Exercícios em Python e mais

Pedro Luis Kantek Garcia Navarro
UFPR

17 de dezembro de 2021

Sumário

O autor, pelo autor

Meu nome é Pedro Luis Kantek Garcia Navarro, conhecido como Kantek, ou Pedro Kantek. Nasci em Curitiba há mais de 60 anos. Sou portanto brasileiro, curitibano e coxa-branca com muito orgulho, mas sendo filho de espanhóis (meus 7 irmãos nasceram lá), tenho também a nacionalidade espanhola. Aprendi a falar em *castellano*, era o que se falava na minha casa, o português é meu segundo idioma, só visto na escola. Estudei no Colégio Bom Jesus e quando chegou a hora de escolher a profissão, lá por 1972, fui para a engenharia civil, mas sem muita convicção. As alternativas: medicina (arghhh!) ou direito (arghhh! arghhh!).

Durante a copa do mundo de futebol de 1974 na Alemanha, ao folhear a Gazeta do Povo, achei um pequeno anúncio sobre um estágio na área de processamento de dados (os nomes informática e computação ainda não existiam). Lá fui eu para um estágio na CELEPAR, que acabou mais de 35 anos depois. Na CELEPAR fui de tudo: programador, analista, suporte a BD (banco de dados), suporte a TP (teleprocessamento), coordenador de auto-serviço, coordenador de atendimento, ... Minha carreira lá terminou na área de governo eletrônico. Desde cedo encasquetei que uma boa maneira de me obrigar a continuar estudando a vida toda era virar professor. Comecei essa desafiante carreira em 1976, dando aula num lugar chamado UUTT, que não existe mais. Passei por FAE, PUC e cheguei às Faculdades Positivo em 1988. Na década de 80, virei instrutor itinerante de uma empresa chamada CTIS de Brasília, e dei um monte de cursos por este Brasil afora (Manaus, Recife, Brasília, Rio, São Paulo, Fortaleza, Floripa, ...). Em 90, resolvi voltar a estudar e fui fazer o mestrado em informática industrial no CEFET. Ainda peguei a última leva dos professores franceses que iniciaram o curso. Em 93 virei mestre, e a minha dissertação foi publicada em livro pela editora Campus (Downsizing de sistemas de Informação. Rio de Janeiro: Campus, 1994. 240p, ISBN:85-7001-926-2). O primeiro cheque dos direitos autorais me manteve um mês em Nova Iorque, estudando inglês. Aliás, foi o quarto livro de minha carreira de escritor, antes já havia 3 outros (MS WORD - Guia do Usuário Brasileiro. Rio de Janeiro: Campus, 1987. 250p, ISBN:85-7001-507-0, Centro de Informações. Rio de Janeiro: Campus, 1985. 103p, ISBN:85-7001-383-3 e APL - Uma linguagem de programação. Curitiba. CELEPAR, 1982. 222p). Depois vieram outros. Terminando o mestrado, rapidamente para não perder o fôlego, engatei o doutorado em engenharia elétrica. Ele se iniciou em 1994 na UFSC em Florianópolis. Só terminou em 2000, foram 6 anos inesquecíveis, até porque nesse meio tive que aprender o francês - mais um mês em Paris aprendendo-o. Finalmente virei engenheiro, 25 anos depois de ter iniciado a engenharia civil. Esqueci de dizer que no meio do curso de Civil desistí (cá pra nós o assunto era meio chato...) em favor de algo muito mais emocionante: matemática. Nessa época ainda não havia cursos superiores de informática. Formei-me em matemática na PUC/Pr em 1981. Em 2003, habilitei-me a avaliador de cursos para o MEC. Para minha surpresa, fui selecionado e virei delegado do INEP (Instituto Nacional de Pesquisas Educacionais) do Governo Brasileiro. De novo, visitei lugares deste Brasilzão que sequer imaginava existirem (por exemplo, Rondonópolis, Luiziânia, Rio Grande, entre outros), sempre avaliando os cursos na área de informática: sistemas de informação, engenharia e ciência da computação. Atualmente estou licenciado da PUC. Já fiz um bocado de coisas na vida, mas acho que um dos meus sonhos é um dia ser professor de matemática para crianças: tentar despertá-las para este mundo fantástico, do qual – lastimavelmente – boa parte delas nunca chega sequer perto ao longo de sua vida. Em 2018, comecei a dar aulas na UFPR.

