n . Uma solução para tal sistema é um conjunto de n valores para as incógnitas, de maneira a que quando esses valores substituímos as variáveis nas equações, todas elas são simultaneamente satisfeitas. Por exemplo, o sistema

$$\begin{cases} x + y + z = 1 \\ x - y - z = 1 \\ 2x + 3y - 4z = 9 \end{cases}$$

tem a solução $x = 1$, $y = 1$ e $z = -1$. Você pode verificar e se certificar disso substituindo estas variáveis no sistema. O mesmo sistema também pode ser escrito na forma matricial

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 2 & 3 & -4 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 9 \end{pmatrix}$$

Para quem, como nós, conhece a multiplicação matricial, o sistema acima também pode ser escrito como

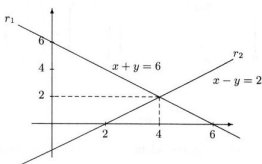
$$A \cdot x = b,$$

onde A é chamada Matriz de Coeficientes, b é o termo independente e x é o vetor solução. Dado um sistema linear qualquer o mesmo pode ser classificado em termos do número de soluções que ele admite:

- Sistema possível com solução única. Por exemplo, seja

$$\begin{cases} x + y = 6 \\ x - y = 2 \end{cases}$$

Este sistema, que pode ser representado na figura

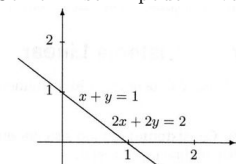


admite uma única solução, representada pelo ponto $(4, 2)$.

- Sistema possível com infinitas soluções. Por exemplo, seja

$$\begin{cases} x + y = 1 \\ 2x + 2y = 2 \end{cases}$$

Como se pode ver na figura

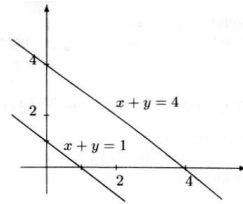


as soluções são infinitas (sobre toda a linha)

- Sistema impossível, sem solução. Por exemplo

$$\begin{cases} x + y = 1 \\ x + y = 4 \end{cases}$$

Como se pode ver na figura, as retas são paralelas e portanto não têm ponto comum.



Método do escalonamento de Gauss

Resolver um sistema linear $A \cdot x = b$ é achar um sistema equivalente $A' \cdot x = b'$ que tenha solução imediata, ou pelo menos mais simples do que o sistema original. Dois sistemas são ditos equivalentes se, embora diferentes, têm a mesma solução. As operações que são realizadas sobre os sistemas e que não lhes alteram as soluções, são:

- Trocar uma linha (equação) de lugar com outra.
- Multiplicar uma linha (equação) por uma constante
- Trocar uma linha por uma soma de 2 linhas

Seja por exemplo, resolver o sistema

$$\begin{cases} x + 4y + 4z = 23 \\ 3x + 3y + z = 27 \\ 2x + 5y + 2z = 25 \end{cases}$$

Passando para uma matriz mais simples

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 3 & 3 & 1 & 27 \\ 2 & 5 & 2 & 25 \end{array} \right)$$

aplicando à terceira linha a propriedade que diz $L_3 = L_3 - p \cdot L_1$ e escolhendo p de modo que surja um zero na primeira posição da linha, fica: $2, 5, 2, 25 - p \cdot 1, 4, 4, 23$ e escolhendo $p = 2$, a nova linha passa a ser $0, -3, -6, -21$ e a matriz fica:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 3 & 3 & 1 & 27 \\ 0 & -3 & -6 & -21 \end{array} \right)$$

aplicando à segunda linha a propriedade que diz $L_2 = L_2 - p \cdot L_1$ e fazendo $p = 3$ para que surja um zero na primeira posição e a linha fica: $3, 3, 1, 27 - p \cdot 1, 4, 4, 23$. Com $p = 3$, a nova linha fica sendo $0, -9, -11, -42$ e a matriz agora é:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 0 & -9 & -11 & -42 \\ 0 & -3 & -6 & -21 \end{array} \right)$$

Finalmente, elimina-se o -3 na matriz acima de maneira a deixar a matriz escalonada: $L_3 = L_3 - p \cdot L_2$ e fazendo $p = 0.333333$, a segunda linha fica sendo $0, 0, -2.33333, -7$ e a matriz escalonada agora é:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 0 & -9 & -11 & -42 \\ 0 & 0 & -2.33333 & -7 \end{array} \right)$$

Examinando a última linha, percebe-se que $z = \frac{-7}{-2.33333} = 3$. Levando este valor para a linha anterior, acha-se $y = 1$ e levando os dois valores na primeira linha, acha-se $x = 7$. Ou seja, o vetor $\{7, 1, 3\}$ é a solução procurada do sistema. O algoritmo em Python que resolve este sistema é

```
def gauss(a,b):
    n=len(b)
    x=[0]*n
    for passo in range(0,n-1):
        for i in range(passo+1,n):
            pivot=a[i,passo]/a[passo,passo]
            for j in range(0,n):
                a[i,j]=a[i,j]-(pivot*a[passo,j])
            b[i]=b[i]-pivot*b[passo]
```

```
x[n-1]=b[n-1]/a[n-1,n-1]
for i in range(i,-1,-1):
    x[i]=b[i]
    for j in range(i+1,n):
        x[i]=x[i]-a[i,j]*x[j]
    x[i]=x[i]/a[i,i]
return(x)
import numpy as np
a=np.array([[3.0,-1,1],[1,2,-1],[4,-2,-1]],float)
b=np.array([11.0,6,-5],float)
x=gauss(a,b)
print(x)
```

Método LU

No método LU, deve-se transformar a matriz A (dos coeficientes) em um produto matricial $L \cdot U$ onde a matriz U (de Upper) é a própria matriz A devidamente escalonada (como visto no método anterior) e a matriz L (de Lower) é uma matriz que contém unidades na diagonal principal, zeros acima destas posições e que contém os pivots do método anterior abaixo da diagonal principal.

No exemplo anterior, as matrizes L e U são

$$U = \begin{pmatrix} 1.00 & 4.00 & 4.00 \\ 0 & -9.00 & -11.00 \\ 0 & 0 & -2.33 \end{pmatrix}$$

$$L = \begin{pmatrix} 1.00 & 0 & 0 \\ 3.00 & 1.00 & 0 \\ 2.00 & 0.33 & 1 \end{pmatrix}$$

Note que a multiplicação matricial entre $L \cdot U$ recupera a matriz A . Obtendo estas duas matrizes, é fácil calcular y (o vetor de termos independentes que é calculado junto com a matriz escalonada no método anterior e depois o vetor x de soluções. Em termos algorítmicos, fica $[L]\{y\} = \{b\}$. Agora é praticamente imediato calcular o vetor $\{y\}$. Finalmente, o sistema se resolve calculando $[U]\{x\} = \{y\}$. Acompanhe o algoritmo

```
def lu1(u,b):
    n=len(u)
    x=[0]*n
    y=[0]*n
    L=np.zeros((n,n))
    for passo in range(0,n):
        for i in range(passo+1,n):
            pivot=u[i,passo]/u[passo,passo]
            for j in range(0,n):
                u[i,j]=u[i,j]-pivot*u[passo,j]
            L[i,passo]=pivot
    for i in range(0,n):
        L[i,i]=1
    y[0]=b[0]
    for i in range(1,n):
        y[i]=b[i]
        for j in range(0,i):
            y[i]=y[i]-L[i,j]*y[j]
    x[n-1]=y[n-1]/u[n-1,n-1]
    for i in range(n-1,-1,-1):
        x[i]=y[i]
        for j in range(i+1,n):
            x[i]=x[i]-u[i,j]*x[j]
        x[i]=x[i]/u[i,i]
    print('-----L-----')
    print(L)
    print('-----y-----')
    print(y)
    print('-----u-----')
    print(u)
    print('-----x-----')
    print(x)
import numpy as np
a=np.array([[3.0,-1,1],[1,2,-1],[4,-2,-1]],float)
b=np.array([11.0,6,-5],float)
lu1(a,b)
```

Para você fazer

Resolva pelos dois métodos o sistema

$$\begin{pmatrix} 3 & 5 & 6 & 2 & 3 & 42 \\ 4 & 1 & 6 & 7 & 3 & 50 \\ 5 & 6 & 3 & 1 & 4 & 34 \\ 3 & 7 & 5 & 4 & 4 & 59 \\ 1 & 2 & 7 & 5 & 6 & 52 \end{pmatrix}$$

e mostre no primeiro método a matriz escalonada e no segundo método as matrizes L e U (pode usar o verso desta folha, além de apresentar os 5 valores do vetor resposta x , que por óbvio devem ser os mesmos em ambos os métodos)

x	y	z	w	t



Tomografia Computadorizada Axial

O coeficiente de atenuação linear média μ_t de cada pixel é comparado com o coeficiente da água, μ_a , definindo o número CT:

$$CT = 1000(\mu_t - \mu_a)/\mu_a.$$

A água é utilizada como referência porque seu coeficiente de atenuação é similar ao dos tecidos moles, e é um material fácil de obter para calibrar os aparelhos. O coeficiente 1000 é utilizado para obter números inteiros.

O número CT, ou coeficiente de Hounsfield, é definido como -1000 para o ar e 0 para a água. Para os tecidos em geral, ele depende da energia do feixe empregado. Por exemplo, para 80 keV, se o coeficiente de atenuação linear típico de ossos é de $0,38\text{ cm}^{-1}$, e da água $0,19\text{ cm}^{-1}$, o número CT dos ossos é de +1000. Pode ser ainda maior para ossos corticais. Estes valores também variam de aparelho para aparelho, já que os coeficientes dependem da distribuição de energia do feixe. A radiação observada, I , está relacionada com a radiação na fonte, I_0 , por: $I = I_0 e^{-\mu x}$.

Tecido	CT	Tecido	CT
Ar	-1000	Sangue	35 a 55
Pulmão	-900 a -400	Sangue Coagulado	80
Gordura	-110 a -65	Músculo	40 a 60
gua	0	Fígado	50 a 85
Rim	30	Ossos	130 a 250

Por convenção, altos valores de CT são imageados como branco, e baixos como preto. Como o olho humano não pode distinguir os milhares de coeficientes, utilizamos a técnica de janelas (windowing), para graficar somente os valores em uma certa faixa.

Aquisição de dados Na tomografia computadorizada mais comum, um tubo com um feixe de cerca de 0,6 mm de diâmetro gira em torno do paciente, e emerge do paciente sobre um detector com aproximadamente 700 sensores, que convertem a intensidade em uma corrente. Cada pulso de raio-X dura 2 a 3 ms, completando uma volta em cerca de 1 s. Cada 360° gera 300 somas.

Cada vez que o tubo emite um pulso, cada detector mede o logaritmo da intensidade que recebe. Este valor representa a soma de todos os números CT dos voxels atravessados pelo raio, completando uma projeção. Cada voxel é atravessado pelo feixe em diferentes direções, durante a rotação do anel. O número CT de cada voxel está portanto representado em várias somas.

Algoritmo de reconstrução Vamos pensar em uma sessão transversal de um corpo humano. Imagine também um conjunto de emissores de raio-X, colocados paralelos em uma estrutura rígida que gira em um eixo estabelecido no centro do corpo humano. No lado oposto (ultrapassado o corpo) estão os detectores do raio-X.

medida em que a máquina gira, inumeros conjuntos de Raios-X vão sendo tomados. Para uma determinada posição angular (digamos horizontal) tem-se um conjunto de x valores de atenuação. O valor x é a quantidade de emissores e detetores, e a atenuação é a diminuição da potência do Raio-X emitido numa ponta e recebido na outra após passar pelas estruturas do corpo que está sendo estudado.

Por exemplo, na horizontal, os raios inferiores, tem que atravessar a coluna (supondo que o paciente esteja deitado de costas), ao passo que os raios superiores ou não atravessam nada, ou apenas alguma gordura (isso para os mais gordinhos). Quando a estrutura estiver tirando raios X na vertical, a coluna do paciente atenuará os raios do meio do conjunto.

Os detectores não tiram fotografias, ao invés eles mandam sinais elétricos proporcionais a potência do Raio-X recebido diretamente para um computador. Este, de posse da informação referente ao ângulo de tomada e dos valores de todos os sensores, vai reconstituindo do corpo que está sendo "cortado".

A técnica mais simples de reconstrução é a chamada "back projection", e todas as demais que vieram depois são melhoramentos e variações desta.

Sua formulação matemática é bem compacta:

$$P(\theta, t) = \int_R f(x, y) \delta(x \sin \theta - y \cos \theta - t) dx dy$$

Essa fórmula pode ser lida como: P é a atenuação medida no sensor t , posicionado em um ângulo θ em relação ao eixo horizontal. x, y são as coordenadas dos pontos da imagem e δ é um filtro que tem a função de só considerar os pontos que estão no caminho do fluxo t de Raios X.

Exercício Suponha uma caixa fechada e lacrada. Você sabe que existem lá dentro três bananas de dinamite. Qualquer tentativa de abrir a caixa implicará na explosão. Você submete a caixa a uma tomografia axial. O objetivo é saber onde estão as 3 bananas de dinamite.

Considere a caixa mapeada em uma matriz de 11 x 11. As bananas estão verticais na caixa. O material de que é feita a caixa é transparente aos Raios-X. Já a dinamite (trinitrotolueno) é um material que absorve completamente os RX.

A tomografia será composta de 6 tomogramas, obtidos com inclinações de 0, 30, 60, 90, 120 e 150 graus. A quantidade de emissores/receptores de RX é 11.

Os mapas de atravessamento de cada emissor/receptor em cada uma das inclinações são:

0 grau	30 graus	60 graus
B B B B B B B B B B B	8 8 9 9 A A B B 0 0 0	4 5 6 7 8 0 9 A B 0 0
A A A A A A A A A A	7 7 8 8 9 9 A A B B 0	4 5 5 6 7 8 9 A B B 0
9 9 9 9 9 9 9 9 9 9	6 7 7 8 8 9 9 A A B B	3 4 5 6 7 0 8 9 A B 0
8 8 8 8 8 8 8 8 8 8	5 6 6 7 7 8 8 9 9 A A	3 4 4 5 6 7 8 9 A A B
7 7 7 7 7 7 7 7 7 7	4 5 5 6 6 7 7 8 8 9 9	2 3 4 5 6 0 7 8 9 A B
6 6 6 6 6 6 6 6 6 6	0 4 0 5 0 6 0 7 0 8 0	2 3 3 4 5 6 7 8 9 9 A
5 5 5 5 5 5 5 5 5 5	3 3 4 4 5 5 6 6 7 7 8	1 2 3 4 5 0 6 7 8 9 A
4 4 4 4 4 4 4 4 4 4	2 2 3 3 4 4 5 5 6 6 7	1 2 2 3 4 5 6 7 8 8 9
3 3 3 3 3 3 3 3 3 3	1 1 2 2 3 3 4 4 5 5 6	0 1 2 3 4 0 5 6 7 8 9
2 2 2 2 2 2 2 2 2 2	0 1 1 2 2 3 3 4 4 5 5	0 1 1 2 3 4 5 6 7 7 8
1 1 1 1 1 1 1 1 1 1	0 0 0 1 1 2 2 3 3 4 4	0 0 1 2 3 0 4 5 6 7 8
90 graus	120 graus	150 graus
1 2 3 4 5 6 7 8 9 A B	0 0 1 2 3 0 4 5 6 7 8	0 0 0 1 1 2 2 3 3 4 4
1 2 3 4 5 6 7 8 9 A B	0 1 1 2 3 4 5 6 7 7 8	0 1 1 2 2 3 3 4 4 5 5
1 2 3 4 5 6 7 8 9 A B	0 1 2 3 4 0 5 6 7 8 9	1 1 2 2 3 3 4 4 5 5 6
1 2 3 4 5 6 7 8 9 A B	1 2 2 3 4 5 6 7 8 8 9	2 2 3 3 4 4 5 5 6 6 7
1 2 3 4 5 6 7 8 9 A B	1 2 3 4 5 0 6 7 8 9 A	3 3 4 4 5 5 6 6 7 7 8
1 2 3 4 5 6 7 8 9 A B	2 3 3 4 5 6 7 8 9 9 A	0 4 0 5 0 6 0 7 0 8 0
1 2 3 4 5 6 7 8 9 A B	2 3 4 5 6 0 7 8 9 A B	4 5 5 6 6 7 7 8 8 9 9
1 2 3 4 5 6 7 8 9 A B	3 4 4 5 6 7 8 9 A A B	5 6 6 7 7 8 8 9 9 A A
1 2 3 4 5 6 7 8 9 A B	3 4 5 6 7 0 8 9 A B 0	6 7 7 8 8 9 9 A A B B
1 2 3 4 5 6 7 8 9 A B	4 5 5 6 7 8 9 A B B 0	7 7 8 8 9 9 A A B B 0
1 2 3 4 5 6 7 8 9 A B	4 5 6 7 8 0 9 A B 0 0	8 8 9 9 A A B B 0 0 0

- Para reconstituir a caixa contendo as 3 dinamites, você deve:
- ★ Criar uma matriz 11 x 11 contendo inicialmente zeros.
 - ★ Para cada tomograma, e para cada um dos emissores/sensores, você deve localizar qual o ângulo usado e quanto de absorção foi registrado.
 - ★ Multiplique este valor pela quantidade de pixels percorridos e divida por 11 (normalização devido à geometria da malha)
 - ★ Some este valor calculado a todos os pixels que fazem parte deste caminho (verifique isto nos mapas de atravessamento).
 - ★ Ao final do processo, divida a matriz pelo número de tomogramas (6). Os 3 maiores valores encontrados correspondem as bananas de dinamite.
- Considere o seguinte caso exemplo,

Exemplo Os valores dos 6 tomogramas, são:

°	j=1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	10	10	0	0	10	0	0
30	0	0	9	0	0	10	0	0	0	7	0
60	0	7	0	0	0	10	0	0	0	7	0
90	0	10	0	0	0	10	0	0	10	0	0
120	0	0	9	0	0	10	11	0	0	0	0
150	0	0	0	0	22	10	0	0	0	0	0

e a imagem resultante é:

col=1	2	3	4	5	6	7	8	9	10	11
.0	1.7	1.7	.0	2.3	2.5	.0	.9	3.3	2.0	.0
.0	1.7	.0	1.7	1.4	1.7	.9	3.4	3.7	6.0	4.0
3 .3	3.3	1.7	4.7	1.7	3.3	1.7	4.2	11.0	5.6	3.3
.0	3.3	1.7	1.4	1.7	1.7	5.6	6.0	4.2	3.4	.9
.9	1.7	1.4	1.7	7.3	5.6	3.3	3.7	1.7	.9	.0
2.5	4.7	3.0	5.6	1.7	10.0	3.7	1.7	3.3	1.7	2.5
3.0	10.9	5.6	3.3	5.0	3.3	7.0	3.3	3.3	1.7	2.5
5.3	4.2	3.9	1.4	1.7	3.7	1.7	.0	3.3	1.7	.0
3.0	1.7	.9	1.7	3.4	3.0	.0	1.7	1.7	.0	1.7
.0	1.7	.0	2.5	2.0	3.0	1.4	1.7	1.7	.0	.0
.0	1.7	1.7	2.9	.0	1.7	.0	1.4	4.7	.0	.0

Como se pode ver no exemplo acima, as bananas de dinamite estão colocadas nas posições onde houve maior absorção dentro da caixa. Tais posições são: linha 3, coluna 9, com valor de absorção=11.0; linha=7, coluna 2, valor=10.9 e linha 6, coluna 6 valor 10.0

Para você fazer

Sejam os seguintes 6 tomogramas. Onde estão as bananas ?

°	j=1	2	3	4	5	6	7	8	9	10	11
0	0	10	0	0	10	0	0	0	0	10	0
30	5	0	0	11	0	0	0	0	9	0	0
60	5	0	0	11	0	0	0	11	0	0	0
90	0	0	10	10	0	10	0	0	0	0	0
120	0	0	0	11	22	0	0	0	0	0	0
150	0	0	9	0	0	10	0	11	0	0	0

Bibliografia: The Turing Omnibus, A.K.Dewdney. Obs: Não confundir número da linha com número do jato. São coisas distintas.

As 3 bananas de dinamite estão nas posições:

=====:



Manuseio de arquivos tipo BMP

O padrão BMP (bit mapping) é um dos mais simples padrões de representacao de imagens no ambiente windows. Grosso modo, uma imagem BMP é um arquivo que contém as seguintes informações:

- Bloco de controle
- Tabela cores (se a imagem for mapeada)
- Imagem

O bloco de controle tem a seguinte configuração

Tipo	Tam	Desl	Nome	Descrição	exemplo
char	2	0	type	constante igual a BM	424D
long int	4	2	size	tamanho do arquivo	C6 04 00 00
int	2	6	reserved	zeros	00 00
int	2	8	reserved	zeros	00 00
long int	4	A	off	desloc até a imagem	36 04 00 00
long int	4	E	size	resto do bloco controle	28 00 00 00
long int	4	12	width	n. de colunas	0A 00 00 00
long int	4	16	height	n. de linhas	0C 00 00 00
int	2	1A	planes	planos	01 00
int	2	1C	bitcount	bits / pixel	08 00
long int	4	1E	compress	compressao	00 00 00 00
long int	4	22	sizeimage	tamanho da imagem	90 00 00 00
long int	4	26	pix/m H	pixels / m na horizontal	CE 0E 00 00
long int	4	2A	pix/m V	na vertical	D8 0E 00 00
xxx	8	2E	xxx	xxx	

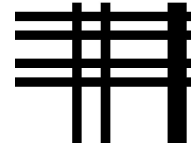
- Note que
- A tabela de cores tem tamanho variável (1024 bytes no máximo), começa no endereço 36 e cada entrada contém as quantidades de azul/verde e vermelho. (note a inversão)
 - O quarto byte de cada entrada é sempre zeros binários.
 - A imagem começa em 536 (se a tabela = 1024) e cada linha está alinhada em múltiplo de 32 bits, por questões de eficiência.
 - Na tabela, cada número indica a quantidade de tinta, assim: FF = branco e 00 = preto.
 - Como em todas as entradas da tabela de cores, tem-se R = G = B, pode-se concluir que esta imagem é monocromática, ou como se diz “branco e preto”.
 - a ordem das cores na tabela de cores é BGR, ou o contrário do nome do modelo que é RGB.

Seja como exemplo uma imagem devidamente *dumpeada*: É uma imagem branca, contendo 4 linhas V e 4 linhas H de cor cinza.

```

0 1 2 3 4 5 6 7 8 9 A B C D E F
-- -- -- -- -- -- -- -- -- --
0000 42 4D 62 05 00 00 00 00 00 00 00 36 04 00 00 28 00 BM.....6...(.
0010 00 00 13 00 00 00 0F 00 00 00 01 00 08 00 00 00 .....
0020 00 00 60 01 00 00 00 00 00 00 00 00 00 00 00 01 .....
0030 00 00 00 00 00 00 00 00 00 00 01 01 01 00 02 02 .....
0040 02 00 03 03 03 00 04 04 04 00 05 05 05 00 06 06 .....
0050 06 00 07 07 07 00 08 08 08 00 09 09 09 00 0A 0A .....
0060 0A 00 0B 0B 0B 00 0C 0C 00 0D 0D 0D 00 0E 0E .....
0070 0E 00 0F 0F 0F 00 10 10 10 00 11 11 11 00 12 12 .....
0080 12 00 13 13 13 00 14 14 14 00 15 15 15 00 16 16 .....
...
0410 F6 00 F7 F7 F7 00 F8 F8 F8 00 F9 F9 F9 00 FA FA .....
0420 FA 00 FB FB FB 00 FC FC FC 00 FD FD FD 00 FE FE .....
0430 FE 00 FF FF FF 00 FF FF FF FF FF FF 7F FF 7F .....
0440 FF FF FF FF FF FF 7F 7F FF 00 FF FF FF FF FF FF .....
0450 7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 FF FF .....
0460 FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F .....
0470 FF 00 FF FF FF FF FF FF 7F FF FF 7F FF FF FF FF .....
0480 FF FF 7F 7F FF 00 FF FF FF FF FF FF 7F FF FF 7F .....
0490 FF FF FF FF FF FF 7F 7F FF 00 FF FF FF FF FF FF .....
04A0 7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 7F 7F .....
04B0 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F .....
04C0 7F 00 FF FF FF FF FF FF 7F FF FF 7F FF FF FF FF .....
04D0 FF FF 7F 7F FF 00 7F 7F 7F 7F 7F 7F 7F 7F 7F .....
04E0 7F 7F 7F 7F 7F 7F 7F 7F 00 FF FF FF FF FF FF .....
04F0 7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 FF FF .....
0500 FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F .....
0510 FF 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F .....
0520 7F 7F 7F 7F 7F 00 FF FF FF FF FF FF 7F FF FF 7F .....
0530 FF FF FF FF FF FF 7F 7F FF 00 7F 7F 7F 7F 7F 7F .....
0540 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 00 FF FF .....
0550 FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F .....
0560 FF 00 ..
```

A imagem tem 15 linhas e 19 colunas. Os riscos estão nas linhas 7, 9, 12 e 14 e nas colunas 7, 10, 17 e 18.



Para você fazer

Você verá abaixo as principais partes de um arquivo grafico BMP de nome: IXAGE001.BMP Trata-se de uma imagem retangular, branca, contendo 4 linhas verticais e 4 linhas horizontais de cor cinza.

```

0 1 2 3 4 5 6 7 8 9 A B C D E F
-- -- -- -- -- -- -- -- -- --
0000 62 6D 56 05 00 00 00 00 00 00 00 36 84 00 00 3A 00 BM.....6...(.
0010 00 00 10 00 00 00 12 00 00 00 81 00 08 00 00 00 .....
0020 00 00 A6 81 00 00 00 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 00 00 00 00 00 00 81 81 00 82 82 .....
0040 82 00 83 83 83 00 84 84 84 00 85 85 85 00 86 86 .....
0050 86 00 87 87 87 00 88 88 88 00 89 89 89 00 0A 0A .....
0060 0A 00 8B 8B 8B 00 8C 8C 8C 00 0D 0D 0D 00 8E 8E .....
0070 8E 00 8F 8F 8F 00 0C 0C 0C 00 0B 0B 0B 00 92 92 .....
...
0410 2B 00 FB FB FB 00 4A 4A 4A 00 57 57 57 00 17 17 .....
0420 17 00 11 11 11 00 10 10 10 00 40 40 40 00 21 21 .....
0430 21 00 20 20 20 00 FF FF FF 7F 7F FF 7F FF 7F FF .. ...}}.}.
0440 FF FF FF FF FF FF 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F .....}}}}}}}}}}
0450 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F .....}}}}}}}}}}
0460 7F 7F 7F 7F 7F 7F FF FF FF 7F 7F FF 7F FF 7F FF .....}}}}}}...}.
0470 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
0480 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
0490 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
04A0 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
04B0 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
04C0 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
04D0 FF FF FF FF FF FF 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F .....}}}}}}}}}}
04E0 7F 7F 7F 7F 7F 7F FF FF FF 7F 7F FF 7F FF 7F FF .....}}}}...}.
04F0 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
0500 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
0510 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
0520 FF FF FF FF FF FF FF FF FF 7F 7F FF 7F FF 7F FF .....}}.}.
0530 FF FF FF FF FF FF 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F .....}}}}}}}}}}
0540 7F 7F 7F 7F 7F 7F FF FF FF 7F 7F FF 7F FF 7F FF .....}}}}...}.
0550 FF FF FF FF FF FF .....

```

Note que no arquivo, a primeira linha começa em X'436'. Na imagem, esta linha corresponde à linha inferior. Ou seja, no arquivo a contagem começa de cima para baixo, e na imagem, de baixo para cima.

Pergunta-se:

Quantas linhas tem a imagem ? _____ e quantas

colunas ? _____

os riscos cinza estão nas linhas

_____, _____, _____ e _____,

e nas colunas

_____, _____, _____ e _____,



- 1 - /

vermelho = amarelo. A tabela de cores começa em X'36'. Preencha a tabela de cores:

Cor 0	Cor 1	Cor 2
Cor 3	Cor 4	Cor 5
Cor 6	Cor 7	Cor 8

A linha do pé da imagem começa em X'436'. Localize os bytes na memória. Pela análise desta linha é possível localizar a:

cor de fundo: _____.

Repare que cada linha tem apenas 18 bytes, mas que na memória elas ocupam 20 bytes (múltiplo de 32 bits ou 4 bytes)

Localize as colunas: e as linhas

Col _____,cor: _____ Lin _____,cor: _____
Col _____,cor: _____ Lin _____,cor: _____
Col _____,cor: _____ Lin _____,cor: _____
Col _____,cor: _____ Lin _____,cor: _____

Primeiro risco desenhado: L= _____ Ultimo risco C= _____

Respostas do exemplo: A imagem tem 18 colunas por 15 linhas. O fundo da mesma é da cor cinza. As colunas estão nas posições 3 (magenta), 7 (azul), 13 (branca) e 16 (verde). As linhas são: 2 (vermelho), 6 (ciano), 11 (amarelo) e 13 (preto). A primeira linha a ser desenhada foi a linha 11 e a última coluna foi a 3. Este arquivo está disponível com o nome de exemp753.bmp.

Para você fazer

1. Você verá abaixo as principais partes de um arquivo gráfico BMP que foi devidamente impresso aqui: Trata-se de uma imagem retangular, contendo 4 linhas verticais e 4 linhas horizontais de cores diversas, diferentes da cor de fundo.

```

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
    --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
0000  42 4D 8A 05 00 00 00 00 00 00 36 04 00 00 28 00 .|. |.....6....
0010  00 00 11 00 00 00 11 00 00 00 01 00 08 00 00 00 .....
0020  00 00 60 01 00 00 00 00 00 00 00 00 00 00 00 01 .....
0030  00 00 00 00 00 00 00 00 00 FF 00 FF 00 FF 00 00 FF .....
0040  00 00 FF 00 00 00 FF FF FF 00 00 FF FF 00 FF FF .....
0050  00 00 80 80 80 00 00 00 00 00 00 09 09 09 00 0A 0A .....
0060  0A 00 0B 0B 0B 00 0C 0C 0C 00 0D 0D 0D 00 0E 0E .....
0070  0E 00 0F 0F 0F 00 10 10 10 00 11 11 11 00 12 12 .....

...
0410  F6 00 F7 F7 F7 00 F8 F8 F8 00 F9 F9 F9 00 FA FA .....
0420  FA 00 FB FB FB 00 FC FC FC 00 FD FD FD 00 FE FE .....
0430  FE 00 FF FF FF 00 01 01 03 01 01 01 08 01 01 01 .....
0440  01 01 02 01 01 06 01 01 00 00 04 04 03 04 04 04 .....
0450  04 04 04 04 04 04 04 04 04 04 04 04 00 00 01 01 .....
0460  03 01 01 01 08 01 01 01 01 02 01 01 06 01 01 .....
0470  00 00 01 01 03 01 01 01 08 01 01 01 01 01 02 01 .....
0480  01 06 01 01 00 00 01 01 03 01 01 01 08 01 01 01 .....
0490  01 01 02 01 01 06 01 01 00 00 07 07 03 07 07 .....
04A0  08 07 07 07 07 07 07 07 06 07 07 00 00 01 01 .....
04B0  03 01 01 01 08 01 01 01 01 02 01 01 06 01 01 .....
04C0  00 00 01 01 03 01 01 01 08 01 01 01 01 01 02 01 .....
04D0  01 06 01 01 00 00 01 01 03 01 01 01 08 01 01 01 .....
04E0  01 01 02 01 01 06 01 01 00 00 01 01 03 01 01 01 .....
04F0  08 01 01 01 01 01 02 01 01 06 01 01 00 00 00 .....
0500  03 00 00 00 08 00 00 00 00 00 02 00 00 06 00 00 .....
0510  00 00 01 01 03 01 01 01 08 01 01 01 01 01 02 01 .....
0520  01 06 01 01 00 00 05 05 03 05 05 05 05 05 05 .....
0530  05 05 05 05 05 05 05 05 00 00 01 01 03 01 01 .....
0540  08 01 01 01 01 01 02 01 06 01 01 00 00 01 01 .....
0550  03 01 01 01 08 01 01 01 01 01 02 01 01 06 01 01 .....
0560  00 00 .....
0570 .....
0580  08 08 08 08 08 08 08 00 00 00 .....
```

Pergunta-se: Quantas colunas ? _____ quantas linhas ? _____. Qual o nome da cor de fundo ? _____. Qual o número da linha que foi desenhada primeiro? (informe o número da linha, NÃO a nome da cor) _____. Qual o número da coluna que foi desenhada por último? (idem) _____

2. Agora considere um arquivo semelhante a este, gravado em disco com o nome de YXAGE001.BMP. Neste segundo exercício, examine o arquivo BMP e a imagem gerada e responda:

Qual o nome da 6ª cor na tabela de cores ? _____ Qual o nome da 7ª cor na tabela de cores ? _____

Qual o número da linha que foi desenhada primeira ? _____ Qual o número da coluna que foi desenhada por último ? _____

Observação: No arquivo, a primeira linha começa em X'436'. Na imagem, esta linha corresponde à linha inferior. Ou seja, no arquivo a contagem começa de cima para baixo, e na imagem, de baixo para cima. Comece a contar em 1. Uma boa ferramenta para examinar o conteúdo hexadecimal do arquivo é o site <http://www.onlinehexeditor.com/>.



Acompanhe no exemplo: Quantidade de colunas _____ e Quantidade de linhas _____.

Relembrando: as cores estão na forma de misturas de AZUL / VERDE / VERMELHO. azul + vermelho = magenta; azul + verde = ciano; verde +

PAI: Convoluções (ponto, linha e borda)

Processamento de imagens: melhoria imagens para serem vistas por humanos. Análise de imagens: melhoria e tratamento de imagens para alguma decisão a ser tomada por um computador. Olho humano: esfera aquosa de 2 cm de diametro. A luz entra pela iris que mede entre 2 e 8 mm. Na retina há 7 milhões de cones (cor) e 100 milhoes de bastonetes (monocromático). O olho opera em ampla faixa na ordem de 10^{10} . (Não ao mesmo tempo, é claro).

Em termos computacionais uma imagem é uma matriz numérica. Cada número é $f(x,y)$ e o resultado de $i(x,y) \times r(x,y)$. i=iluminância e r=refletância. Alguns i: dia ensolarado=900, nublado=100, escritório=10, noite de lua cheia=0,01 lumens/m² ou lux. Alguns r: neve=0,93, parede branca=0,80, aço inox=0,65, veludo negro=0,01.

Imagens são formadas por valores no intervalo L_{min} até L_{max} . L_{min} vale 0 e L_{max} em geral é uma potência de 2. Imagens coloridas são formadas por 3 cores primárias aditivas.

Operações Aritméticas. Experimente pensar como ficaria o resultado de:

Para imagens Branco e Preto

- O complemento para L_{max} : _____.
- Multiplicação de cada valor por 2: _____.
E o excesso ? _____.
- Divisão por k: _____.
- Limiarização: _____.
- Arredondamento para a dezena mais próxima : _____.
- Duas imagens somadas pixel a pixel: _____.
- Subtração de 2 imagens: _____.
- AND em imagens com 2 níveis de cinza : _____.
- OR idem: _____.

Para imagens color

Os exemplos acima para 1, 2 ou 3 cores ao mesmo tempo. Alguns exemplos:

- Cor vermelha=0: _____.
- Cor verde $\times$ 2: _____.
- Os 3 pixels iguais à média entre os 3 valores : _____.
- A negativa do vermelho combinada com a azul e verde normais : _____.
- A negativa do verde e do azul combinada com anormal do vermelho : _____.
- um degradê vertical na verde combinada com as demais normais : _____.
- um degradê horizontal vermelho combinada com as demais normais : _____.
- degradê vertical verde + degradê horizontal vermelho + azul normal : _____.
- degradê CENTRAL do vermelho : _____.
- limiarização do azul no nível 100 : _____.
- limiarização do azul e do vermelho em 150 : _____.
- transformada do seno coxabranca aplicado ao verde : _____.
- Imagine mais algumas operações... (pode batizá-las à vontade)

Operações orientadas à vizinhança

Cada pixel tem o seu valor afetado pelo conteúdo dos pixels seus vizinhos. Por exemplo, imagine uma imagem onde cada pixel é substituído pela média dos seus 8 vizinhos: _____.

Para realizar estas operações em geral é usado o operador CONVOLUÇÃO. Neste operador uma máscara deslizante "passeia"por todos os pixels da imagem original, dando origem a uma nova imagem (convoluida).

Suponha a seguinte imagem original

a	b	c
d	e	f
g	h	i

Primeiro preenche-se as linhas e colunas externas à imagem com um valor qualquer. Depois, convolui-se a imagem. Se o preenchimento for com 1 a

matriz ficará:

1	1	1	1	1
1	a	b	c	1
1	d	e	f	1
1	g	h	i	1
1	1	1	1	1

Se a máscara for, por exemplo

2	2	2
3	3	3
1	1	1

 o novo pixel a' será = $a' =$

$2 \times 1 + 2 \times 1 + 2 \times 1 + 3 \times 1 + a \times 3 + b \times 3 + 1 + d + e \div 9$. e assim, sucessivamente para todos os pixels da imagem original. Note que como não há negativos na máscara, há que se dividir por 9

Para você fazer

- Considere uma imagem monocromática I de 4 linhas por 6 colunas formada por 16 níveis de cinza (0 a 15).
- Considere as seguintes máscaras:

$$M_1 \text{ (detetor de pontos isolados)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$M_2 \text{ (detetor de linhas horizontais)} = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

$$M_3 \text{ (detetor de linhas verticais)} = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

$$M_4 \text{ (detetor de Sobel vertical)} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$M_5 \text{ (detetor de Sobel horizontal)} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$M_6 \text{ (Laplaciano } 3 \times 3) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- Para efeito de convolução, imagine linhas adicionais encima e embaixo de I, com valores medios (7). Idem para as colunas.
- No cálculo, se o valor for negativo, considere resultado igual a 0. Se o resultado for fracionário, trunque. Se o valor for maior que 15, considere 15.
- Gere a matriz A convoluindo I com M_2
Gere a matriz B convoluindo I com M_3
Gere a matriz C convoluindo I com M_4

- Dada a matriz I

	1	2	3	4	5	6
1	7	9	7	8	7	8
2	5	9	5	5	5	5
3	7	9	7	8	7	5
4	6	9	6	8	6	6

- Calcule A, B e C e responda:

A[1; 2]	A[3; 1]	A[3; 5]	B[1; 6]	B[2; 4]	C[2; 1]

1



PAI: Equalização de histograma

Esta técnica está incluída nas técnicas de modificação do histograma. Elas alteram as cores da imagem modificando os valores de "tinta" de cada pixel. Antes de estudá-la precisa-se definir o que é histograma de uma imagem.

Histograma

Histograma de uma imagem é um conjunto de números que indicam quantos pixels apresentam determinado nível de cinza. Exemplo, seja uma imagem de 4 linhas x 10 colunas e 8 níveis de cinza

0	0	0	0	1	1	2	2	3	2
0	0	0	1	1	2	2	3	3	3
0	0	0	1	2	3	4	5	6	6
0	0	0	1	1	3	3	3	4	1

teria o seguinte histograma:

0	1	2	3	4	5	6	7

As vezes ele é apresentado como um gráfico e as vezes como % do total.

Transformações ponto-a-ponto

As transformações que usam o histograma são chamadas ponto-a-ponto pois só usam o valor original do pixel e não a sua vizinhança.

Equalização de histograma

A transformação mais conhecida é a chamada EQUALIZAÇÃO DE HISTOGRAMA que busca distribuir os valores de cada nível de cinza de maneira a ter um histograma uniforme. Para esta transformação usa-se uma função, que pode ser – por exemplo – a função de distribuição acumulada.

Esta transformação tem a habilidade de aumentar o contraste das imagens forçando o surgimento de detalhes importantes onde antes só havia manchas cinza. Pode também ser usada em imagens coloridas, aplicando-se o método a cada cor individualmente.



Figura 1: Imagem original Figura 2: e equalizada

Outras transformações parecidas

Distribuição linear de níveis de cinza

Uma vez obtido o intervalo de variação de níveis de cinza, o mesmo é esticado de maneira a abranger todo o espectro. O histograma muda de figura geométrica. Exemplo: Seja uma imagem 4 x 10 com 8 níveis de cinza (0..7)

0	0	0	0	1	1	2	2	3	2
0	0	0	1	1	2	2	3	3	3
0	0	0	1	2	3	3	3	2	1
0	0	0	1	1	3	3	3	2	1

Neste exemplo, a quantidade de níveis de cinza usados é: _____. Como a variação da imagem é de 8 níveis, pode-se aumentar o salto de cada nível, por um fator _____ melhorando o contraste de maneira linear.