O que é

Este é um curso de programação baseado em tarefas (programas com grau crescente de complexidade) e baseado em Python.

O autor, há muitos anos, quando ainda havia cabelo...

 ©75-21, Pedro Kantek

Capítulo 1

A título de explicação

Neste material a maioria dos códigos é em Python e está escrita em azul. Mas, alguma coisinha está em C (ou C++) e neste caso ela aparece em marrom. Menos ainda, mas ainda assim presente está o APL, que aparece em vermelho. Execuções de programas que independem de qual linguagem o produziu, aparecem em verde. Mais para o fim do texto, surgem códigos Python importados diretamente do repositório de códigos Euler. Estes aparecem em preto.

Capítulo 2

Exercícios

2.1 Salada de frutas

Uma salada de frutas, produzida industrialmente leva maçã, banana, abacaxi e açúcar nas proporções de 1 Kg : 2 Kg : 0,5 Kg : 0,5 Kg respectivamente. Escreva um programa que leia 4 quantidades representando os pesos disponíveis dos quatro ingredientes, calcule e escreva a quantidade de salada que poderá ser construida, desprezando-se as quantidades que ultrapassam a proporção.

Exemplos:

Se for lido	só serão usadas	logo total
10,10,10,10	5,10,2.5,2.5	20Kg
10,20,10,10	10,20,5,5	40Kg
10,10,1,1	2,4,1,1	8Kg

Ao estudar o problema e os exemplos, verifica-se que quando há 10 kg de cada ingrediente (exemplo 1) não se pode fazer 40Kg de salada, pois ao fazer isso deixa-se de observar as proporções dadas no enunciado. De fato, o exemplo aponta 20 e não 40 kg de salada.

Examinando o enunciado, ve-se que para fazer 4kg de salada, (algo que se poderia chamar de 1 módulo de salada) usam-se 1kg, 2kg, 0.5kg e 0.5kg de ingredientes respectivamente. A pergunta agora é: com os valores disponíveis de ingredientes quantos módulos de salada podem ser feitos ?

A resposta, ainda no exemplo 1, é de limitados à maçã, podem ser feitos 10 módulos. Limitados pela banana, são apenas 5 módulos, e pensando no abacaxí e no açucar seriam 20 módulos. Para manter as proporções corretas, deve-se usar o menor valor dos módulos, no caso 5. Este valor esgota o ingrediente crítico (a banana neste exemplo) e tentar produzir mais salada aqui, significaria que faltaria banana.

Decidido a produzir 5 módulos de salada, o peso final é imediato. 5 módulos vezes 4 kg de salada por módulo correspondem a 20kg de salada, que é o resultado correto dado pelo exemplo.

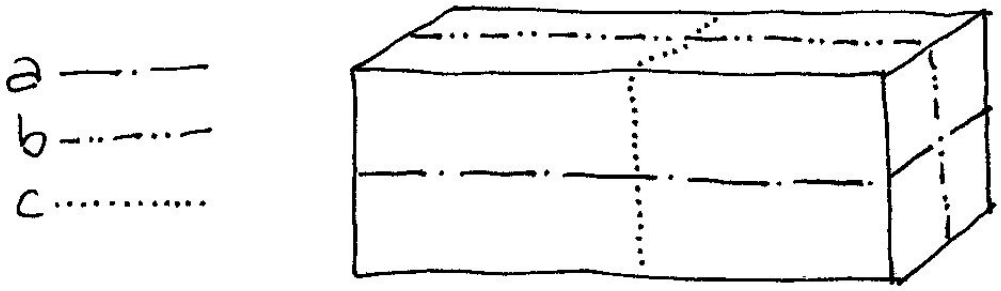
Eis o programa

```
def salada():
  print("Informe as disponibilidades de maçã, banana, abacaxí e açuçar")
  dma=float(input())
  dba=float(input())
  dab=float(input())
  dac=float(input())
  modu=[dma/1, dba/2, dab/0.5, dac/0.5]
  minimo=min(modu)
  print("A quantidade possível é ",minimo*4)
salada()
```

2.2 pacote no correio

De acordo com o regulamento do serviço postal de um país distante, não pode seguir pelo correio, nenhum pacote cujo comprimento (maior dimensão) somado à amarração seja superior a 72 polegadas. Amarração é o comprimento do menor barbante que possa toda a volta ao pacote. Construa um programa que leia 3 dimensões de um eventual pacote paralelepipedal (um paralelepípedo é um prisma cujas faces são paralelogramos. Um paralelepípedo tem seis faces, sendo que duas a duas são idênticas e paralelas entre si). e imprima ”SIM”se ele puder ser enviado e ”NÃO”senão. As dimensões estão em centímetros e deve usar a conversão 1pol=2.54cm

Ao estudar a definição, percebe-se que há 3 barbantes possíveis em um paralelepípedo. No desenho



percebe-se que os 3 barbantes são: $2 \times a + b$, $2 \times a + c$ e $2 \times b + c$. Então o teste deve ser feito somando-se a maior dimensão ao menor barbantes e esta soma deve ser menor do que 72 polegadas.

Como os dados do pacote estão em centímetros, a conta toda vai ser feita em centímetros e apenas no momento da compraração é que este valor vai ser convertido a polegada.

Veja o programa:

```
def correio():
  print("Informe as dimensões do paralelepípedo")
  a=float(input())
  b=float(input())
  c=float(input())
  barba=[2*(a+b), 2*(a+c), 2*(b+c)]
  menor=min(barba)
  lista=[a,b,c]
  maior=max(lista)
  dime=(menor+maior)/2.54
  if dime<=72:
    print("sim")
  else:
    print("nao")
correio()
```

2.3 Jogo do Palito

Este é o nosso primeiro vídeo-game. Trata-se de uma disputa entre computador e humano para ver quem ganha. O jogo começa com o humano escolhendo um número de palitos entre 20 e 30. Depois o computador tira 1, 2 ou 3 palitos. A seguir o humano faz o mesmo e assim por diante até o esgotamento dos palitos. Quem tirar o último palito perde.

Eis o código

```
def palito():
  n=0  # seq vencedora: 1, 5, 9, 13, 17, 21, 25, 29
  while n<20 or n>30:
    n=int(input("com quanto começamos ?"))
  while n>0:
    seq=n%4
    if seq==1:
      j=1 #perdi, jogo 1 esperando o cara errar...
    if seq==2:
      j=1
    if seq==3:
      j=2
    if seq==0:
      j=3
    print("eu joguei", j)
    n=n-j
    print("existem", n, " palitos")
    if n<=0:
      lixo=input("por incrivel que pareça, perdi...")
      return
    tv=0
    while tv<1 or tv>3 or n-tv<0:
```

```
      tv=int(input("jogue: "))
    n=n-tv
    print("Existem ",n," palitos")
    if n<=0:
      lixo=input("seu babaca, eu ganhei")
      return
palito()
```

Se quiser comparar com a versão C++, ei-la:

```
#include <iostream>
#include<cstdlib>
using namespace std;
main() {
   int n,seq,j,tv,lixo;
   n=0;
   while ((n<20) || (n>30)) {
 cout<<"Com quanto comecamos ?"<<endl;
 cin>>n;
   }  // seq vencedora: 1, 5, 9, 13, 17, 21, 25, 29
   while(n>0){
      seq = n % 4;
      if (seq==1) {
 j=rand()%3+1;  // perdi, jogo aleatório esperando ele errar
      }
      if (seq==2) {
 j=1;
      }
      if (seq==3) {
  j=2;
  }
      if (seq==0) {
  j=3;
  }
      cout<<"eu joguei "<<j<<endl;
      n=n-j;
      cout<<"existem "<<n<<" palitos \n"<<endl;
      if (n <= 0) {
  cout<<"Por incrivel que pareca, perdi"<<endl;
  cin>>lixo;
  break;
      }
      tv=0;
      while ((tv<1) || (tv>3) || ((n-tv)<0)) {
  cout<<"jogue"<<endl;
  cin>>tv;
      }
    n=n-tv;
    cout<<"existem "<<n<<" palitos "<<endl;
    if (n<=0) {
       cout<<"babaca, eu ganhei\n";
       cin>>lixo;
       break;
      }
   }

}
```