Compressão de histograma

Usa o caminho inverso da equalização, e quando aplicada a uma imagem diminui-lhe o contraste.

Limiarização

Muito usada em análise de imagens (quando o computador tem que tomar uma decisão em relação ao que vê). Visa destacar o objeto de seu fundo. Por exemplo, ao examinar a qualidade de fabricação de uma peça, usando a limiarização, um programa poderia decidir sobre a área da peça (em pixels). Note que a técnica é necessária, por pois mais simples que seja a imagem não é possível trabalhar com valores absolutos de cinza (pode estar um dia claro ou um dia nublado, ser de manhã ou de noite...).

Por exemplo, a imagem 4 x 10, com 8 níveis:

0	0	0	0	1	1	2	2	3	2
0	0	0	1	1	2	2	3	3	3
0	0	0	1	2	3	3	3	2	1
0	0	0	1	1	3	3	3	2	1

Seria convertida na imagem

0	0	0	0	0	0	7	7	7	7
0	0	0	0	0	7	7	7	7	7
0	0	0	0	7	7	7	7	7	0
0	0	0	0	0	7	7	7	7	0

se fosse limiarizada com nível ≥ 2

Exemplo de equalização de histograma

Para estudar o exemplo, considere a imagem 2 x 7 com 10 níveis de cinza

1	2	3	2	2	3	1
1	1	3	2	2	1	1

1. Você deve construir o histograma: 0=0; 1=6; 2=5; 3=3; 4=0; 5=0; 6=0; 7=0; 8=0; 9=0

2. Construa a tabela de controle (nn x 6) (nn=níveis de cinza)

col 1 Escreva todos os valores de cinza (0 a 9)

col 2 Escreva o resultado do histograma na ordem em que foi construído

col 3 Escreva (coluna 1) dividido pelo maior nível de cinza (ou seja 9, neste caso)

col 4 Escreva (coluna 2) dividido pelo numero de pixels (ou 14)

col 5 Acumule a (coluna 4)

col 6 Escreva o indice da (coluna 3) que mais se aproxima da (coluna 5)

3. Finalmente, os pixels que tiverem valor (coluna 1) devem ser reescritos na imagem que esta sendo equalizada com o valor da (coluna 6).

(1) cinza	(2) hist	(3) $col_1 \div 9$	(4) $col_2 \div 14$	(5) col_5 acum	(6) novo valor
0	0	0.000	0.000	0.000	0
1	6	0.111	0.428	0.428	4
2	5	0.222	0.357	0.785	7
3	3	0.333	0.214	0.999	9
4	0	0.444	0.000	0.999	9
5	0	0.556	0.000	0.999	9
6	0	0.667	0.000	0.999	9
7	0	0.778	0.000	0.999	9
8	0	0.889	0.000	0.999	9
9	0	1.000	0.000	0.999	9

4. E com isto, a imagem original ficaria:

4	7	9	7	7	9	4
4	4	9	7	7	4	4

Para você fazer

[Uma boa referência: www.khoral.com]

Considere uma imagem IA representada por uma matriz 9 por 9 de pontos em 10 níveis de cinza (de 0 a 9). O divisor da coluna 4, será 81, neste caso.

	1	2	3	4	5	6	7	8	9
1	6	6	6	6	1	6	6	8	7
2	7	7	6	3	7	6	8	6	6
3	8	8	7	0	7	6	6	7	7
4	6	7	7	2	6	9	8	8	9
5	8	7	4	6	8	6	6	7	7
6	7	8	6	6	7	6	6	6	8
7	7	6	7	6	6	8	6	7	9
8	6	9	6	6	6	8	6	6	5
9	7	8	6	9	8	6	6	7	6

Aplique o algoritmo de equalização de histograma a IA gerando IB e responda:

IB[4; 1]	IB[6; 2]	IB[9; 6]	IB[7; 6]	IB[6; 1]	IB[7; 9]
5	8				

1



- 1 - /

O padrão GIF

Header: 6 bytes (obrigatorio)

0-2: GIF

3-5: 87a ou 89a

Descritor de tela logica: 7 bytes (obrigatorio)

0-1: largura da imagem em pixels (max=65536)

2-3: altura da imagem (max=idem)

4 bit 0: indicador de tabela cores global (0=nao, 1=sim)

bits 1-3: numero de bits por cor - 1(111=7 aqui significa 8 bits/cor)

bit 4: 1=tabela global classificada, 0=nao

bits 5-7: tamanho da tabela (2^esta valor+1) entradas

5: indice da cor de background

6: razao da altura [entre 1:4 e 4:1 = (valor+15)/64]

Se tiver tabela global de cores, vem a seguir (tamanho entre 6 e 768 b)

Descritor de imagem: tamanho 10 bytes (obrigatorio ao menos 1)

identificador 2C

byte 0: 2C

1-2: deslocamento lateral desta imagem

3-4: deslocamento vertical ...

5-6: largura desta imagem

7-8: altura desta imagem

byte 9 - bit 0: 0=nao ha tabela local de cores; 1=sim

bit 1: 1=imagem entrelacada, 0=nao

bit 2: 1=tabela local ordenada, 0=nao

3-4: reservado

5-7: tamanho da tabela local de cores

Dados: primeiro byte: numero inicial de bis usados pelo LZW

Demais: sub-blocos de dados. Cada um comeca por byte de tamanho e termina por 00.

Outros Blocos

21F9 = bloco de controle

21FE = bloco de comentario

2101 = bloco de texto plano

21FF = bloco de aplicacao

3B = terminador de fluxo

21F9 - Bloco de controle:

byte 0: tamanho

byte 1-bits 0-2: reservado (zeros)

bits 3-5: metodo de substituicao (0=nada, 1=nao ha substituicao;

2=area retornada a cor de background, 3=voltar ao que havia antes;

4 a 7:uso futuro)

bit 6: 1=aguardar acao do operador, 0=nao aguardar

bit 7: 0=nao ha cor transparente 1=ha cor

byte 2-3: retardo em centesimos de segundo

byte 4: cor transparente (quando ha)

byte 5: 00 (terminador)



Para você fazer

Examine o arquivo GIF gimage01.gif

Localize os blocos de controle e responda: 4 primeiros números do vetor de retardos:

O tamanho da sub-imagem número 3 é

largura	altura

O tamanho da sub-imagem número 7 é

largura	altura



- 1 - /

Padrão FITS

FITS é uma sigla que significa *Flexible Image Transport System* e é um padrão aberto que define um formato de arquivo digital usado para armazenar, transmitir e processar imagens (originalmente) científicas. O padrão nasceu na área de astronomia justamente para permitir o intercâmbio de imagens digitais obtidas em telescópios de todas as naturezas possíveis. O arquivo é suficientemente genérico para acomodar muitos dados científicos além da imagem em si. Por exemplo, é comum haver dados de fotometria, informações de calibração, filtros usados além de muitas informações sobre a imagem em si, o que chamamos metadados. O padrão FITS surgiu em 1981 e evoluiu gradualmente desde então, a versão mais recente é a 4.0, adotada em julho de 2016. O mote do FITS é *once FITS, always FITS* e ele tem a pretensão de ser legível e manuseável mesmo depois que o software e o hardware usados para criar a imagem original não existam mais. O formato ganhou visibilidade quando foi escolhido pela Biblioteca do Vaticano para disponibilizar imagens de documentos daquele lugar, talvez a mais importante biblioteca de documentos antigos do mundo, repleta de manuscritos dos séculos V, VIII, XI e assim por diante. Suas principais características:

Não proprietário O formato FITS é mantido pela comunidade científica internacional, em particular a comunidade de astrônomos e astrofísicos e como tal ele não é formato proprietário. Atualmente a especificação é mantida pelo Grupo de Trabalho FITS da IAU, União Internacional de Astronomia.

Bem documentado O formato FITS está completamente documentado na Referência FITS, livremente disponível. A versão 3 foi publicada no *Astronomy and Astrophysics* volume 524 de dezembro de 2010.

Largamente adotado O formato é comumente usado pelos astrônomos. Há muitas aplicações de software para criar e ver os arquivos, além de permitir a importação e exportação em outros formatos também usuais.

Transparência Uma grande vantagem é a simplicidade do formato. Nada é comprimido (o que introduziria grande risco em caso da corrupção de um único bit...) e os dados são guardados ora em ASCII puro, ora em formatos de codificação padronizados.

Auto contido Todas as informações necessárias para representar corretamente a imagem (na tela ou na impressora) estão incluídas dentro do arquivo. Links externos ou fontes de recursos externos não são permitidos.

Independência de dispositivo Não há nenhuma dependência a algum hardware ou software específicos em nenhum ponto do arquivo. Esta característica é muito importante na compatibilidade para a frente, pois não temos idéia do que vem por aí.

Formato

O arquivo é formado por conjuntos de header + dados, tantos quantos houver interesse. O primeiro conjunto é chamado primário e é obrigatório. Quando só tem o primário o arquivo é conhecido como Arquivo FITS básico e quando tem uma ou mais extensões seguindo o conjunto primário é chamado de Arquivo FITS multi-extensão. Cada conjunto (chamado na documentação de *FITS structure*) é formado por um número inteiro de blocos FITS, cada um com 2880 bytes e escrito em ASCII puro (2880 = 80 × 36). Todos estes blocos ficam juntos, antes dos dados. Por ASCII puro entende-se caracteres ocupando as posições 32 até 126 do código ASCII.

O bloco de header contém uma série de palavras chaves seguida de seu valor. Assim, cada header tem espaço para 36 palavras chave. O último bloco de header deve conter a palavra chave

END que marca o fim lógico do header. O que sobrar de espaço depois, deve ser preenchido com espaços. O conjunto de dados que vem depois, (se presente) deve consistir de um arranjo (array) de dados de 1 a 999 dimensões (isto é definido pela palavra chave NAXIS. O array inteiro é representado por um stream contínuo de bits que começa no primeiro bit do primeiro bloco de dados. Cada dado consiste de um número fixo de bits como determinado na palavra chave BITPIX. O índice do eixo 1 é o que varia mais rapidamente, depois o do eixo 2, e assim por diante. O último eixo (NAXIS) é o que varia mais lentamente. Não há espaço ou qualquer caractere especial entre o último valor em uma linha ou plano e o primeiro valor da próxima linha ou plano. A contagem começa em 1. Veja na figura como é isso:

```
A(1, 1, ..., 1),
A(2, 1, ..., 1),
...,
A(NAXIS1, 1, ..., 1),
A(1, 2, ..., 1),
A(2, 2, ..., 1),
...,
A(NAXIS1, 2, ..., 1),
...,
A(1, NAXIS2, ..., NAXISM),
...,
A(NAXIS1, NAXIS2, ..., NAXISM)
```

Se o bloco de dados não preenche o último bloco ele deve ser preenchido com bits zero. O bloco de header é formado por registros de 80 caracteres (36 por bloco) e eles consistem de uma palavra chave, um indicador de valor (se necessário), um valor opcional e um comentário opcional. As palavras podem aparecer em qualquer ordem, com poucas exceções. A palavra chave deve ter até 8 posições, alinhada à esquerda e com brancos à direita, sempre em letras maiúsculas. As posições 9 e 10 podem ter “=” ou “_”. A seguir o valor da palavra chave e opcionalmente um comentário. Entre o valor e o comentário deve-se escrever “_” ou “_”. Em todo este conteúdo (inclusive comentários) deve-se usar apenas os caracteres ASCII de 32 até 126 – hexadecimal 20 até 7E. Variáveis de valor lógico devem ser representadas por T ou F. Eis as principais palavras chave:

```
SIMPLE Deve conter um valor verdade (T) se este
arquivo obedece ao padrão FITS.

BITPIX Contém um inteiro e indica o número de
bits que representa cada valor no array de
dados. Os valores possíveis são: 8, 16, 32,
64 – para inteiros – e -32 ou -64 para ponto
flutuante.

NAXIS Inteiro não negativo não maior que 999 indi-
cando quantos eixos tem o array de dados.

NAXISn Esta palavra deve estar presente para to-
dos os valores de n de 1 até NAXIS. Indica o
tamanho da dimensão do array de dados.

END Indica final do cabeçalho. Esta palavra chave
não tem valor associado

Veja-se um exemplo disto:

SIMPLE = T / file does conform to FITS standard
BITPIX = 16 / number of bits per data pixel
NAXIS = 2 / number of data axes
NAXIS1 = 250 / length of data axis 1
NAXIS2 = 300 / length of data axis 2
OBJECT = 'Cygnus X-1'
DATE = '2008-10-27'
END
```

O número total de bits do array de dados primário, sem contar o preenchimento eventualmente neces- sário para completar o bloco de 2880 caracteres, é $N_{bits} = |BITPIX| \times (NAXIS1 \times NAXIS2 \times \dots \times NAXISM)$

A cor no arquivo FITS

Paradoxalmente, a cor não tem um significado fixo no arquivo FITS. Ela pode (deve) ser livre- mente interpretada numa negociação entre grava- dor e leitor. O formato lava as mãos sobre isso. Para imagens coloridas do uso no nosso dia a dia, pode-se fazer a seguinte combinação: true color (8 bits=0..255) por canal R, G, B e mais um quarto canal de transparência. Total 32 bits por pixel.

Neste caso, pode-se usar o inteiro de 32 bits no re- gistro do arquivo. Se uma maior precisão de cor é exigida (como por exemplo na captura de pergami- nhos e iluminuras da Biblioteca Vaticana), pode-se fazer: 16 bits por canal R, G ou B chegando por- tanto a 65.536 níveis em cada cor, e mais um can- al de transparência com o mesmo nível de valo- res. Total 64 bits por pixel, usando-se neste caso os inteiros de 64 bits. Ou pode-se definir qualquer outra combinação, eventualmente usando-se outros esquemas de cor (como CMYB, por exemplo). O que resta importante no formato, é que cada valor sempre tem uma configuração que é múltipla de 8 bits, ou seja não há aqui os problemas de trunca- mento, padding ou stuffing bits de outros formatos.

Como ver arquivos FITS

Existe uma grande variedade de visualizadores de arquivos FITS, e como tudo no padrão, são livres para download e instalação. Um programa que é bem completo e funciona inclusive como editor para este tipo de arquivo é o FV, que tem versões para plataformas windows, unix e Mac. Seu site é <https://heasarc.gsfc.nasa.gov/ftools/fv/>. Vale uma visita. Outro é DS9, que pode ser baixado em <http://ds9.si.edu/site/Home.html>. Especial referência deve ser feita em relação à linguagem Python. Usando a vocação desse ambi- ente, (106.000 pacotes em mar/17), há um pacote completo chamado ASTROPY que dá toda a funci- onalidade para operar magnificamente com arqui- vos FITS. Mais detalhes em www.astropy.org, onde há um tutorial muito bom.

Para você fazer

- Você vai receber o arquivo de nome `im760-01.fits`
- Deve analisar seu conteúdo usando pelo menos 3 ferramentas:
- Um editor ASCII convencional (NOTE- PAD++ por exemplo) para estudar o bloco de header
 - Um editor de formato binário por exemplo wxHexEditor ou então o site <http://www.onlinehexeditor.com/>. Este último funciona online e não exige instala- ção de software. Aqui examina-se os blocos de dados do arquivo FITS
 - Um visualizador de arquivos FITS como o FV ou DS9 que permitem examinar com- pletamente o arquivo

Responda a seguir, as seguintes questões:

1. Quantas elementos tem o arquivo na 1ª di- mensão ?
2. E na 2ª dimensão ?
3. Quantos blocos de header+dados no ar- quivo ?
4. Qual o maior nível de cinza na imagem ?
5. Qual o menor nível de cinza na imagem ?

1	2	3	4	5



Redes de Petri tipo condição/evento

Uma rede de Petri é uma representação matemática para sistemas distribuídos discretos. Inventadas pelo Dr Carl Petri como parte de sua tese de doutorado na Alemanha.

Para a utilização das Redes de Petri do tipo mais simples (C/E), precisamos conhecer quais os elementos que serão modelados, isto é, como traduzir os sistemas em estudo em entidades próprias das redes. Uma Rede de Petri do tipo Evento/Condição é constituída de 4 elementos, a saber:

- Um conjunto de condições, que são estados possíveis do sistema em estudo.
- Um conjunto de eventos, que ocorrem dentro do sistema (e que podem modificar os estados do sistema).
- Uma função de entrada que relaciona condições e eventos, mapeando para cada condição, um conjunto de eventos.
- Uma função de saída que também relaciona os eventos e as condições e que para cada evento mapeia um conjunto de condições.

Matematicamente falando, portanto, uma Rede de Petri R é uma quádrupla $R = (C, V, E, S)$, onde C é o conjunto de condições, V o de eventos, E a função de entrada e S a função de saída.

Diz-se que uma determinada condição C_k é uma condição de entrada para um dado evento V_j , se $C_k \in E(V_j)$. Identicamente, diz-se que uma condição C_m é uma condição de saída para um dado evento V_n , se $C_m \in S(V_n)$.

Trocando isso em miúdos, uma Rede de Petri será um conjunto de condições e eventos, sendo que a cada evento estarão associados dois conjuntos de condições. O de entrada e o de saída.

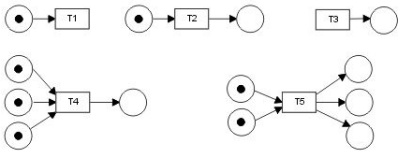
As condições C, serão representados por círculos. Os eventos V, por meio de retângulos. Os mapeamentos da função de entrada serão representados por meio de arcos orientados saindo das condições e entrando nos eventos. Os mapeamentos da função de saída, por meio de arcos orientados saindo dos eventos e entrando nas condições.

Marcação

O conceito de marca é outra primitiva das Redes de Petri (assim como condições e eventos). Marcas são colocadas nas condições e permitem o estudo dinâmico dos modelos. Matematicamente falando, a marcação é uma função que relaciona as condições a um conjunto de números 0 ou 1. Por exemplo, um determinado conjunto de condições $P = \{A, B, C, D, E\}$ pode ter associada a marcação $\{0,0,1,1,0\}$, significando que no instante inicial os condições A, B e E não terão marcas, o C e o D terão uma marca. A definição deve ser entendida como A, B e E tendo valor "falso" e C e D tendo valor verdadeiro. Esta função pode receber o nome de μ , e portanto a definição completa de uma Rede de Petri do tipo C/E, passa a ser a quádrupla $R = (C, V, E, S, \mu)$. Da marcação de uma Rede de Petri, segue-se o conceito de *execução* da rede. Que nada mais vem a ser do que o comportamento dinâmico da rede. A rede passa a ter "vida" quando disparos passam a ocorrer na rede. As marcas controlam os disparos, através de uma regra bem simples:

Numa rede C/E, um evento está habilitado a ocorrer se todas as condições de entrada contam com marcas E todos os condições de saída não tem marca. Ou dito de outra maneira, se todas as suas condições de entrada são verdadeiras e todas as suas condições de saída são falsas.

Quando um evento está habilitado ele dispara, e entende-se pelo disparo a retirada das marcas da entrada e a colocação de marcas nas saídas. Note-se que não necessita haver conservação de marcas. Vejamos os exemplos, na figura abaixo.



Note-se que T_1 está habilitado, uma vez que sua entrada está com marca e ela não tem saída. T_2 tem sua entrada marcada e sua saída livre. T_3 não tem saídas marcadas, T_4 e T_5 tem todas as entradas marcadas e nenhuma saída marcada. Todos os casos obedecem à lei da habilitação que é: *todas as entradas devem estar marcadas e nenhuma saída pode estar marcada*. A ocorrência de cada um dos eventos retira TODAS as marcas de entrada e coloca marcas em TODAS as saídas.

Suponha, como exemplo:

Pré-condição () -> []

(1)	1	0	0	1	0	0	1	0	0	0	0
(2)	0	0	0	0	0	0	0	0	0	0	1
(3)	0	1	0	0	0	0	0	0	0	0	0
(4)	0	1	0	0	0	0	0	0	0	0	1
(5)	0	0	0	0	0	0	0	0	0	0	0
(6)	0	0	0	0	0	0	0	0	0	0	1
(7)	1	0	0	0	0	0	0	0	0	0	0
(8)	0	0	0	0	0	1	0	0	0	0	0
(9)	0	0	0	1	1	0	0	0	0	0	0

Pós-condição [] -> ()

(1)	0	0	0	0	0	0	0	0	0	0	0
(2)	0	0	0	0	0	0	0	0	0	0	0
(3)	0	0	0	0	0	0	0	0	0	1	0
(4)	0	0	0	0	1	0	0	0	0	0	0
(5)	0	0	0	0	0	0	0	0	0	0	0
(6)	0	0	0	1	0	1	0	1	0	0	0
(7)	0	0	0	0	0	1	1	0	1	1	1
(8)	0	0	0	0	0	0	0	0	0	0	0
(9)	0	0	0	0	0	1	0	0	0	0	1

Para este exemplo, as respostas serão:

1. Divisão: 6, 10 e 11.
2. Junção: 1, 2, 4 e 11.
3. Iniciam caminhos alternativos: 1, 4 e 9.
4. Encerram caminhos alternativos: 6, 7 e 9.
5. Eventos habilitados: 3 e 8.

Para você fazer

Imagine uma Rede de Petri de evento/condição, definida pelas matrizes

Pré-condição () -> []

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
(1)	0	1	0	0	0	1	0	1	0	0	1
(2)	0	1	0	0	0	0	0	0	0	0	0
(3)	0	0	0	0	0	0	0	0	0	0	0
(4)	0	0	0	0	0	0	1	0	0	0	0
(5)	0	0	0	0	0	0	0	1	0	0	0
(6)	0	0	0	0	1	0	0	0	0	0	0
(7)	0	1	1	0	0	0	0	0	1	1	0
(8)	0	0	0	0	0	0	0	0	0	0	0
(9)	0	0	0	0	0	0	0	0	0	0	0

Pós-condição [] -> ()

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
(1)	0	0	0	0	0	0	0	0	0	0	0
(2)	0	0	1	0	0	0	0	0	0	0	0
(3)	0	0	1	0	0	0	0	0	0	1	1
(4)	0	0	0	1	0	0	0	1	0	0	1
(5)	0	0	0	0	0	1	0	0	0	0	0
(6)	0	0	0	0	0	0	1	1	0	0	0
(7)	0	0	0	0	0	0	0	0	0	0	0
(8)	0	0	0	1	0	1	0	0	0	0	0
(9)	0	0	0	0	0	0	0	0	0	0	0

Desenhe a Rede de Petri e responda:

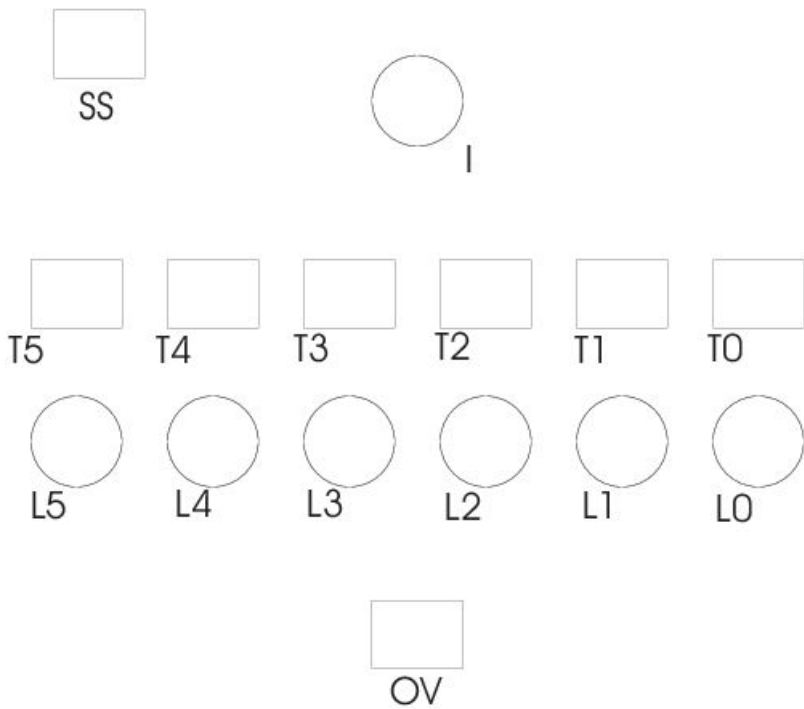
1. Quais os eventos que dividem o fluxo ? (um evento coloca 2 ou mais marcas):_____.
2. Quais os eventos que juntam fluxos ? (um evento retira 2 ou mais marcas):_____.
3. Quais as condições que iniciam caminhos alternativos ? (uma mesma condição é pré-condição de dois ou mais eventos):_____.
4. Quais as condições que encerram caminhos alternativos ? (uma única condição é pós-condição de dois ou mais eventos):_____.
5. Supondo a marcação: 0 0 0 1 0 1 1 1 1 1 , quais os eventos habilitados ? (pré-condições marcadas e pós-condições desmarcadas):_____.



- 1 - /

O que faz esta Rede de Petri?

Preencha as pré e pós condições na rede abaixo, de acordo com as matrizes de pré e pós condições e estude o funcionamento desta rede.



Preencha na rede acima, as seguintes matrizes

Pré-condições ()→[]								
	SS	T ₀	T ₁	T ₂	T ₃	T ₄	T ₅	OV
I	0	1	1	1	1	1	1	1
L ₀	0	0	1	1	1	1	1	1
L ₁	0	0	0	1	1	1	1	1
L ₂	0	0	0	0	1	1	1	1
L ₃	0	0	0	0	0	1	1	1
L ₄	0	0	0	0	0	0	1	1
L ₅	0	0	0	0	0	0	0	1

Pós-condições []→()								
	SS	T ₀	T ₁	T ₂	T ₃	T ₄	T ₅	OV
I	1	0	0	0	0	0	0	0
L ₀	0	1	0	0	0	0	0	0
L ₁	0	0	1	0	0	0	0	0
L ₂	0	0	0	1	0	0	0	0
L ₃	0	0	0	0	1	0	0	0
L ₄	0	0	0	0	0	1	0	0
L ₅	0	0	0	0	0	0	1	0

👉 Para você fazer

1. Para a marcação: $L_5=1, L_4=1, L_3=1, L_2=1, L_1=0, L_0=1, I=1$,
que evento(s) estará(ão) habilitado(s) ? _____.
- Obs: um evento está habilitado se TODAS as suas pré-condições estão marcadas e TODAS as pós-condições estão desmarcadas.
2. Para a marcação: $L_5=1, L_4=0, L_3=1, L_2=1, L_1=0, L_0=0$, Qual a marcação que existirá na rede após 8 disparos de SS ?
 L_5 :_____ L_4 :_____ L_3 :_____ L_2 :_____ L_1 :_____ L_0 :_____ I: 0
3. Para a marcação: $L_5=1, L_4=1, L_3=1, L_2=1, L_1=1, L_0=1$, Qual a marcação que existirá na rede após 7 disparos de SS ?
 L_5 :_____ L_4 :_____ L_3 :_____ L_2 :_____ L_1 :_____ L_0 :_____ I: 0
4. Para a marcação: $L_5=1, L_4=0, L_3=1, L_2=0, L_1=0, L_0=1$, Quantos disparos de SS são necessários para que todas as condições incluindo I, sejam verdadeiras ?
(estejam preenchidas ?) :_____.

Duas notícias verídicas

Antes de entrar no exercício de hoje, vejam-se duas notícias reais:

Estudante passa 5 horas preso em caixa eletrônico de São Carlos

Publicada em 04/09/2008 às 09h23m, via EPTV, Bom Dia São Paulo.
SÃO PAULO - O estudante Bruno Oliveira, de 18 anos, ficou cinco horas trancado em um caixa eletrônico no centro de São Carlos, a 229 quilômetros da capital, na madrugada desta quinta-feira. O jovem tentava fazer um depósito na agência, que é interligada à rede 24 horas, mas nenhuma máquina estava funcionando. Quando tentou deixar a agência, a porta travou e as luzes apagaram.
Oliveira só foi liberado cinco horas depois, quando o gerente do banco foi encontrado.
Segundo o estudante, além de não poder usar o banheiro, ele comeu somente biscoitos que foram passados por baixo da porta e bebeu refrigerante por um canudinho, pela fresta do vidro. Ele afirmou que pretende processar o banco.

Homem fica 8 horas preso em caixa eletrônico no DF

BRASÍLIA - Um homem ficou trancado durante oito horas em um caixa eletrônico no fim de semana, em Ceilândia, no Distrito Federal. De acordo com a Polícia Civil, Manoel Neto foi sacar dinheiro no sábado e, às 22h, as portas do banco fecharam automaticamente e ele não pôde sair. Os bombeiros foram chamados e o retiraram do local às 6h. Manoel foi levado para o Hospital Regional de Ceilândia com dor de cabeça e liberado em seguida.
Publicado no www.oglobo.com.br dia 22/09/2008.

Uma ATM modelada por Petri

Imagine uma máquina ATM (automatic teller machine = banco 24 horas) devidamente modelada por uma rede de Petri do tipo evento-condição como segue

Condições: ()

- P₁ ATM pronta
- P₂ cartão sendo identificado no sensor da porta
- P₃ ATM vazia
- P₄ cliente entrando na ATM
- P₅ cliente dentro da ATM
- P₆ cartão magnético identificado
- P₇ senha foi digitada
- P₈ luz vermelha externa acesa
- P₉ cliente identificado
- P₁₀ saldo sendo consultado no banco
- P₁₁ dinheiro sendo contado pela ATM
- P₁₂ cliente aguardando
- P₁₃ cliente pronto para sair

Eventos: []

- T₁ chega um cliente
- T₂ cartão do cliente é bom
- T₃ cartão do cliente não é bom
- T₄ ATM destrava a porta
- T₅ ATM trava a porta
- T₆ ATM engole o cartão
- T₇ ATM pede a senha da Conta Corrente
- T₈ Banco disse: a CC é válida, com saldo positivo
- T₉ Cliente digita o valor da retirada
- T₁₀ Saldo da conta é maior do que a retirada
- T₁₁ ATM autoriza o pagamento
- T₁₂ Banco disse: CC sem saldo
- T₁₃ ATM libera o cartão
- T₁₄ Banco disse: Retenha o cartão
- T₁₅ Retirada maior do que o saldo
- T₁₆ ATM destrava a porta
- T₁₇ O dinheiro da ATM é insuficiente

Pré-condições: () → []

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P1	1																
P2		1	1														
P3				1													
P4					1												
P5						1											
P6							1										
P7								1				1		1			
P8																1	
P9									1								
10										1					1		
11											1						1
12												1					
13																1	

Pós-condições: [] → ()

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P1			1													1	
P2	1																
P3		1															
P4				1													
P5					1												
P6						1											
P7							1										
P8					1												
P9								1							1		
10									1								
11										1							
12											1	1					1
13													1	1			

Observação: Os zeros não foram impressos para despoluir as matrizes



Para você fazer

- Desenhe a Rede de Petri em folha em branco, PARA SER ENTREGUE JUNTO COM A RESPOSTA AO EXERCICIO PEDIDO. Identifique o que é cada evento e cada condição.
- Supondo a marcação $M=(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$, indique qual o(s) próximo(s) eventos que fica(m) habilitado(s). Responda na folha.
- Indique com caneta de OUTRA COR, onde incluiria a condição CÂMERA DE TV EXTERNA LIGADA (para filmar possíveis encrencas enquanto tem alguém dentro da ATM). Desconsidere a câmera para responder a 2.
- Não esqueça de pôr seu NOME e o número 1 0 0 0 0 0 0 0 0 0 0 0 0 na folha



- 1 - /

Compressão de dados

A compressão de dados visa diminuir a redundância de dados. Logo apenas conjuntos de dados que tenham redundância podem ser comprimidos com vantagem. A compressão pode se dar em camadas: a. aplicativo: o aplicativo controla. Exemplo: uma tabela de cidades; b. do ambiente: o sistema operacional aplica. Para o aplicativo nenhuma mudança surge. Exemplo: PKZIP; c. Física: o hardware comprime. Exemplo: compressão feita pelo modem.

Os benefícios esperados são: a. diminuição de espaço e de tráfego. b. criptografia; c. melhoria de uso da CPU (já que os aplicativos em geral são IO bound); d. ultrapassar limites de mídias (exemplo um DVD). Certos arquivos são altamente propícios à compressão, por causa do fenômeno de "localidade de referência". Exemplos: música digital, vídeo...

Conceitos: Razão de compressão: tamanho descomprimido ÷ tamanho comprimido. Figura de mérito: o inverso da razão de compressão. Fração de redução: 1 - figura de mérito.

Prática Localize um arquivo médio de texto:

Nome: _____. Tamanho: _____. Localize um compressor (ex: PKZIP) Faça a compressão rápida (pkzip: -es).

Tempo: _____, Tamanho: _____,

Razão: _____, Fração redução: _____.

Faça a compressão máxima (pkzip: -ex).

Tempo: _____, Tamanho: _____,

Razão: _____, Fração redução: _____.

Localize uma imagem BMP:

Nome: _____. Tamanho: _____.

Faça a compressão rápida (pkzip: -es).

Tempo: _____, Tamanho: _____,

Razão: _____, Fração redução: _____.

Faça a compressão máxima (pkzip: -ex).

Tempo: _____, Tamanho: _____,

Razão: _____, Fração redução: _____.

Técnicas Usuais

Código Morse Só aparece aqui por razões históricas. O código morse é um sistema de representação de letras, números e sinais de pontuação através de um sinal codificado enviado intermitentemente. Foi desenvolvido por Samuel Morse e Alfred Vail em 1835, criadores do telégrafo elétrico (importante meio de comunicação à distância), dispositivo que utiliza correntes elétricas para emissão ou recepção de sinais. Veio na continuação do telégrafo visual em uso na França desde o século XVIII.

Uma mensagem codificada em Morse pode ser transmitida de várias maneiras em pulsos (ou tons) curtos e longos: * pulsos elétricos transmitidos em um cabo; * ondas mecânicas (perturbações sonoras); * sinais visuais (luzes acendendo e apagando); * ondas eletromagnéticas (sinais de rádio);

Este sistema representa letras, números e sinais de pontuação apenas com uma sequência de pontos, traços, e espaços.

Com o desenvolvimento de tecnologias de comunicação mais avançadas, o uso do código morse é agora um pouco obsoleto, embora ainda seja empregado em algumas finalidades específicas, incluindo rádio faróis, e por CW (continuous wave-ondas contínuas), operadores de radioamadorismo. Código morse é o único modo de modulação feito para ser facilmente compreendido por humanos sem ajuda de um computador, tornando-o apropriado para mandar dados digitais em canais de voz.

Os sinais são:

1. Sinal curto, ponto ou 'dit' (.) 2. Sinal longo, traço ou 'dah' (-) 3. Intervalo entre caracteres (entre pontos e traços) 4. Intervalo curto (entre letras) 5. Intervalo médio (entre palavras) 6. Intervalo longo (entre sentenças)

Como o comprimento variável de caracteres do código morse dificulta a adaptação à comunicação automatizada, o Morse foi amplamente substituído por formatos mais regulares, incluindo o Código Baudot e ASCII.

Para construir a tabela morse, usa-se a busca dicotômica na sequência, com ponto à esquerda e traço à direita. E,T,I,A,N,M,S,U,R,W,D,K,G,O,H,V,F,Ü,L,Ä,P,J,B,X,C,Y,Z,Q,...

Exemplo Traduza para o português:

...-1..1...-12.1...-12. Resposta: _____.

Comprimento de fileira Escolhe-se um caracter pouco frequente (no nosso exemplo será Z). Sempre que um conjunto de mais de 3 caracteres se repetir, todos eles serão substituídos pelo conjunto Z+carater a transmitir +quantidade de repetição.

No exemplo A.....DIVIDA.....E.DE.0000000033.REAIS, ficaria AZ.5DIVIDAZ.7E.DE.Z0833.REAIS

Observações: 1.naturalmente, se o caracter de contagem for um byte binário, o contador poderá chegar até 258, pois ele pode começar em 4, neste caso, reservando-se o '0' para a ocorrência simples do caracter.

Se por acaso, o caractere de controle existir no arquivo, ele pode aparecer na saída como duplicado, com tamanho 1.

Exemplo: A.....LUZ.E.BRANCA ficaria AZ.7LUZZ1.E.BRANCA

Exemplo Comprima usando esta técnica:

AAABBBBBB0000000574-REAIS. Resposta: _____.

Exemplo Traduza para o português:

EU\~6QUE\R80-\0672. Resposta: _____.

Substituição de padrões A substituição de padrões é –em geral – um método de compressão de aplicativo, já que é dependente dos padrões da aplicação. É muito usada em compressão de códigos fonte (os padrões são as palavras reservadas) ou em grandes sistemas de dados comerciais (nomes de pessoas, de ruas, de cidades, de conjuntos de zeros, CEPs, etc).

Nestes casos há uma tabela com os padrões mais frequentes (256, 65536,...) que são convertidos para 1 ou 2 bytes. Se o arquivo for binário, não há este limite, pode-se usar qualquer tamanho de tabela de padrões (2^9 , 2^{10} , 2^{11} ...). Apenas há que se sinalizar a existência do padrão comprimido. Pode ser o sempre presente caracter de controle acima visto.

Huffman Este método já foi estudado lá em estruturas de dados. Continua o mesmo. A codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo. Ele foi desenvolvido em 1952 por David A. Huffman que era, na época, estudante de doutorado no MIT, e foi publicado no artigo "A Method for the Construction of Minimum-Redundancy Codes".

Supondo o dicionário:

P0010000	U0100
V0010001	M0101
C001001	D0011
Z01100	E0000
N01101	R0001
G00101	0111
B1100	.100
I1101	A101
L0111	

Exemplo Descomprima:

26 98 68 D8 B4 (38 b.s.) Resposta: _____.



Para você fazer Traduza para português, as mensagens morse

1)
.-1--.-1---1...1-1---12
2)
-..1..1...1-.-.1---1...12
3)
.-1...1.-.1---1.-.1.-12

Comprima as frases a seguir, usando o método de comprimento de fileira. Use a \ como caracter de controle.

4)
000000100-----0000000200----111111222
5)
-----VIVA-----EU----E---TU-----TATU--

Descomprima as frases a seguir, usando o método do comprimento de fileira. O caractere de controle usado é a \.

6)
\~8SALVE~100-REAIS~-8\H5~-4
7)
0\~6VALOR~-7E-\08\34~-7

Descomprima:

8)
CE F6 72 35 E0 37 bits significativos
9)
3E 32 95 0E C0 36 bits significativos
10)
20 06 3E 67 90 38 bits significativos

RESPONDA NO VERSO 1

Compressão de dados - parte 2

Codificação Relativa Este tipo de compressão se aplica apenas em casos particulares, mas quando se aplica ele é muito bom. Novamente é uma aplicação do conceito de localidade de referência. Enviado (ou gravado) o registro inicial, para os subseqüentes serão enviadas apenas as alterações em relação ao elemento anterior. Não se pode colocar aqui um esquema geral, fica apenas o conceito. Imagine-se por exemplo uma aplicação que faz captura remota de dados de telemetria. Este esquema é usado no compactador campeão que é da Intel e comprime imagens em movimento, alcançando razão de compressão de 350:1.

Compressão de Shannon-Fano Similarmente ao código Huffman, esta compressão gera cadeias de bits de comprimento variável para cada caracter. Para sua implementação devemos conhecer o arquivo a transmitir, e a partir dele, construir a tabela de frequência de caracteres, em ordem decrescente.

Supondo, o seguinte conjunto FEEDDDCCBBBBAAAAAA. Isto geraria as seguintes frequências:

A=30%	B=25%	C=15%	D=15%	E=10%	F=5%

Agora, o conjunto é dividido em duas partes, tão logo a soma acumulada de probabilidades **ultrapassa** 50%. Ficaria

A=30%	B=25%	C=15%	D=15%	E=10%	F=5%

Atribui-se a primeira metade, o dígito 0 e à segunda, o dígito 1. Fica:

A=30%	B=25%	C=15%	D=15%	E=10%	F=5%
0	0	1	1	1	1

Procede-se igualmente na segunda metade, e assim por diante, ficando:

A=30%	B=25%	C=15%	D=15%	E=10%	F=5%
0	0	1	1	1	1
		0	0	1	1
				0	1

E temos então:

A = 00	B = 01	C = 100	D = 101	E = 110	F = 111

A tabela do processo Shannon-Fano pode ser ambígua, já que dificilmente teremos uma divisão em 50% exatos. Em alguns casos sua eficiência será igual a de Huffman (que é a eficiência ótima), podendo entretanto ser pior.