2.4 MDC e MMC

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

 ©75-21, Pedro Kantek

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 est˜ão 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. Acompanhe: Se $a = 27, b = 100$, $MDC(27, 100)$ é trocado por $MDC(100, 27\%100)$, sendo que o segundo operando é igual a 27, já que a divisão de 27 por 100 é igual a zero, com resto igual a 27.

Este algoritmo é recursivo, já que a função MDC no seu corpo tem uma chamada à função MDC, sempre com um caso menor em direção ao caso base, que aqui é $b == 0$.

1: inteiro função MDC(inteiro a, b)
2: **if** b = 0 **then**
3: devolva a
4: **else**
5: devolva MDC(b,(a mod b))
6: **end if**

Suponha que far-se-á a chamada ao algoritmo acima, como segue: MDC(1200, 1119) O 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

Eis o programa python

```
def mdc(a,b):
  if b==0:
    return a
  else:
    return mdc(b,a%b)
def exercicio():
  a=int(input("Informe a "))
  b=int(input("Informe b "))
  x=mdc(a,b)
  print(x,(a*b)//x)
exercicio()
```

2.5 Busca

Este é o problema fundamental da ciência da computação, a saber: Dado um vetor V e uma chave k, quer-se saber se $k \in V$ e se isto ocorrer em que posição, ou se – ao contrário – $k \notin V$. Reespecificando melhor o problema vamos supor um vetor de inteiros $int\ V[1000]$ em C++ ou um array $V = nd.array(1000, int)$ em python ou ainda em APL: $V \leftarrow 1000\rho 0$. Independente da linguagem trata-se de um arranjo unidimensional homogêneo de inteiros, no qual se quer saber se existe ou não um determinado valor também inteiro. Existindo deve-se retornar qual a sua posição. Se houver mais de uma instância do valor k, deve-se retornar apenas a primeira. Se não houver, deve-se retornar um valor inválido para sua posição, tipicamente -1, significando que k não existe em V.

Eis uma primeira versão simples do algoritmo

```
def busca(lis,k):
  i=0
  while i<len(lis):
    if lis[i]==k:
      return i
    i=i+1
  return -1
print(busca([1,7,33,4,21,7],121))
```

Uma possível melhora no desempenho do algoritmo é conseguida mantendo a lista (ou o vetor) em ordem. Agora, a conclusão pela inexistência não precisa ir até o final da lista podendo parar no exato ponto em que a chave buscada deveria estar, se lá estivesse.

```
def busca(lis,k):
  lis.sort()
```

```
  i=0
  while i<len(lis)and lis[i]<=k:
    if lis[i]==k:
      return i
    i=i+1
  return -1
print(busca([1,17,33,4,21,7],14))
```

Repare que ao ordenar a lista acima, ela fica [1,4,7,17,21,33] e nela a busca pelo número 14, não precisa ir até o final, podendo parar ao encontrar o 21.

Outra estratégia é o uso de uma sentinela. Sentinela é um truque bastante usado em programação e ele faz analogia com o funcionário de uma empreiteira arrumando uma estrada. Ele sinaliza com uma bandeira alguma condição.