Exemplo Descomprima 85 71, com 16 bits significativos e universo='LLLLLAAAOOOHH.'. Resposta: _____. O dicionário: A=00, B=01, C=100, D=101, E=110, F=111.

Código de vírgula Tanto no código Huffman, quanto no Shannon-Fano, temos uma dificuldade adicional, principalmente em transmissão remota de arquivos. Se um único bit for perdido no meio do string, perde-se toda a recuperação posterior do arquivo. Diz-se neste caso que foi-se o sincronismo entre o código e o processo. Para evitar este inconveniente, pode-se utilizar um terminador de caractere, (usualmente o bit 1 isoladamente). Naturalmente este código não seria fornecido a nenhum caracter, e com isto perde-se eficiência na compressão, que no entanto é compensada pela possibilidade de perda de bits sem a perda adicional. Agora, os caracteres são conhecidos pela quantidade de zeros, como por exemplo:

A=30%	B=25%	C=15%	D=15%	E=10%	F=5%
1	01	001	0001	00001	000001

Este código usa um conceito de separação similar ao do código Morse.

Exemplo Descomprima 48 C2 0C, com 22 bits significativos e universo = 'AAAAA00000LLLLHHH..C'. Resposta: _____.

Compressão auto-adaptável As técnicas até agora vistas pressupõe um conhecimento prévio sobre o universo de caracteres a comprimir. Mas como utilizar estas técnicas quando nada se sabe sobre o arquivo a comprimir ? Para dar resposta a esta questão surgiram as técnicas auto-adaptáveis, que garantindo um comprimento variável de bits por byte (eficiência) prescindem de uma análise anterior no arquivo.

Vamos imaginar uma tabela de caracteres a transmitir (pode ser uma tabela de todo o código ASCII, por exemplo)

A	B	C	D

Como nada sabemos sobre a ocorrência de caracteres, vamos atribuir códigos de tamanho variável, sem qualquer critério. Pode ficar:

caracter	A	B	C	D
codificação	1	01	001	0001

Vamos acrescentar a esta tabela um contador de caracteres processados,

caracter	A	B	C	D
codificação	1	01	001	0001
contador	0	0	0	0

e agora, vamos fazer um chinezinho para ver como esta técnica se comporta
Suponhamos que o primeiro caracter que chega para ser enviado é um caracter "C". O algoritmo consulta a tabela, manda o código "001", que é o que NO MOMENTO corresponde a "C", e em seguida incrementa o contador de "C", e a tabela fica

caracter	A	B	C	D
codificação	1	01	001	0001
contador	0	0	1	0

A seguir, a tabela é ordenada, e temos:

caracter	C	A	B	D
codificação	1	01	001	0001
contador	1	0	0	0

Digamos que agora chegue um caracter D. O mesmo é enviado como 0001, o contador é incrementado, e a tabela reordenada. Fica

caracter	C	D	A	B
codificação	1	01	001	0001
contador	1	1	0	0

Agora chega de novo um caracter "C". Ele é enviado como 1, pois agora o C ocupa outra posição na tabela. Seu contador, obviamente é incrementado.

E o processo flue assim por diante. Do lado do receptor, a tabela de início é igual a do transmissor. Cada caracter que chega é decodificado com a tabela atual, e um contador também é incrementado, seguindo-se a uma reordenação da tabela. Assim procedendo, as duas tabelas são reordenadas sincronamente, e com isso a compressão é feita com sucesso.

Note-se que neste exemplo, empregou-se um código de vírgula ao invés de um dicionário à la Huffman ou mesmo Shannon-Fano. Ao escolher a vírgula o algoritmo ficou mais fácil de explicar e entender. Entretanto ele é menos eficiente.

Exemplo Descomprima 14 AD 98, com 21 bits significativos e dicionário inicial igual ao do exemplo. Resposta: _____.

Algoritmo LZW As iniciais de seus autores: Lempel, Ziv e Watch este algoritmo agrupa caracteres que costumam aparecer juntos. Ao invés de usar 8 (ou menos) bits por caracter, ele usa 12, 14 bits para representar grupos de *n* caracteres que sejam comuns. Tem desempenho ótimo. É usado no padrão GIF de imagens e será visto nas próximas aulas.

Para você fazer

1. Descomprima a seguinte mensagem usando o algoritmo de Shannon-Fano:
Universo da mensagem

EEEEEEEDDDDD00000RRRR...PPN

Mensagem a descomprimir (em hexadecimal)

10 53 8E 50 bits signif= 28

A mensagem é:

2. Descomprima a seguinte mensagem usando o código de virgulas
Universo da mensagem

AAAAAAAABBBBBB.....00000NNNNEETTTC

Mensagem a descomprimir (em hexadecimal)

01 84 10 32 8C bits signif= 38

A mensagem é:

3. Descomprima a seguinte mensagem usando a compressão auto adaptável.
O dicionário inicial é

A1...ctd=000
B01...ctd=000
C001.ctd=000
D0001ctd=000

Mensagem a descomprimir (em hexadecimal)

24 4C 91 bits signif= 24

A mensagem é:

1

Técnicas Usuais

Código Morse Só aparece aqui por razões históricas. O código morse é um sistema de representação de letras, números e sinais de pontuação através de um sinal codificado enviado intermitentemente. Foi desenvolvido por Samuel Morse e Alfred Vail em 1835, criadores do telégrafo elétrico (importante meio de comunicação à distância), dispositivo que utiliza correntes elétricas para emissão ou recepção de sinais. Veio na continuação do telégrafo visual em uso na França desde o século XVIII.

Código morse é o único modo de modulação feito para ser facilmente compreendido por humanos sem ajuda de um computador, tornando-o apropriado para mandar dados digitais em canais de voz.

Os sinais são:

1. Sinal curto, ponto ou 'dit' (.)
2. Sinal longo, traço ou 'dah' (-)
3. Intervalo entre caracteres (entre pontos e traços)
4. Intervalo curto (entre letras)
5. Intervalo médio (entre palavras)
6. Intervalo longo (entre sentenças)

Para construir a tabela morse, usa-se a busca dicotômica na sequência, com ponto à esquerda e traço à direita. E,T,I,A,N,M,S,U,R,W,D,K,G,O,H,V,F,Ü,L,Ä,P,J,B,X,C,Y,Z,Q,...

Exemplo: Traduza para o português:

...-1..1...-1.-12.1.-12. Resposta: _____.

Comprimento de fileira Escolhe-se um caracter pouco frequente (no nosso exemplo será Z). Sempre que um conjunto de mais de 3 caracteres se repetir, todos eles serão substituídos pelo conjunto Z+carater a transmitir +quantidade de repetição.

No exemplo A.....DIVIDA.....E.DE.0000000033.REAIS, ficaria AZ.5DIVIDAZ.7E.DE.Z0833.REAIS

Observações: 1.naturalmente, se o caracter de contagem for um byte binário, o contador poderá chegar até 258, pois ele pode começar em 4, neste caso, reservando-se o '0' para a ocorrência simples do caracter.

Se por acaso, o caractere de controle existir no arquivo, ele pode aparecer na saída como duplicado, com tamanho 1.

Exemplo: A.....LUZ.E.BRANCA ficaria AZ.7LUZZ1.E.BRANCA

Exemplo: Comprima usando esta técnica:

AAABBBBBBB0000000574-REAIS. Resposta: _____.

Exemplo: Traduza para o português:

EU\~6QUE\R80-\0672. Resposta: _____.

Substituição de padrões A substituição de padrões é –em geral – um método de compressão de aplicativo, já que é dependente dos padrões da aplicação. É muito usada em compressão de códigos fonte (os padrões são as palavras reservadas) ou em grandes sistemas de dados comerciais (nomes de pessoas, de ruas, de cidades, de conjuntos de zeros, CEPs, etc).

Nestes casos há uma tabela com os padrões mais freqüentes (256, 65536,...) que são convertidos para 1 ou 2 bytes. Se o arquivo for binário, não há este limite, pode-se usar qualquer tamanho de tabela de padrões (2^9 , 2^{10} , 2^{11} ...). Apenas há que se sinalizar a existência do padrão comprimido. Pode ser o sempre presente caracter de controle acima visto.

Huffman Este método já foi estudado lá em estruturas de dados. Continua o mesmo. A codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo. Ele foi desenvolvido em 1952 por David A. Huffman que era, na época, estudante de doutorado no MIT, e foi publicado no artigo "A Method for the Construction of Minimum-Redundancy Codes".

Supondo o dicionário:

P0010000	Z01100	B1100	U0100	E0000	.100
V0010001	N01101	I1101	M0101	R0001	A101
C001001	G00101	L0111	D0011	O111	

Exemplo: Descomprima:

26 98 68 D8 B4 (38 b.s.) Resposta: _____.

Codificação Relativa Este tipo de compressão se aplica apenas em casos particulares, mas quando se aplica ele é muito bom. Novamente é uma aplicação do conceito de localidade de referência. Enviado (ou gravado) o registro inicial, para os subseqüentes serão enviadas apenas as alterações em relação ao elemento anterior. Não se pode colocar aqui um esquema geral, fica apenas o conceito. Imagine-se por exemplo uma aplicação que faz captura remota de dados de telemetria. Este esquema é usado no compactador campeão que é da Intel e comprime imagens em movimento, alcançando razão de compressão de 350:1.

Compressão auto-adaptável As técnicas até agora vistas pressupõem um conhecimento prévio sobre o universo de caracteres a comprimir. Mas como utilizar estas técnicas quando nada se sabe sobre o arquivo a comprimir ? Para dar resposta a esta questão surgiram as técnicas auto-adaptáveis, que garantindo um comprimento variável de bits por byte (eficiência) prescindem de uma análise anterior no arquivo.

Vamos imaginar uma tabela de caracteres a transmitir (pode ser uma tabela de todo o código ASCII, por exemplo)

A	B	C	D

Como nada sabemos sobre a ocorrência de caracteres, vamos atribuir códigos de tamanho variável, sem qualquer critério. Pode ficar:

caracter	A	B	C	D
codificação	1	01	001	0001

Vamos acrescentar a esta tabela um contador de caracteres processados,

caracter	A	B	C	D
codificação	1	01	001	0001
contador	0	0	0	0

e agora, vamos fazer um chinezinho para ver como esta técnica se comporta. Suponhamos que o primeiro caracter que chega para ser enviado é um caracter "C". O algoritmo consulta a tabela, manda o código "001", que é o que NO MOMENTO corresponde a "C", e em seguida incrementa o contador de "C", e a tabela fica

caracter	A	B	C	D
codificação	1	01	001	0001
contador	0	0	1	0

A seguir, a tabela é ordenada, e temos:

caracter	C	A	B	D
codificação	1	01	001	0001
contador	1	0	0	0

Digamos que agora chegue um caracter D. O mesmo é enviado como 0001, o contador é incrementado, e a tabela reordenada. Fica

caracter	C	D	A	B
codificação	1	01	001	0001
contador	1	1	0	0

Agora chega de novo um caracter "C". Ele é enviado como 1, pois agora o C ocupa outra posição na tabela. Seu contador, obviamente é incrementado.

E o processo flue assim por diante. Do lado do receptor, a tabela de início é igual a do transmissor. Cada caracter que chega é decodificado com a tabela atual, e um contador também é incrementado, seguindo-se a uma reordenação da tabela. Assim procedendo, as duas tabelas são reordenadas sincronamente, e com isso a compressão é feita com sucesso.

Note-se que neste exemplo, empregou-se um código de vírgula ao invés de um dicionário à la Huffman ou mesmo Shannon-Fano. Ao escolher a vírgula o algoritmo ficou mais fácil de explicar e entender. Entretanto ele é menos eficiente.

Exemplo Descomprima 14 AD 98, com 21 bits significativos e dicionário inicial igual ao do exemplo. Resposta: _____.

Algoritmo LZW As iniciais de seus autores: Lempel, Ziv e Watch este algoritmo agrupa caracteres que costumam aparecer juntos. Ao invés de usar 8 (ou menos) bits por caracter, ele usa 12, 14 bits para representar grupos de n caracteres que sejam comuns. Tem desempenho ótimo. É usado no padrão GIF de imagens e será visto nas próximas aulas.

Para você fazer Traduza para português, as mensagens morse

- 1) -. -1.-1.-.1.-.-1.-1---12
- 2) -. -1..-1.-.1...1---1...12
- 3) -. .1.1-1.-.1.-1...12

Comprima as frases a seguir, usando o método de comprimento de fileira. Use a \ como caracter de controle.

- 4) YYYYYYYYYXXXXXXKKKKKKKKXZZZZZZDDDDDDJ
- 5) HHHHHHHHAAAAAAHHHHHHHH-----GH-----

Descomprima usando Huffman: (use o mesmo dicionário do exemplo)

- 6) AC 47 84 82 48 39 bits significativos
- 7) 26 8D 95 80 74 38 bits significativos
- 8) 5A 42 C1 4D 58 39 bits significativos

Descomprima a seguinte mensagem usando a compressão auto adaptável. O dicionário inicial é

A1...ctd=000
B01...ctd=000
C001.ctd=000
D0001ctd=000

Mensagem a descomprimir (em hexadecimal)

- 9) 18 C8 92 bits signif= 23
- 10) 46 89 92 bits signif= 23

Ache um lugar adequado para responder as 10 perguntas feitas. Se quiser, use o verso.



- 1 - /

Teoria das Filas

A Teoria das Filas é um ramo das probabilidades que estuda a formação de filas. Uma fila surge sempre que há escassez de algum recurso, ou seja, quando a demanda é maior do que a oferta para ele. O tratamento matemática das filas permite obter características de desempenho prévios à sua existência real.

Fundamentalmente, o tratamento de um problema é obter dados sobre tamanho da fila - L_f e tempo total dos clientes - T , a partir dos dados de entrada (tipicamente ritmo de chegada de clientes - λ , tempos de atendimento - μ e número de servidores - n).

Servidor único

Supondo que os tempos interchegadas estão descritos por uma distribuição exponencial e o número de clientes chegando por uma distribuição de Poisson, então supõe-se conhecida a taxa de chegada: é o valor de λ . Vamos supor neste exemplo uma taxa de chegada de 16 clientes/hora.

Conhecido o tempo médio de serviço, pode-se obter a taxa de atendimento que vem a ser o inverso deste. Assim, se T_s é igual – por exemplo – a 2 minutos, a taxa de atendimento será $\mu = 1/2 \text{ cli/min}$ ou $\mu = 0,5 \text{ cli/min}$ ou ainda $\mu = 30 \text{ cli/hora}$.

A taxa média de ocupação do servidor - ρ é dada pela fórmula $\rho = \lambda/\mu$. Neste exemplo será de $\rho = 16/30 = 0,53\%$.

O cálculo do tamanho da fila - L_f é dado por $L_f = \lambda^2/[\mu \cdot (\mu - \lambda)]$. No exemplo, $L_f = 16^2/30 \cdot (30 - 16) = 256/30 \cdot 14 = 256/420 = 0,60$. A leitura deste número informa, que em média, haverá 0,6 clientes na fila.

O cálculo da quantidade de clientes em atendimento coincide com taxa de ocupação do servidor, neste caso $\rho = 0,53$.

O cálculo de quantidade total de clientes - L é dado pela fórmula $L = \lambda/\mu - \lambda$ ou no nosso exemplo, $L = 16/30 - 16 = 16/14 = 1,14$. A leitura deste número informa que em média haverá 1,14 clientes no sistema.

Como não poderia deixar de ser, a quantidade de clientes L , é igual à soma da quantidade de clientes na fila L_f com a quantidade de clientes em atendimento L_s , ou $L = L_f + L_s$ ou no nosso exemplo $1,14 = 0,60 + 0,53$, a menos dos erros de arredondamento.

O cálculo dos tempos, segue raciocínio análogo, embora com fórmulas mais simples. Assim, o tempo total médio no sistema é dado por $T = L/\lambda$. O tempo de fila médio é dado por $T_f = L_f/\lambda$ e o tempo médio de atendimento é dado por $T_s = L_s/\lambda$. No exemplo, $T = 1,14/16 = 0,071 \text{ hr} = 4,27 \text{ min}$.

Já o tempo de fila é dado por $T_f = L_f/\lambda$ ou no exemplo $T_f = 0,60/16 = 0,037 \text{ hr} = 2,25 \text{ min}$. Finalmente, o tempo de atendimento T_s é dado por $T_s = L_s/\lambda = 0,53/16 = 0,033 \text{ hr} = 1,98 \text{ min}$. Como não poderia deixar de ser, a soma de ambos os tempos T_f e T_s tem que reproduzir o tempo total, assim $2,25 + 1,98 = 4,23$ que coincide com o T , a menos dos erros de arredondamento.

Em resumo, eis as fórmulas a serem usadas

variável	descrição	origem
λ	taxa média de chegada de clientes em clientes/unid tempo	dado do problema
μ	taxa de atendimento por servidor em clientes/unid tempo	É o inverso do tempo médio de atendimento, ou $1/T_s$, também dado
ρ	taxa de uso do servidor	$\rho = \lambda/\mu$
L_f	tamanho médio da fila	$L_f = \frac{\lambda^2}{\mu \cdot (\mu - \lambda)}$
L_s	quantidade média de clientes sendo atendidos	$L_s = \rho = \lambda/\mu$
L	quantidade total de clientes no sistema	$L = L_f + L_s$
T_f	tempo médio na fila	$T_f = L_f/\lambda$
T_s	tempo médio de serviço	$T_s = L_s/\lambda = 1/\mu$
T	tempo total no sistema	$T = T_f + T_s$

Múltiplos servidores com filas individuais

Deve-se notar que se o sistema tem múltiplos servidores, cada um com a sua fila própria e que se os clientes uma vez entrando em uma fila NÃO trocam de fila, então o sistema pode ser estudado como se fossem vários sistemas independentes. A análise matemática parte da divisão da taxa de chegada pelo número de servidores e a análise segue o visto acima.

Por exemplo, num sistema de 3 servidores com filas independentes e imutáveis, com taxa de chegada de 48 clientes/hora, então, deve-se dividir esta taxa (48 cl/hr) pelo número de servidores (3) e prosseguir a análise com um sistema individual com taxa de 16 chegadas/hora.

Múltiplos servidores com fila única

A formulação matemática aqui é mais complexa, mas torna-se simples pela consulta a tabelas ou gráficos como os expostos em *MARTIN, James. Design of Real Time Computer Systems*, páginas 417 a 422, um dos quais está reproduzido ao final deste texto. A taxa de chegada ainda é a mesma, λ . Idem para o tempo de atendimento T_s e para a taxa de chegada $\mu = 1/T_s$. A taxa de ocupação tem fórmula diferente, considerando o número de servidores ativos n . Ou seja, $\rho = \lambda/n \cdot \mu$.

Neste tipo de sistemas é necessário obter a probabilidade de ter um cliente esperando, com a probabilidade de que todos os M servidores estejam

ocupados, em sistemas com tempos interchegadas e tempos de serviços descritos por distribuições exponenciais, com todos os servidores igualmente carregados e com atendimento em base FCFS (*first come, first served*) é

$$P(ocup) = \frac{1 - \left[\frac{\sum_{i=0}^{M-1} \frac{(M \times \rho)^i}{i!}}{\sum_{i=0}^M \frac{(M \times \rho)^i}{i!}} \right]}{1 - \left(\rho \times \left[\frac{\sum_{i=0}^{M-1} \frac{(M \times \rho)^i}{i!}}{\sum_{i=0}^M \frac{(M \times \rho)^i}{i!}} \right] \right)}$$

Se você achar esta fórmula muito chata, obtenha o mesmo valor consultando o gráfico da figura que consta na aula AUTOSII publicada no portal.

O tamanho da fila, tem a seguinte formulação: $L_f = [P(todos \text{ ocupados}) \cdot \rho] / (1 - \rho)$. No hipotético exemplo de 48 clientes/hora e 3 servidores, ter-se-ia: $\rho = \lambda/n \cdot \mu = 0,53$. Olhando o grafico da figura da aula, descobre-se que para $\rho = 0,53$ e $n = 3$, descobre-se que a $P(ocupados) = 0,27$. Com isso, a $L_f = (0,27 \times 0,53) / (1 - 0,53) = 0,304$.

A quantidade de clientes sendo atendidos, L_s é igual a λ/μ . No exemplo, $L_s = 48/30 = 1,60$. Como existem 3 servidores, na média haverá 1,60 clientes sendo atendidos.

A quantidade de clientes no sistema, $L = L_f + L_s$, neste caso é de $L = 0,30 + 1,60 = 1,90$ clientes.

Quanto aos tempos, tem-se o Tempo de serviço - T_s , que continua sendo de 2 minutos.

O tempo de fila - T_f , possui como fórmula $T_f = L_f/\lambda$, ou devidamente manipulado, $T_f = P(todos \text{ ocup}) / [(n \times \mu) - \lambda]$. No exemplo, segundo a primeira formulação, tem-se $T_f = 0,30/48 = 0,00625 \text{ hr} = 0,375 \text{ min}$. Já usando a segunda formulação, obtem-se $T_f = 0,27 / (3 \times 20) - 48 = 0,27/42 = 0,00642 \text{ hr} = 0,385 \text{ min}$ com os erros devidos ao arredondamento. Finalmente, o tempo total T , como sempre, é a soma entre T_f e T_s , dando como resultado $T = 0,38 + 2 = 2,38 \text{ min}$.

Para o caso de multiservidores com fila única, obtem-se o seguinte formulário

variável	descrição	origem
λ	taxa média de chegada de clientes em clientes/unid tempo	dado do problema
μ	taxa de atendimento por servidor em clientes/unid tempo	É o inverso do tempo médio de atendimento, ou $1/T_s$, também dado
n	número de servidores	dado do problema
ρ	taxa de uso do servidor	$\rho = \frac{\lambda}{n \times \mu}$
L_f	tamanho médio da fila	$L_f = \frac{P(ocup) \times \rho}{1 - \rho}$
L_s	quantidade média de clientes sendo atendidos	$L_s = \lambda/\mu$
L	quantidade total de clientes no sistema	$L = L_f + L_s$
T_f	tempo médio na fila	$T_f = L_f/\lambda = \frac{P(ocup)}{(n \times \mu) - \lambda}$
T_s	tempo médio de serviço	$T_s = L_s/\lambda = 1/\mu$
T	tempo total no sistema	$T = T_f + T_s$

A conclusão do exemplo visto anteriormente, quando comparamos o desempenho entre um sistema de filas individuais ($T = 4,2 \text{ min}$) e um de fila única ($T = 2,4 \text{ min}$), percebe-se que SEMPRE é mais eficiente trabalhar com uma fila única.

Para você fazer

Você foi convidado – como analista de problemas e de suas soluções, a estudar o dimensionamento do Supermercado "O Faminato de Ouro". Outros já estudaram o problema e chegaram a algumas conclusões:

- A taxa de chegada de clientes ao mercado é de 210 clientes por hora. Nenhum cliente é perdido (desiste da compra). As filas são atendidas na modalidade FCFS.
- O tempo médio de atendimento de cada caixa é de 3 minutos, com baixo (desprezível) desvio padrão.
- O número de caixas a serem instalados no mercado é de 15 caixas, que sempre estarão funcionando todos.

A dúvida do dono do mercado é quanto à organização da fila. Embora haja problemas operacionais para organizar uma fila única, alguém soprou para o dono que o desempenho do sistema é muito melhor. Ele não sabe o que fazer.

Você deve calcular o Tempo médio que os clientes gastarão no mercado em 2 casos: havendo 15 filas individuais e imutáveis (os clientes não trocarão de fila), OU havendo uma única fila de atendimento. Responda em minutos inteiros, despreze as decimais.

Tempo médio total ($T_f + T_s$) para 15 filas	Tempo médio total ($T_f + T_s$) para fila única
minutos	minutos

Simulação ad hoc

Embora o formulário matemático visto na aula passada sobre a Teoria das Filas seja tentador por ser de fácil e rápida aplicação, a verdade é que infelizmente, nos casos reais, quase nunca ele pode ser usado. Sistemas reais, sempre têm um detalhe ou outro que os afasta da teoria.

Nesta aula, veremos 2 exemplos disso, o que nos obrigará a modelar o sistema “na marra”.

Problema 1: Um Banco

O Banco Queromeu SA, embora pequeno, tem uma política agressiva de conquista de novas praças. Por essa razão, ele pretende se instalar em Curitiba, numa agência localizada em imóvel recentemente adquirido na Rua XV de Novembro. Estudos conduzidos em praças similares a Curitiba permitiram à organização estimar alguns parâmetros. Em um dia normal de funcionamento, serão esperados os seguintes clientes

num.	hora	che- gada	tempo	num.	hora	che- gada	tempo
1	10 h 15		20	21	11 h 25		20
2	10 h 20		25	22	11 h 30		5
3	10 h 20		20	23	11 h 35		25
4	10 h 25		10	24	11 h 35		25
5	10 h 25		10	25	11 h 40		30
6	10 h 30		20	26	11 h 45		10
7	10 h 30		5	27	11 h 45		30
8	10 h 35		30	28	11 h 50		25
9	10 h 45		25	29	11 h 50		5
10	10 h 45		5	30	11 h 50		20
11	10 h 50		30	31	11 h 50		15
12	10 h 50		5	32	11 h 50		25
13	10 h 55		20	33	12 h 5		5
14	10 h 55		25	34	12 h 20		15
15	10 h 55		5	35	12 h 25		20
16	11 h		15	36	12 h 35		5
17	11 h 5		5	37	12 h 40		15
18	11 h 20		30	38	12 h 45		10
19	11 h 20		15	39	13 h 20		25
20	11 h 20		25	40	13 h 40		5

A agência abrirá das 10h00 às 14h00, ou seja por 4 horas diárias. Haverá espaço para 5 caixas, e uma área de espera de 12 clientes. Quando houver filas individuais, o cliente sempre vai escolher a menor delas e depois, não trocará mais de fila até ser atendido. Se houver empate entre 2 filas, o cliente escolherá a de menor número. O tempo de troca entre clientes (*setup time*) é zempre 0. Para facilitar, você pode considerar o tique de relógio de 5 em 5 min. O preço imaginado para um cliente perdido é de R\$ 900 . O preço diário de manter um caixa trabalhando é de R\$ 1600 . Quando a agência fechar, quem estiver sendo atendido, o é até o fim.

Porque não é possível Usar o formulário: Como há um espaço limitado para clientes, os que não couberem – quando chegarem – irão embora e não serão atendidos. Também, quando a agência fecha, os que estiverem na fila são mandados embora, sem atendimento.

🔗 Para você fazer

Realize a simulação pedida, e ao final, responda

item	qtd clientes atendidos	custo médio por cliente atendido
1 único caixa		
3 caixas com fila única		não responder
3 caixas com filas indivi- duais		não responder
5 caixas com fila única		

Problema 2: fila dentro do Sist. Operacional

Imagine-se um sistema operacional multiprogramado despachando tarefas. Como o processador é um só, e existem muitos processos querendo-o usar, formam-se filas. Se o algoritmo for FIFO também conhecido como FCFS (*first come, first served*), as fórmulas podem ser usadas. Entretanto, aqui vão ser usados 2 algoritmos distintos:

SJF *Shortest Job First*. Neste esquema, quando o processador vai atender alguém, ele escolhe na fila de espera, aquele que tiver a menor requisição de tempo. Se houver empate, o primeiro será retirado.

RR *Round Robin*. Neste esquema preemptivo, a fila será atendida em ordem, mas com uma quantidade fixa de tempo para cada processo. Após receber este *quantum*, o processo vai para o fim da fila.

Para este problema, haverá um padrão de chegadas de requisições de tempo. A unidade a ser considerada é o segundo. Não haverá nenhum tempo de *overhead* para troca de processos. Irão ser usados para cada processo, dois tempos, assim definidos:

duração do processo tempo de fim - tempo de início + 1

elapsed time tempo de fim - tempo de chegada + 1

Acompanhe no exemplo feito, usando o método **Shortest Job First**

proc	T cheg	Pedido	Tinicio	Tfim	Duração	Elapsed
H	1	5	1	5	5	5
Z	2	5	9	13	5	12
K	6	3	6	8	3	3
D	10	3	16	18	3	9
U	11	10	63	72	10	62
W	13	2	14	15	2	3
J	14	9	54	62	9	49
C	15	6	21	26	6	12
S	18	2	19	20	2	3
X	21	11	73	83	11	63
I	24	8	27	34	8	11
M	31	6	42	47	6	17
Y	32	4	35	38	4	7
G	37	6	48	53	6	17
B	38	3	39	41	3	4

Veja o resultado do mapa do processador, onde cada letra corresponde a 1 seg. Espaços são só para clareza e separam 10 segundos.

HHHHKKZZZ ZZZWDDSS CCCCCCIIII IIIIIYYYBB BMMMMMGGG GGGJJJJJJ
JJUUUUUUU UUXXXXXXXX XXX

O que se quer saber são as médias de duração e de elapsed. No exemplo acima, elas são de 5.5 segundos e 18.5 segundos respectivamente.

Porque não é possível Usar o formulário: Porque a disciplina das filas não é FCFS.

🔗 Para você fazer

Imagine a seguinte situação em um ambiente multiprogramado com o método **Shortest Job First** .

proc	T cheg	Pedido	Tinicio	Tfim	Duração	Elapsed
D	1	10				
E	7	3				
C	13	11				
Y	16	11				
H	18	2				
X	20	11				
T	22	7				
J	24	3				
K	27	6				
L	31	10				
M	32	10				
N	35	4				
A	38	7				
F	44	3				
S	45	9				

Para preencher a tabela acima, use o seguinte espaço de tempo

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150

Responda aqui, qual a média de duração e de elapsed do exercício proposto com o método **Shortest Job First** .

duração	elapsed

1

ARENA

Neste software, considerado um padrão na área de simulação e disponível sem ônus para estudantes ¹ o software permite o desenho do modelo, sua descrição, inclusive analítica e posteriormente a sua simulação. Siman vem a ser a linguagem de simulação do pacote ARENA.

O ARENA foi lançado pela Systems Modelling em 1993 e é o sucessor do SIMAN (em 1982, o primeiro simulador para PC) e do CINEMA (1984, já com representação visual das simulações). O SIMAN foi uma evolução da arquitetura do GPSS, este lançado em 1961 pela IBM. Em 1998, a Rockwell Software incorporou a Systems Modelling.

Instalação do ARENA

No CD anexo, localize o diretório ARENA70. Baixe-o para um diretório temporário. Execute neste o instalador (SETUP.EXE), responda às perguntas e depois disso o ARENA terá sido instalado.

👉 Para você fazer

Nos exercícios a seguir, você deve:

- entender o problema, seu entorno, suas variáveis e o comportamento do sistema em estudo.
- definir o sistema dentro do ambiente ARENA, testando-o e depurando-o.
- executar a simulação no ARENA pelo menos 3 vezes
- responder nesta folha o valor numérico solicitado que SEMPRE estará disponível em algum relatório do ARENA. Só resta saber em qual...

Exercícios

- Uma central de atendimento de emergências de ambulâncias atende a uma chamada a cada 15 minutos. Os tipos de chamada tem as seguintes características:

tipo de chamada	% das chamadas	tempo de atendimento
Falsa	15%	12 min +/- 2
Emergência sem risco de vida	85% das não falsas	25 +/- 5min
Emergência com risco de vida	15% das não falsas	20 +/- 10

Suponha existirem muitas ambulâncias. Simule o sistema para um total de 500 chamadas. Determine

- O tempo médio para o atendimento das chamadas
- Suponha que existe apenas uma ambulância. Ela pode dar conta do serviço ?
- Quantas ambulâncias seriam necessárias para um atendimento razoável ?
- O que é atendimento razoável ?

respostas

- Uma linha de montagem de placas de computadores consiste de 4 estações que adicionam componentes a um circuito impresso. Cada estação é composta por 2 serviços: inspeção e montagem. Quando uma placa entra em uma estação ela é primeiro inspecionada e se aprovada a ela são agregados mais componentes. Se uma placa falha na inspeção, volta ao posto de montagem anterior para retrabalho. Se uma placa falha na primeira inspeção ela é retirada do sistema. Uma nova placa entra no sistema a cada 2 min. Os tempos de inspeção são uniformemente distribuídos entre 1,3 e 2,1 minutos. Já os tempos de montagem seguem uma distribuição normal de 1,8 min de média e 0,4 min de desvio padrão. As probabilidades de falhas nas diversas estações são: 4 %, 6%, 1 % e 5 % respectivamente nas estações 1 a 4. Os tempos de transferência dentro da estação são negligenciáveis (zero). A passagem da montagem para a inspeção seguinte é de 1 min. Entre a inspeção e o posto de montagem anterior em caso de falhas, gasta-se 2,5 min. Modele e simule o sistema, pelo menos 3 vezes, determinando estatísticas sobre

- o tempo de produção de uma placa
- o número de placas produzidas em 8 h de trabalho

- o número de placas defeituosas
- o número total de placas no sistema.

respostas

- Existe uma agência de viagens instalada dentro de um grande e importante hotel em nossa cidade. O grande apelo promocional do hotel é "VOCE NUNCA ESPERA MAIS DE 360 SEGUNDOS", e ele quer que nenhum serviço demore mais do 360 segundos até a pessoa ser atendida. Sabendo que haverá em média 10 clientes/hora chegando, com tempo de atendimento de 15 minutos, pergunta-se quantos atendentes deverá haver

- com filas individuais
- com fila única

Para que nenhum cliente demore mais do 360 segundos para começar a ser atendido ?

respostas

- Considere um lavacar. Ele está instalado em um terreno imenso, no qual cabem muitos (∞) carros. Os carros chegam em uma distribuição exponencial, com média de 3 carros por hora. O processo de lavagem dos carros demora em média 15 minutos. Existe apenas um empregado para lavar os carros. Pergunta-se:

- qual o tempo médio dos carros dentro do Lavacar ?
- em média quantos carros haverá na fila da lavagem ?
- Se o lavacar funcionar das 08h00 as 20h00 diariamente, quantos carros em média serão lavados por dia ?
- qual o tempo médio dos carros na fila ?

respostas

- Considere um lavacar. Ele está instalado em um terreno imenso, no qual cabem muitos (∞) carros. Os carros chegam em uma distribuição exponencial, com média de 6 carros por hora. O processo de lavagem dos carros demora em média 15 minutos. Existem 2 empregados para lavar os carros. Pergunta-se:

- qual o tempo médio dos carros dentro do Lavacar no caso de se fazer fila única ?
- em média quantos carros haverá na fila da lavagem para fila única ?
- Se o lavacar funcionar das 08h00 as 20h00 diariamente, quantos carros em média serão lavados por dia ? (fila única)
- qual o tempo médio dos carros na fila ? (fila única)

respostas

¹Vendido/cedido em diversas configurações, o Arena pode ser copiado e instalado sem reservas na versão para estudantes. Ela tem as funcionalidades do software completo, mas limita a construção de modelos às seguintes quantidades: 50 blocos, 100 Siman, 100 animações e 100 entidades

Números Randômicos ou aleatórios

Importantes em muitos campos do conhecimento humano. Pelo menos

simulação de eventos Randomicidade é necessária para deixar as coisas "reais". Física nuclear, comportamento de sistemas, pesquisa operacional... *é barato simular num computador*

amostragem quando é impossível examinar todas as amostras

jogos em computador já imaginaram um jogo que sempre joga igual ?

computação evolutiva Operadores de seleção, recombinação, mutação.

estruturas de dados listas skip, por exemplo

protocolos de comunicação CSMA-CD, por exemplo

criptografia a base dos sistemas DES e RSA, entre outros

redes neurais inicialização é aleatória

Sugestões de obter real aleatoriedade: a) relógio do computador: baixa aleatoriedade; b) entrada de som do computador, devidamente processada por uma função HASH; c) movimentos do mouse (usada, por exemplo no PGP).

Números pseudo-aleatórios (pseudo-randômicos)

Na geração de uma seqüência de números quasi randômicos, existem vários métodos. O primeiro, devido a John V. Neumann, é de 1946.

Metade do quadrado (V. Neumann)

A seqüência começa com um primeiro número (X_0) chamado SEMENTE de k dígitos. O próximo número (X_1) obtém-se:

- elevando X_0 ao quadrado
- obtendo os k dígitos centrais do quadrado. Se este tem número ímpar de dígitos, deve-se pegar a parte central apenas e complementar com 0s.

Este esquema tem interesse apenas histórico pois gera ciclos pequenos.

Exemplos Seja a semente = 2389 e k=4. Elevando-a ao quadrado, tem-se 5707321. Obtendo-se os 4 dígitos centrais (no caso apenas 3), fica-se com 73 e portanto $X_1=73$. Elevando este ao quadrado, fica 5329, que é $X_2=5329$. Elevando-se ao quadrado X_2 , obtém-se 28398241 e pegando-se os k dígitos centrais, tem-se $X_3=3982$. Elevando este ao quadrado tem-se 15856324 e $X_4=8563$.

Acompanhe agora dois exemplos individuais:

Semente=6924, k=4, $X_1=9417$, $X_2=6798$, $X_3=2128$, $X_4= 283$

Semente=3702, k=4, $X_1=7048$, $X_2=6743$, $X_3=4680$, $X_4=9024$

Método de Fibonacci

Um segundo método proposto em 1958 por por G.J. Mitchell e D.P. Moore, baseia-se na seqüência de Fibonacci e tem como fórmula: $X_n = (X_{n-24} + X_{n-55}) \bmod m$ com $n \geq 55$, sendo os primeiros 55 elementos (de X_0 a X_{54}) inteiros arbitrários não todos pares e m par. A escolha de 24 e 55 não é aleatória e está justificada no Knuth, vol. 2, pags. 26 e 27.

Exemplo seja a seqüência inicial:

80	538	11	794	483	950	229	90	235	533	107	282	863	901
996	317	477	188	671	822	524	691	629	386	699	838	333	973
37	493	492	246	188	708	207	767	864	434	498	627	584	319
513	877	27	173	370	986	835	930	516	246	448	468	941	

e seja m = 342

Então o quinquagésimo sexto termo, X_{55} será obtido fazendo-se ($X_{31} + X_0$) mod 342, o que dá (80+246) mod 342=326 e assim $X_{55}=326$. O próximo termo, X_{56} é obtido fazendo-se (538+188) mod 342=42 e assim por diante. ($X_{57}= 35$). Veja a seguir, um exemplo individual. Seja a seqüência inicial (***):

404	504	252	622	100	900	448	437	987	294	753	556	674	663
226	916	208	204	989	205	792	824	119	130	58	499	295	257
401	383	839	68	872	170	485	916	97	822	109	135	248	828
903	153	140	91	20	269	286	61	618	633	840	423	656	

E, seja m= 300 Então, $X_{55}= 172$, $X_{56}= 176$, $X_{57}= 122$, e assim por diante.

Método Congruencial Linear

O terceiro é o chamado método congruencial linear, devido a D. H. Lehmer em 1949, e ainda hoje muito usado.

Este método inicia com quatro números:

o módulo m (com $m > 0$)

o multiplicador a (com $0 \leq a < m$)

o incremento c (com $0 \leq c < m$)

a semente X_0 (com $0 \leq X_0 < m$)

O enésimo termo da seqüência é calculado pela fórmula: $X_{n+1} = ((a.X_n) + c) \bmod m$

Exemplo

Se m=7382, a= 821, c= 1 então $X_0=5733$; $X_1=4460$; $X_2= 189$; $X_3= 148$; $X_4=3397$

Mais alguns exemplos individuais:

1. m= 8331 , a= 21 , c= 1 , $X_0= 7761$; $X_1= 4693$; $X_2= 6913$; $X_3= 3547$; $X_4= 7840$

2. m= 8080 , a= 821 , c= 1 , $X_0= 4856$; $X_1= 3337$; $X_2= 558$; $X_3= 5639$; $X_4= 7860$

3. m= 9488 , a= 421 , c= 1 , $X_0= 6689$; $X_1= 7622$; $X_2= 1919$; $X_3= 1420$; $X_4= 77$

As sugestões para obtenção dos números m, a, c e X_0 são:

- X_0 pode ser qualquer. Se o programa for ser usado muitas vezes, pode sempre ser o último número gerado na última série. Pode ser também uma combinação de data e hora.
- O número m pode ser tão grande quanto se possa (operar). Knuth sugere que seja a capacidade de uma palavra do computador hospedeiro.
- Se m for uma potência de 2, a deve ser tal que $a \bmod 8 = 5$. Se m for potência de 10, deve-se ter $a \bmod 200 = 21$. A variável a deve estar entre $0.1m$ e $0.99m$ e seus dígitos (seja em representação binária ou decimal) não devem ter um padrão regular simples.
- c pode ser qualquer desde que não tenha múltiplo comum com m .
- Valores usuais são $c = 1$ ou $c = a$.
- Os dígitos menos significantes (à direita) de cada número gerado não são inteiramente randômicos. Para tanto, deve-se imaginar a geração de um aleatório compreendido entre 0 e 1 e devolver $X_n \div m$. Para computar um inteiro entre 0 e k-1, deve-se multiplicar $X_n \div m$ por k e truncar o resultado.