Aqui é igual: a sentinela é um valor qualquer que sinaliza alguma condição específica no algoritmo. Neste caso, a sentinela vai ser o valor k que está sendo buscado. Este valor vai ser incluído no final da lista (Daí porque esta técnica não é usada com listas ordenadas). Agora, o valor k vai ser encontrado por bem ou por mal, e este fato diminui um teste que não precisa mais ser feito. O ponto agora é onde o k é encontrado: Se antes do final, ele já existia na lista e este fato pode ser reportado. Se o valor k encontrado é aquele que acabamos de colocar no final da lista, então a conclusão é de que ele não existia anteriormente na lista.

Eis o algoritmo

```
def busca(lis,k):
  i=0
  lis.append(k)
  while lis[i]!=k:
    i=i+1
  if i==len(lis)-1:
    return -1
  return i
print(busca([1,17,33,4,21,7],21))
```

Até agora os custos associados a cada busca estão a seguir descritos. Considere-se o número de testes para localizar uma chave em um vetor de 1000 elementos (na média):

método	custo	acessos para 1000 chaves
busca linear positiva	O(n)	500*2
busca linear negativa	O(n)	1000*2
busca ordenada positiva	O(n)	500*3
busca ordenada negativa	O(n)	500*3
busca sentinela positiva	O(n)	500
busca sentinela negativa	O(n)	1000

Agora vai-se usar uma abordagem inovadora que é buscar um custo de busca de $O(1)$ muito mais rápido sob qualquer comparação que se faça.

A idéia é inventar uma função matemática qualquer que recebendo k devolva a posição i onde a chave estará (se $k \in V$) ou ela onde deveria estar. Busca-se o elemento indicado por i e se ele for k a busca foi bem sucedida. Se ele não for k a busca foi mal sucedida. Em ambos os casos gasta-se $O(1)$.

Como não existe almoço grátis, esta estratégia impõe como exigência a impossibilidade de recuperar a lista na sua ordem original, além de um overload de carga de processamento para tratar as colisões.

Para discutir esta função de mapeamento (hash), é preciso estabelecer 2 conceitos:

- O espaço de chaves que representa todos os possíveis valores da chave k. Num exemplo hipotético, se estivermos falando do CPF como chave este espaço é de 000.000.000 a 999.999.999 ou 1 bilhão.
- O espaço de endereços que representa os espaços reservados no computador para guardar os dados. Numa universidade com 20.000 alunos este espaço poderia ser de 50.000.

Como tradicionalmente o espaço de chaves é sempre muito maior do que o espaço de endereços a função estudada precisa comprimir o primeiro no segundo. Neste processo de compressão pode ocorrer (e sempre ocorre) um fenômeno chamado **colisão**: duas chaves distintas geram o mesmo endereço na área de dados. Este fato precisa ser tratado e sempre impõe algum *overhead* ao processamento.

No nosso exemplo pedagógico vamos usar o CPF para acessar um espaço no qual será reservado espaço para 1000 pessoas. Para minimizar a ocorrência de colisões, vai-se aumentar o espaço para 5.000

Agora busca-se o primo mais próximo de 5000 para baixo que é 4.999. A nossa função de hash é

$$I = K mod 4999$$

A colisão será tratada de maneira bem simples, colocando-se o colidido em outra estrutura e indicando-se o ocorrido com a troca de sinal do valor no colisor. Acompanhe no código

 ©75-21, Pedro Kantek

```
import numpy as np
da=np.array(5000,int)
def fha(k):
  return k%4999
def inclusao(k):
  i=fha(k)
  if da[i]==0:
    da[i]=k
    return 0
  if da[i]==k:
    return i
  if da[i]<0:
    #...colisão a tratar...
    pass
def busca(k):
  i=fha(k)
  if da[i]==0:
    return -1
  if da[i]==k:
    return i
  if da[i]<0:
    #...colisão a tratar...
    pass
```

Completando a tabela de custos de acesso

método	custo	acessos para 1000 chaves
tabela hash	O(1)	1

Outra possibilidade de melhorar os custos de acesso está em usar a busca binária. Trata-se de processo similar ao que humanos usam ao buscar uma palavra em um dicionário. Claro, que neste caso está excluída a possibilidade de busca sequencial: a ninguém ocorrerá ler um dicionário sequencialmente até achar a palavra buscada.