Para você fazer

Neumann

Ache os valores pedidos:

Semente=4593, k=4

$X_1=$ _____ ; $X_2=$ _____ ; $X_3=$ _____ ; $X_4=$ _____.

1º RESPOSTA: coloque X_4 da seqüência acima na resposta 1

Semente=4331, k=4

$X_1=$ _____ ; $X_2=$ _____ ; $X_3=$ _____ ; $X_4=$ _____.

2º RESPOSTA: coloque X_4 da seqüência acima na resposta 2.

Fibonacci

Use a mesma seqüência anterior, marcada com (***) e

m= 344 e calcule

$X_{55}=$ _____ ; $X_{56}=$ _____ ; $X_{57}=$ _____ ; $X_{58}=$ _____.

3º RESPOSTA: coloque X_{58} acima na resposta 3

Congruencial Linear

Ache os valores pedidos:

a) $m= 9521$, $a= 221$, $c= 1$, $X_0= 6684$

$X_1=$ _____ ; $X_2=$ _____ ; $X_3=$ _____ ; $X_4=$ _____.

4º RESPOSTA: coloque X_4 acima na resposta 4

b) $m= 5677$, $a= 621$, $c= 1$, $X_0= 6855$

$X_1=$ _____ ; $X_2=$ _____ ; $X_3=$ _____ ; $X_4=$ _____.

5º RESPOSTA: coloque X_4 acima na resposta 5

Respostas

1	2	3	4	5

1

Bibliografia: The Art of Computer Programming, Donald Knuth, vol2. cap. 1. Seguranca de Dados com Criptografia - metodos e algoritmos, Daniel Carvalho e The Turing Omnibus, de A. K. Dewdney.



- 1 - /

Simulação Este é um dos papeis mais nobres que um computador pode desempenhar. Sua absoluta generalidade permite programar nele a simulação de quaisquer fenômenos naturais ou artificiais, desde que se possa construir um sistema fenomenológico aderente ao objeto de estudo. Quanto mais próximo deste, mais valiosos serão os resultados obtidos. Eis alguns exemplos instigantes:

★ Querendo estudar a disseminação de um vírus mortal em uma grande cidade, é impossível proceder a um teste real. Criando-se um conjunto de postulados, leis de formação, premissas e algoritmos convenientes, pode-se disseminar o tal vírus, fazendo perguntas, hipóteses, questões do tipo **e se ?**, sem correr nenhum risco, a custos baixos e de uma maneira muito rápida.

★ Ao projetar um grande estacionamento, pode-se querer modelar o tempo de saída em função da quantidade de postos de cobrança e das saídas físicas, através de um modelo.

★ Ao projetar facilidades de deslocamento em uma rede de transporte integrado, pode-se estudar o que acontece com os diversos modais vis-a-vis os correspondentes investimentos em cada um deles.

★ Depois da proibição de explosões nucleares na atmosfera do planeta (por volta dos anos 70) só se pôde continuar estudando o assunto dentro de um computador.

O computador é a máquina respondedora por natureza. Ela pode responder qualquer pergunta que se lhe queira fazer. Na medida em que ele estiver bem (corretamente) programado, suas respostas serão exatas, precisas, rapidamente obtidas e inquestionáveis. Não há outra maneira de conseguir tais respostas.

Quando o mecanismo que se quer simular é conhecido, a simulação é mais simples e as respostas mais rápidas. Ao contrário, quando o processo em estudo só é conhecido em parte, a simulação se transforma em algo exploratório, a própria simulação nos ajuda a entender, compreender e construir o processo original.

Neste exercício vai-se ficar restrito a problemas integralmente conhecidos. Quiçá estes sejam os que mais certificam o valor do computador como respondedor de perguntas.

Aleatoriedade Um pré-requisito importante para simular é ter um bom (confiável) gerador de aleatoriedade. Números randômicos são imprescindíveis para ter um mínimo de aderência a cenários reais. A natureza é randômica (a hora em que um cliente entrará em uma farmácia não obedece a nenhuma lei regular). Na ausência de circuitos de hardware capazes de gerar randomicidade, precisamos nos contentar com pseudo-randomicidade (isto é aleatoriedade gerada por algoritmos – o que infelizmente é um contrasenso).

Porta da Esperança Este programa de TV foi apresentado na TV americana entre os anos de 1963 e 1976 ininterruptamente. O apresentador era o simpático Monty Hall, que tinha uma assistente bonita com (poucas) roupas provocativas, Carol Merrill, miss Califórnia, 1957. Foram 4500 episódios em 27 anos Chamava-se *Let's make a deal*. Ainda na década de 90 foi relançado e mesmo hoje volta e meia reaparece. A coisa é simples: um telespectador convidado sobe ao palco e deve escolher uma de três portas que estão fechadas. Em uma porta há um presente valioso (vamos imaginar uma masserati 0 Km de cor vermelha), e nas outras duas portas os prêmios são de muito menor valor: por hipótese: a coleção completa da revista CARAS em uma porta e na outra a discografia completa do cantor Fagner.

Quando o telespectador escolhe uma porta, a mesma NÃO é aberta ainda, o apresentador do programa (que sabe perfeitamente onde está a ferrari – esta informação é muito relevante) escolhe uma das portas ainda fechada e com valor baixo (CARAS ou o Fagner), faz um suspense danado, abre a porta e mostra o que tem lá.

A situação então é: a porta escolhida pelo telespectador permanece fechada, uma segunda porta que tem um presente de baixo valor foi escancarada e a terceira porta permanece fechada.

O apresentador lança a pergunta ao telespectador. *Quer mudar sua escolha original ?* A pergunta que vai-se fazer ao computador é se vale a pena mudar a escolha ou se esta hipótese (a mudança da porta escolhida) é absolutamente irrelevante.

As respostas que podem ser dadas são três:

é irrelevante não faz a menor diferença trocar ou não a porta escolhida, até porque não se sabe o que há por trás das duas portas que permanecem fechadas.

é vantajosa trocar a porta originalmente escolhida. Quem agiu assim – na média – saiu-se melhor.

é desvantajosa trocar a porta. Quem agiu assim, sempre saiu perdendo.

O programa

```
1: m ← aleatorio(1-3) // onde está a masserati
2: f ← aleatorio(1-3) // o que o fregues escolheu
3: se m=f
4:   ruins ← 1-2-3 menos m
5:   mostra ← ruins[aleatorio(1-2)]
6:   nm ← a porta podre ainda fechada
7:   se quer mudar ?
8:     MUDOU E PERDEU
9:   senão
10:    NAO MUDOU E GANHOU
11:  fim{se}
12: senão
13:   ruins ← (1-2-3) menos m
14:   mostra ← a porta ruim que não foi escolhida por f
15:   nm ← a porta ruim ainda fechada // a escolha do fregues...
16:   se quer mudar ?
17:     MUDOU E GANHOU
18:   senão
19:     NAO MUDOU E PERDEU
20:   fim{se}
21: fim{se}
```

Como observação final, havia uma colunista de matemática em inúmeros jornais americanos, chamada Marilyn von Savant. Um dia de 1990 ela publicou uma sugestão de que a hipótese de trocar a porta era sempre a melhor e foi atacada por muita gente boa, em termos ríspidos e nada educados. Ela estimou ter recebido 10.000 cartas xingando-a. 1000 dessas cartas eram de PhDs.

Quantas questões no BD ? Esta é outra simulação que aconteceu de verdade. Um dia, em meados da década 2000, a Secretaria de Educação do Paraná resolveu automatizar a emissão de provas do EJA (antigo supletivo). Nesta modalidade educacional os alunos não precisam frequentar as aulas e quando se sentirem capacitados solicitam à autoridade educacional a realização de uma prova de conhecimento. Aprovados na prova, ganham o grau a que concorreram. Antes da iniciativa, as provas eram produzidas pelo processo manual tradicional e eram aplicadas algumas vezes ao ano em locais fixos e determinados em nosso estado.

A proposta era criar um sistema de computador que permitisse a qualquer aluno apresentar-se num CEEJA (Centro de Educação de Jovens e Adultos) e solicitar fazer a prova. A mesma seria emitida em papel, entregue ao aluno e posteriormente corrigida. Assim, o exame seria descentralizado no tempo e no espaço.

A questão que me foi proposta então foi: quantas questões deve ter o banco de dados de questões ? O critério era que duas provas emitidas em sequência não poderiam ter nenhuma questão repetida. Educadamente disse que estávamos no terreno das probabilidades. Se a probabilidade em questão era 0% (nunca!) então a resposta era fácil: **infinitas** questões.

Diante da óbvia inadequação da resposta, começou a negociação, ou melhor: o estudo e a reespecificação da questão. Refeita a regra do critério ficou assim: *Emitidas duas provas em sequência, com 50 questões cada, a probabilidade de haver 2 questões iguais deveria ser inferior a 1%.*

A maneira de responder a esta questão foi fazer uma simulação. Acompanhe no algoritmo.

```
1: funcao SIMULP(int QTDQUES, int TAMBD)
2: int CTNAO ← 0
3: int CTSIM ← 0
4: int I ← 1
```

```
5: int P1, P2, J, K, ACHOU
6: enquanto I < QTDQUES
7:   P1 ← 50 aleatorios entre 1 e TAMBD
8:   P2 ← 50 aleatorios entre 1 e TAMBD
9:   ACHOU ← 0
10:  para J = 1 ate 50
11:    para K = 1 ate 50
12:      se P1[J] == P2[K]
13:        ACHOU ← 1
14:        J ← K ← 51
15:      fim{se}
16:    fim{para}
17:  fim{para}
18:  se ACHOU == 1
19:    CTSIM++
20:  senão
21:    CTNAO++
22:  fim{se}
23:  I++
24: fim{enquanto}
25: escreva CTNAO, CTSIM
```

 Para você fazer
Simulação Porta da Esperança

1. Rode a simulação acima 100.000 vezes usando pelo menos 2 computadores e 2 linguagens (compiladores) diferentes. Escreva a seguir quais as somas para os 4 valores obtidos:
- | | |
|--------------------|--|
| Não mudou e ganhou | |
| Não mudou e perdeu | |
| Mudou e ganhou | |
| Mudou e perdeu | |

2. Escreva uma conclusão a respeito da simulação. Um sujeito comum, ao participar do programa, deve ou não deve mudar de porta, e porquê. Finalmente diga quem estava certo: a colunista von Savant ou seus detratores.
- | | | |
|--|--|--|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

Questões no BD

1. Rode 10.000 vezes a simulação acima, cada vez com um tamanho de BD, até chegar a respostas no entorno de 0.01t e 0.99t, (1%) onde t é o tamanho que você estipulou. Achado este tamanho, rode a simulação 1.000.000 de vezes com este tamanho. Transcreva aqui os números encontrados:
- | tam. BD | qtd SIM | Qtd NAO |
|---------|---------|---------|
| | | |
2. Obviamente o número acima é (e de fato foi na ocasião) considerado inaceitável, pelo tamanho. Na época isso forçou um relaxamento no critério. Vai-se fazer o mesmo aqui. Reafaça os testes para uma probabilidade de 5%.
- | tam. BD | qtd SIM | Qtd NAO |
|---------|---------|---------|
| | | |
3. E para 15%
- | tam. BD | qtd SIM | Qtd NAO |
|---------|---------|---------|
| | | |
4. E, finalmente, para 30%.
- | tam. BD | qtd SIM | Qtd NAO |
|---------|---------|---------|
| | | |

O problema da parada segundo [McC12]

Já vimos inúmeros belos algoritmos que transformam um bando de peças de metal em gênios que fogem da lâmpada para produzir maravilhas. A pergunta que fica é: será que existe alguma coisa que os computadores não podem nem poderão (nunca!) fazer? Note que não se fala do presente: é claro que existem inúmeras tarefas hoje que os computadores não conseguem fazer: traduzir idiomas corretamente, dirigir um carro entre milhares com segurança, corrigir provas... entre muitas outras coisas. Tais problemas poderão ter alguma solução. Mas a pergunta persiste: em 1.000.000 de anos, haverá ainda algum problema insolúvel?

A resposta é um SIM retumbante. Há problemas que nunca serão resolvidos por computadores. São conhecidos como problemas não computáveis. Tais problemas já eram conhecidos antes mesmo da invenção dos computadores. Dois matemáticos, o americano Alonzo Church e o inglês Alan Turing os identificaram. Vamos seguir os passos que levaram a esta descoberta.

A confiabilidade do software O software melhorou muito nos últimos anos, mas ninguém assumiria que um windows qualquer nunca irá gerar uma tela azul (preta) da morte. Mas quem usou computadores nas décadas de 80 e 90 certamente vai concordar: eles melhoraram muito. Uma explicação é o uso cada vez mais frequente de ferramentas automáticas de verificação. Tais ferramentas (software) analisam outro programa e podem detectar erros potenciais por mais obscuros que sejam.

Daqui pode-se perguntar se não seria possível um super programa verificador certificar com 100% de certeza que ele não conteria erros. A prova de que tal verificador é impossível surge com o que é chamado de prova por contradição. Embora seja uma ferramenta matemática ela é usada o tempo todo por nós. Vamos a um exemplo, mas antes precisamos aceitar 2 fatos:

- A fundação de Curitiba ocorreu na década de 1690.
- Baltazar Carrasco dos Reis foi um dos fundadores (junto com Mateus Martins Leme e Heleodoro Ébano Pereira)

Agora suponha a afirmação *Baltazar Reis nasceu no século XX*. A afirmação é Verdadeira ou Falsa? A solução do problema passa por um raciocínio similar a

- Ninguém vive mais de 150 anos
- Portanto se ele fundou Curitiba, deve ter morrido – o mais tardar – em 1840
- Concordamos que 1840 é 60 anos antes do século 20, portanto

É falso que ele nasceu no século XX. Porque é possível afirmar essa falsidade? Simplesmente porque ela contradiz algum fato verdadeiro. Resumidamente (e matematicamente) a coisa funciona assim: Se $T \Rightarrow P$ ou garante-se que se T for verdadeira, P o será também. Mas sabe-se que P é falsa, o que nos permite garantir que T é falsa também (*modus tolens* é o nome técnico da coisa).

Programas analisando programas

Programas sempre seguem suas instruções literalmente. Assim, pode-se afirmar – sendo determinísticos – sempre gerarão a mesma saída. Certo? ou Errado? Não temos informação ainda para responder. A maioria dos programas que executamos geram saídas diferentes a cada vez. Por exemplo compare `winword alfa.doc` vis-a-vis `winword` ou ainda com `winword beta.txt`. Nos 3 casos o mesmo programa vai gerar uma tela inicial diferente. Para simplificar a discussão, vamos combinar que um programa lê um único arquivo de entrada e gera como saída uma tela. Daqui, pode-se rodar qualquer programa com qualquer arquivo de entrada. Obviamente pode-se obter resultados inesperados ao abrir um programa com um arquivo que não se destina a ele. Exemplos `winexcell foto.jpg` ou então `winword gama.xls`. Fica esquisito, mas funciona. Levando a coisa um nível adiante e fazendo `winword winword.exe`. O objetivo aqui é familiarizá-lo com coisas esquisitas (mas possíveis) de fazer com algum programa.

Programas sim-não Para manter as coisas simples, vamos estudar uma classe de programas muito simples (e chata) que só podem ter duas respostas *sim* e *não*. Começemos com o programa `A.exe` que simplesmente gera uma caixa de diálogo com a palavra *SIM* dentro. Igualmente, escreve-se o programa `B.exe` que gera uma caixa com a palavra *NÃO*. Eles não usam entradas (se entrada houver, eles a ignorarão). Um exemplo mais interessante é `SizeChecker.exe` que lê um arquivo de entrada e se a entrada for maior que 10Kbytes imprime *SIM* e caso contrário imprime *NÃO*. Este segundo exemplo é um tipo de programa que gera ambas as saídas a depender de alguma condição. Supondo que `SizeChecker.exe` tem cerca de 8Kbytes, chamado `SizeChecker.exe` a resposta será *NÃO*.

Nosso próximo programa é `AlwaysYes.exe` que examina a entrada e responde *SIM* se a entrada for um programa que sempre responde *SIM*. Caso contrário, `AlwaysYes` responde *NÃO*. Note que ele sempre funciona bem, se você chamá-lo `AlwaysYes windowd.exe` ele responde *NÃO* assim como em `AlwaysYes texto.txt` igualmente volta *NÃO*. Se chamado com um programa como `AlwaysYes SizeChecker.exe` ele vai responder *NÃO*.

Freeze.exe Um novo exemplo, de um programa que congela (tela azul da morte) toda vez que é executado. Um jeito fácil de programá-lo é incluir uma instrução de divisão por zero. Outra possibilidade pode ser (em C)

```
int main(){
    int a = 0;
    while (1==1) {
        a=a+1;
        a=a-1;
    }
}
```

Pode ser necessário (depende da arquitetura ou plataforma sendo usada) matar o programa ou até reiniciar o computador. Em um notebook pode ser necessário até remover as baterias... Espere-se aqui que `AlwaysYes Freeze.exe` deve responder *NÃO*.

Um novo programa agora é `YesOnSelf.exe`: ele é parecido com `AlwaysYes.exe` mas mais simples. Em vez de produzir *SIM* sempre que a entrada dele produz *SIM*, ele analisa o que acontece quando sua entrada é aplicada sobre si mesma. Por exemplo, se fizer `YesOnSelf SizeChecker.exe` o programa vai fazer alguma análise de o que acontece quando `SizeChecker` atua sobre a entrada `SizeChecker.exe`. Neste caso já se viu acima que a saída é *SIM* e portanto `YesOnSelf` vai responder *SIM* também. Se a entrada não é um programa *SIM-NÃO* a saída aqui é *NÃO*. A pergunta agora é o que acontece quando `YesOnSelf YesOnSelf.exe`? Como só há duas possibilidades, pode-se estudar cada uma delas.

SIM é perfeitamente consistente com o que se descreveu, mas antes de comemorar, pense ele respondendo *NÃO*. Isto significaria que `YesOnSelf` deveria responder *NÃO* quando executado em si mesmo. Continua sendo coerente, então parece que `YesOnSelf` pode escolher o que quiser e sua resposta estará correta. Esta liberdade misteriosa no comportamento será daqui a pouco a ponta de um enorme iceberg, mas vamos adiar a discussão.

Outra coisa é que nem `YesOnSelf` nem o mais complicado `AlwaysYes` foi realmente escrito, apenas o seu comportamento foi descrito. Mas, se for possível escrever `AlwaysYes`, então será barba para criar `YesOnSelf`. Porque? Porque `YesOnSelf` é mais simples do que `AlwaysYes`. Ele só tem que examinar uma entrada possível, ao invés de todas as entradas possíveis.

Vamos examinar mais um programa `AntiYesOnSelf.exe`. Como o nome sugere é muito semelhante a `YesOnSelf`. Na verdade é idêntico, mas com suas saídas invertidas. Se `YesOnSelf.exe` responder *SIM* então `AntiYesOnSelf.exe` responderia *NÃO* e vice-versa.

Lembrando, `YesOnSelf` vai dizer *SIM* se quando executado com ele mesmo de entrada disser *SIM* e *NÃO* caso contrário. `AntiYesOnSelf` diz *NÃO* se sua saída seria *SIM* quando executado em si.

O ponto é que programas podem fazer algo diferente de *SIM* ou *NÃO*. Logo, se alguém nos diz que de um programa não sai *SIM* não se pode concluir que então sai *NÃO*. Ele pode gerar lixo, ou mesmo congelar. Esta descrição está em *9 algorithms that changed the world*, de John MacCor-mick.

O problema da parada segundo [For12]

O problema da parada pergunta se é possível construir um programa *A* que leia outro programa *B* e decida ou seja, responda 1, se *B* termina e responda 0 se *B* não termina.

Para responder a esta pergunta vai-se usar a *redução ao absurdo* em uma demonstração de 3 etapas.

1. Vai-se então supor que este programa existe e ele se chama *MAGICO*. Então *MAGICO* lê e analisa o programa *X* respondendo 1 se *X* termina e 0 se *X* não termina.
2. Agora vai-se construir um segundo programa chamado *SUPER*, cuja estrutura é

```
int SUPER(programa X){
    res = MAGICO(X)
    while (res) {
        c++
        c--
    }
}
```

Note o funcionamento de *SUPER*. Lendo o programa *X*, se *X* termina, *SUPER* não termina. Se *X* não termina, *SUPER* termina.

3. Finalmente, vai-se rodar *SUPER(SUPER)*, isto é o programa *SUPER* vai ser alimentado com ele mesmo. Isto é legítimo, já que não se impôs nenhuma restrição no tipo de programa que pode ser lido. Olhe que resultado estranho:

 - se *SUPER(SUPER)* termina, então *SUPER(SUPER)* não termina
 - se *SUPER(SUPER)* não termina, então *SUPER(SUPER)* termina

Essa contradição nos leva a considerar que a premissa inicial (o programa *MAGICO* existe) era falsa e portanto demonstra-se que **Não é possível construir um programa que analise outro e decida se este outro vai ou não encerrar**. Em termos técnicos, diz-se *O problema da parada não é solúvel*. Palmas para Alan Turing.

Uma maneira mais fácil de entender esta coisa é imaginar que Curitiba é uma pequena aldeia que tem apenas um barbeiro. Ele é definido como o profissional que faz a barba dos homens da aldeia que não se barbeiam sozinhos.



O barbeiro da aldeia, que barbeia todos os homens que não se barbeiam sozinhos, se encontra numa situação absurda: sua definição não permite que ele se barbeie sozinho e, ao mesmo tempo, proíbe-o de não se barbear.

Ilustração de *Matemática Moderna*, de Walter Fuchs.

Para você fazer

Analise qualquer uma das 3 descrições acima e convença-se de que matematicamente, *o barbeiro de Curitiba não pode existir*.



Processamento Fonético

Enquanto os códigos são números (com seus correspondentes dígitos verificadores) é fácil aos sistemas computacionais localizar referências e conteúdos armazenados.

Entretanto, nem sempre é possível construir os sistemas baseados em códigos numéricos. O ser humano ao nascer recebe um nome e não um número (benza Deus !). Inúmeras referências no mundo são a nomes. Este fato complica as coisas: nomes dependem de idiomas, ou seja a cada pedacinho do mundo são escritos e falados de maneira diferente. Pior ainda, nomes escritos diferentes são ditos de mesma maneira e também o contrário: nomes ditos diferentemente são escritos iguais.

Outra dificuldade é que a transmissão de nomes é pela fala, e o uso do sistema falante, e do conseqüente sistema ouvinte introduz nova fonte de ruídos.

Na tentativa de amenizar (nunca resolver) este problema, surgiram há muitos anos os sistemas de processamento fonético. Tais sistemas, derivados da fonologia e da antropologia tentam registrar primeiro as falas a partir de seus fonemas e depois mediante algumas substituições e/ou processamentos locais obter chaves alfanuméricas que possam ser usadas na busca de conteúdos dentro de computadores.

A idéia é agregar aos bancos de dados, junto com a chave primária (alfabética), uma outra chave formada pela transformação fonética da chave primária. A busca é feita primeiro na chave primária básica. Se o registro é encontrado, segue-se o processamento normal, como se fosse um sistema SEM processamento fonético.

A diferença surge quando um registro não é localizado (NOT FOUND). Nos sistemas convencionais, sinaliza-se um erro, e adiante, vamos em frente.

Nos sistemas com processamento fonético, antes de dar o erro, o sistema tenta ajudar o operador. Calcula a transformada fonética da chave, e busca no arquivo usando a chave secundária (fonética) registros que tenham o mesmo valor para ela. Encontrando um ou alguns, mostra-os para o operador, perguntando: *É algum desses o que você está procurando ?...*

Soundex

O algoritmo SOUNDEX foi projetado em 1918 por dois pesquisadores da área de fonética. Ele transforma um fonema em uma de seis classificações fonéticas: bilabial (BP), labiodental (FV), dental (z em espanhol), alveolar (r em prato), velar (a língua encosta no véu palatino) e glotal (oclusão da glote = comum no árabe). Esta classificação está baseada na configuração da boca, lábios e língua ao formar os sons.

Naturalmente, um problema deste algoritmo é sua configuração ao idioma (aos fonemas) da língua inglesa, mas como a máquina produtora de sons é a mesma para qualquer pessoa em qualquer lugar, é possível aproveitar o algoritmo.

Para cada palavra do texto de entrada, o SOUNDEX gera um código de 4 caracteres (1 letra + 3 dígitos) e a esperança é que palavras foneticamente iguais ou parecidas tenham o mesmo soundex.

1. Transforme todas as letras em maiúsculo. Como os fonemas são os mesmos (tanto para minúsculo como para maiúsculo) essa conversão pode ser feita sem perda de substância.
2. Retire todos os caracteres estranhos, deixando apenas letras. Uma especial atenção deve ser dada aos números, que em geral são retirados também, a menos que eles sozinhos constituam uma palavra.
3. Individualize cada uma das palavras do nome. Para cada palavra: Retire e guarde a primeira letra da palavra. Ela é o primeiro caractere do código SOUNDEX.
4. Transforme as ocorrências de
A, E, I, O, U, H, W, Y→0; B, F, P, V→1; C, G, J, K, Q, S, X, Z→2;
D, T→3; L→4; M, N→5; R→6.
5. Deve-se remover agora pares de dígitos iguais, um ao lado do outro.
6. Deve-se remover todos os zeros do código
7. Devolver 4 caracteres apenas (se sobram, deve-se truncar. Se faltarem, completar com zeros). Este é o código SOUNDEX.

Seja como exemplo o soundex do primeiro verso da belíssima poesia de Manoel Bandeira, “Vou me embora pra pasargada”.

VOU ME EMBORA PRA PASARGADA
LA SOU AMIGO DO REI
LA TENHO A MULHER QUE EU QUERO
NA CAMA QUE ESCOLHEREI

V000 M000 E516 P600 P262
L000 S000 A520 D000 R000
L000 T500 A000 M460 Q000 E000 Q600
N000 C500 Q000 E246

Algoritmo de Fonetização do Pedro Kantek

Usado em uma antiga (década de 80) aplicação de busca de endereços em Curitiba contra o cadastro de ruas da COPEL. Seu desempenho foi bem satisfatório. A fonetização dos nomes era apenas uma pequena parte do algoritmo, que estava fortemente baseada em tabela de sinônimos. Por exemplo, XV → QUINZE, DESEMBARGADOR → (nada) E, DE, DO, ... → (nada), etc.

1. Deve-se retirar todos os caracteres acentuados existentes, substituindo-os pelos equivalente universais. Assim
à → A; A-grave→A; á→A; é→E; í→I; ó→O; ú→U; A-agudo→A; Ê→E; Î→I; O-agudo→O; U-agudo→U; â→A; ê→E; ô→O; A-circ→A; E-circ→E; O-circ→O; ç→C; Ç→C; ã→A; A-til→A; õ→O; O-til→O; ù→U; Ü→U.
2. Todos os caracteres minúsculos devem ser convertidos a seus caracteres maiúsculos. Essa é a razão pela qual a conversão acima (a dos caracteres acentuados) já devolveu-os em maiúsculo.
3. Tratamento de sinônimos
4. Todas as palavras deverão passar pelo processo de fonetização. Este processo é uma busca e substituição de conjuntos de letras. A tabela de substituições era externa, para poder ser atualizada de maneira independente, e começava com as seguintes ocorrências
quaisquer letras dobradas, exceto SS e RR→uma única ocorrência, SCH→X; CH→X; ASA, ESE, ISI, OSO, USU→AZA, EZE, IZI, OZO, UZU; CA, CO, CU→KA, KO, KU; SSE, SSI→CE, CI; QUE, QUI→KE, KI; consoantes mudas, ao final de palavras→excluí-las; Y→I; OU→O; IE→I; W→V; HA, HE, HI, HO, HU→A, E, I, O, U;

O mesmo exemplo

VO ME EMBORA PRA PAZARGADA
LA SO AMIGO DO REI
LA TENO A MULE KE EU KERO
NA KAMA KE ESKOLEREI

Algoritmo Fonético New York

O código fonético dos sistemas do Estado de Nova Iorque, estabelecido em 1970 e conhecido como NYSIIS foi parte dos sistemas de Identificação e Inteligência dos serviços de Justiça Criminal. Segundo seus autores ele melhora o desempenho em 2.7% sobre o Soundex.

1. Traduza os primeiros caracteres do nome: MAC → MCC; KN → NN; K → C; PH → FF; PF → FF; SCH → SSS.
2. Traduza os últimos caracteres do nome EE → Y; IE → Y; DT, RT, RD, NT, ND → D.
3. Primeiro caracteres da chave = Primeiro caractere do nome
4. Traduza os caracteres remanescentes de acordo com: (um caractere de cada vez) EV → AF; depois todas as vogais (A, E, I, O, U) → A; Q → G; Z → S; M → N; KN → NN senão K → C; SCH → SSS; PH → FF; H → (prévia) se a prévia ou a seguinte é nãovogal; W → (prévia) se a prévia é vogal.
5. Adicione a letra corrente se a corrente não é igual ao caracter anterior da chave.
6. Se o último caractere é S, remova-o.
7. Se os últimos são AY, troque por Y.
8. Se o último é A, remova-o.

Para você fazer

Considere a seguinte linha de “Vou me embora...”

LA A EXISTENCIA E UMA AVENTURA

Escreva o soundex dessa linha
Escreva o pkan dessa linha

Considere outra linha do mesmo poema

QUE NO TEMPO DE EU MENINO

Escreva o soundex da segunda linha
Escreva o pkan da segunda linha

Pós Graduação: um projeto de vida

De início, duas citações: *Ser culto é o único modo de ser livre*, dito por José Martí, herói cubano na libertação da Espanha. Outra citação, escrita na fachada da Universidade de Goundishapour (atual Irã) em 400dC: *A caneta é mais poderosa do que a espada*. Para algumas análises numéricas que serão feitas à frente, considere a população brasileira como formada por _____ (dados de _____). Segundo o IBGE, mais ou menos 10% disso (ou _____) estão entre 20 e 24 anos.

Uma universidade é uma instituição de ensino, pesquisa e extensão que abrange todos os ramos do saber (do latim *universitas*=universo, universalidade). As primeiras universidades são do século XII na Europa. A primeira universidade brasileira é a URJ de 1920. O Ministério da Educação só surge em 1930, no bojo da Revolução de Getúlio Vargas. A revolução constitucionalista de São Paulo (1932) tem como um resultado importante a fundação da USP em 1934, a maior e melhor universidade brasileira. Em 1950, havia 10 universidades, em 1960 eram 20. Após a reforma universitária de 1968, este número cresceu lentamente até chegar a 65 em 1980. Destas, 43 eram públicas e 22 privadas).

Em 1983, a emenda Calmon mandou a União destinar 13% do orçamento e os estados e municípios destinarem 25% do orçamento em educação. A Constituição de 1988 aumentou este valor para 18%. Finalmente, a nova LDB de 23/12/96, alterou o panorama do ensino superior em nosso país, ao liberalizar a criação de cursos universitários de graduação: Em 2002 eram 1637 instituições de ensino superior em nosso País.

Quanto às matrículas, veja-se o crescimento

ano	matrículas
1990	1.540.000
1996	1.868.000
1999	2.369.000
2002	3.479.000
2005	4.450.000

Todas essas matrículas necessitam professores para dar conta. Até os anos 70, só se formavam professores fora do país. Como quase todos eram funcionários públicos, o Governo Federal criou uma repartição chamada CAPES (Comissão de Aperfeiçoamento do Pessoal de Nível Superior) especificamente para mandar professores estudarem fora do Brasil. Aos poucos essa CAPES passou a cuidar (e muito bem) dos cursos de mestrado e doutorado no país.

Em 1965, o Brasil organizou suas regras de pós graduação através do Relatório Sucupira. Os cursos foram divididos em *latu sensu*=especialização e *strictu sensu*=mestrado e depois doutorado. Nessa época entendia-se a pós graduação como formação de mão de obra de ensino. Desde essa época (e até hoje) ninguém controla os cursos de especialização. Qualquer um é livre para lançar e vender cursos de especialização. O que diferencia um mestrado de uma especialização (entre outras coisas) é a duração. Cursos de especialização têm até 360 h/aula.

Mestrado O mestrado é o primeiro degrau da pós graduação. Também é conhecido como quarto grau. Ele tem como pré-requisito uma graduação de no mínimo 2200 h. Demora uns 2 anos, e sua medida de andamento é o **crédito** que corresponde – grosso modo – a 15 horas aula. Após os créditos (em torno de 30), o aluno deve escrever uma dissertação, e defendê-la em sessão pública. O governo brasileiro oferece umas 30.000 bolsas/ano, que no caso do mestrado valem R\$ 1.200/mês por até 3 anos. Outro requisito exigido é a suficiência em inglês.

Mais recentemente, a partir de 1998, surgiram os cursos denominados mestrados profissionalizantes: embora tenham o mesmo arcabouço de qualidade dos mestrados e doutorados, não se dedicam a formar professores e sim profissionais para o mercado de trabalho convencional. Ainda não são significativos em nosso país.

Doutorado É o segundo degrau da pós graduação. Usualmente (mas nem sempre) tem como pré-requisito o mestrado. Também é medido em créditos (em torno de 60), mas usualmente os créditos do mestrado podem ser reaproveitados, caso eles tenham sido cursados na mesma área. Mas, o importante aqui é a tese de doutorado, que deve ser inovadora, e esta é uma característica que é difícil de atender. Além do inglês é necessária a suficiência em um segundo idioma. Antes de deixar o aluno começar a escrever a tese, a universidade exige um exame de qualificação, onde o aluno deve demonstrar proficiência na área da tese. A defesa é mais rigorosa, usualmente a banca tem no mínimo 6 professores, com vários externos à Universidade. A bolsa é de R\$ 1.800,00.

Os programas de pós graduação (M/D) são avaliados a cada 3 anos pela CAPES em notas que variam entre 1 (ruim) e 7 (excelente). Esta nota é importante pois rege a distribuição de bolsas aos alunos.

A seguir, alguns cursos que podem eventualmente interessar, lembrando que não é obrigatório seguir na mesma área onde se fez a graduação. A chamada multidisciplinaridade pode ser magnificamente alcançada fazendo um mestrado em área diferente de onde se fez a graduação, embora surjam dificuldades importantes aqui.

POSCOMP Um exame único para acesso a (quase todos) os cursos de PG na área de informática em nosso país. Cada curso/instituição usa este exame como quiser, mas sua existência certamente facilita a

tarefa de se inscrever em mais de um curso. As inscrições são feitas via internet e a prova ocorre em um único dia em todo o País. Neste ano vai ser dia ____/____/____. O custo é baixo, neste ano de R\$ _____. As inscrições são feitas em www.sbc.org.br/poscomp entre ____/____/____ e ____/____/____. No site acima, há vários exemplares das provas de anos passados que podem ser consultados. Finalmente, vale a regra de que esta prova pode ser feita na “moita”. Se o resultado for ruim, ninguém precisa ficar sabendo e se for bom, é um documento e tanto para o futuro do aluno.

UFPR-MI

nome curso:
site:
cidade:
nota capes e custo:
obs:

PUCPR-MI

nome curso:
site:
cidade:
nota capes e custo:
obs:

UFSC-MCC

nome curso:
site:
cidade:
nota capes e custo:
obs:

UFSC-CC

nome curso:
site:
cidade:
nota capes e custo:
obs:

UFSC-EEL

nome curso:
site:
cidade:
nota capes e custo:
obs:

UTFPR-CPGEI

nome curso:
site:
cidade:
nota capes e custo:
obs:

UFRGS-C

nome curso:
site:
cidade:
nota capes e custo:
obs:

USP-CC

nome curso:
site:
cidade:
nota capes e custo:
obs:

INPE

nome curso:
site:
cidade:
nota capes e custo:
obs:

Etapa 3

Análise das respostas da NASA e conclusão final

item	NASA justificativa	resp ind	erro	resp col	erro
caixa de fósforos	15	Não existe oxigênio na lua para sustentar a chama. Praticamente inúteis.			
comida concentrada	4	Meio eficiente de suprir as necessidades de energia			
corda de nylon de 20m	6	Usado para escalar picos e para manter membros feridos juntos			
paraquedas de seda	8	Proteção contra os raios solares			
unidade portátil de aquecimento (à base de luz solar)	13	Só é necessária no lado escuro da lua			
dois revólveres calibre 45	11	meios possíveis de auto-propulsão			
uma caixa de leite em pó	12	back-up da comida concentrada			
dois tanques de oxigênio de 100 libras	1	Maior necessidade para a sobrevivência			
mapa estelar com as constelações da lua	3	Meio primário de navegação			
balsa auto-inflável	9	O CO2 existente na garrafa de auto-enximento pode ser usado como propulsão			
bússola magnética	14	O campo magnético na lua não é polarizado. A bússola é inútil			
cinco galões de água	2	Reposição da perda significativa de água no lado iluminado			
sinal luminoso	10	Sinal de aviso, quando a nave mãe for localizada			
kit de agulhas de injeção para primeiros socorros	7	Agulhas para vitaminas, remédios etc já são colocadas em uma abertura especial das roupas pela NASA			
transmissor-receptor de FM movido a bateria solar	5	Para comunicação com a nave-mãe. Entretanto o FM exige linha reta entre transmissor e receptor e pequenas distâncias.			

Aqui, os erros devem ser a diferença em valor absoluto (o maior menos o menor) cometido entre a NASA e a resposta individual e depois entre a NASA e a resposta coletiva.

Ao final, totalizar os 2 erros e concluir.

Perdidos na lua

O objetivo desta atividade é comprovar na prática como as decisões coletivas tendem a ser melhores do que as decisões individuais. Este exercício foi retirado e adaptado do livro *How to make meetings work* de DOYLE, Michael e STRAUS, David. O exercício deve ser feito em 3 etapas.

- Usando a tabela a seguir, primeiro você deve responder ao questionário solicitado, DE MANEIRA INDIVIDUAL.
- A seguir, o mesmo questionário deve ser respondido em grupo. Todos os participantes precisam concordar em dar uma resposta ÚNICA na sala. O mecanismo usado para este preenchimento, fica a critério do grupo (votação ? – consenso ?? – quem gritar mais leva ? – DECIDAM).
- Somente depois que as duas respostas estiverem lançadas (a primeira individual, cada um responde como quiser. A segunda, coletiva, todos com a mesma resposta), o questionário deve ser corrigido vis-a-vis com as respostas oficiais da NASA.
- Agora, deve-se medir o erro cometido na resposta individual e na resposta coletiva.
- Finalmente, deve-se comparar a soma dos erros individuais comparada com a soma dos erros coletivos.

Etapa 1-Folha individual de respostas

Sua espaçonave se destruiu ao alunisar. Você precisa planejar seu encontro com sua espaçonave-mãe, 350 Km distante através da superfície iluminada da lua. Seu jeep lunar também foi destruído. Só sobraram os 15 itens listados abaixo.

Sua sobrevivência depende de você encontrar a espaçonave-mãe, e você deve escolher os itens mais importantes para a viagem de 350 Km.

Sua tarefa é atribuir pesos de importância a cada um os itens abaixo. Coloque o número um no item mais importante, o dois no seguinte e assim por diante até o número 15 no menos importante.

Lembre-se que o mais importante é o 1, o segundo é o 2 e assim por diante. O menos importante é o 15.

1. item	2. prioridade individual	3. coletiva
caixa de fósforos		
comida concentrada		
corda de nylon de 20 metros		
paraquedas de seda		
unidade portátil de aquecimento (à base de luz solar)		
dois revólveres calibre 45		
uma caixa de leite em pó		
dois tanques de oxigênio de 100 libras		
mapa estelar com as constelações da lua		
balsa auto-inflável		
bússola magnética		
cinco galões de água		
sinal luminoso		
kit de agulhas de injeção para primeiros socorros		
transmissor-receptor de FM movido a bateria solar		

Etapa 2 - Coletivo

Preenchimento coletivo por consenso da coluna 3