A contrapartida a este método é que o vetor precisa ser guardado e mantido em ordem (aliás, tal como o dicionário). A idéia do método é buscar na metade do vetor qual chave existe. Se a chave buscada for menor que a chave do meio, a busca se concentrará na primeira metade do vetor (e a segunda metade é desprezada) ocorrendo o inverso se a chave buscada for maior do que a que está no meio. Com um único acesso, o tamanho do vetor caiu à metade. Prossegue-se neste método, com sucessivos cortes pela metade até localizar a chave buscada ou a garantia de que ela não está lá.

O custo desta busca com sucessivos cortes à metade é $\log_2 t$, onde t é o tamanho do vetor. Se $t = 1.000$ o logaritmo é 10, o que significa que são necessários 10 acessos para buscar uma chave. Se $R = 1.000.000$ o logarirmo é 20.

Acompanhe o código

```
da=[1,5,7,23,45,67,68,69,80,83,90,100]
def busca(k):
  fim=len(da)-1
  ini=0
  while 1==1:
    met=(ini+fim)//2
    if k <= da[met]:
      fim=met-1
    else:
      ini=met+1
    if da[met]==k or ini>fim:
      break
  if da[met]==k:
    return met
  else:
    return -1
print(busca(51))
```

Atualizando a tabela

método	custo	acessos para 1000 chaves
busca binária	$O(\log_2 t)$	10

Finalmente, a última busca que de alguma maneira resume todo este capítulo. Ela só perde para a tabela hash (mas em compensação não apresenta os problemas desta: permite processar o conjunto em ordem e não gera colisões) e a suprema qualidade é que não exige que os dados estejam em ordem. Na verdade, de uma maneira meio inesperada esta técnica tem melhor desempenho quanto mais DESORDENADOS os dados estiverem.

Trata-se da Árvore Binária de Busca. Antes de descrevê-la, vale a referência de que árvores são meio onipresentes na ciência da computação. O algoritmo de compressão de Huffman usa uma árvore, todos os índices de qualquer banco de dados usam árvores, corretores ortográficos usam árvores e por aí vai...

Uma árvore é uma estrutura hierárquica composta de nodos ao qual se associam diversos outros nodos (chamados de filhos do nodo original). Na árvore binária, o número máximo de filhos é 2. O único nodo que não possui pai, e que é o nodo iniciador da estrutura é chamado raiz da árvore. Curiosamente, na ciência da computação as árvores são desenhadas de ponta cabeça e daí que a raiz é o primeiro nodo, desenhado encima.

Árvores são estruturas ligadas. Neste tipo de estrutura misturam-se dados com endereços (ou ligações). Estas ligações é que são usadas para garantir eficiência na transversalidade dos dados. Em todos os métodos/estruturas vistos até agora, usavam-se estruturas sequenciais, nas quais os dados são contíguos o que permite sua busca sequencial.

A inexistência de um elemento ligado vai ser representada por um endereço (ligação) inválida ou neste caso valendo -1.

Nesta abordagem as ligações vão ser feitas usando-se cursores e não apontadores. Há uma pequena perda de desempenho no uso de cursores (representados pela dupla ligação que eles implicam) mas o ganho é substancial: todos os cursores ficam juntos e podem ser manuseados (e olhados) facilmente. O ganho sobretudo no *debugging* é enorme.

Na árvore binária de busca mais simples, cada nodo terá 3 valores: o valor da chave k, o endereço do filho esquerdo e o endereço do filho direito. Como usamos cursores a árvore vai ser mantida junta numa estrutura sequencial.

acompanhe a criação de uma árvore binária de busca (ABP) com as chaves 8, 17, 2, 21, 45 e 33 nesta ordem:
Inclusão do 8

8	-1	-1

Inclusão do 17

8	-1	1
17	-1	-1

Inclusão do 2

8	2	1
17	-1	-1
2	-1	-1

Inclusão do 21

8	2	1
17	-1	3
2	-1	-1
21	-1	-1

Inclusão do 45

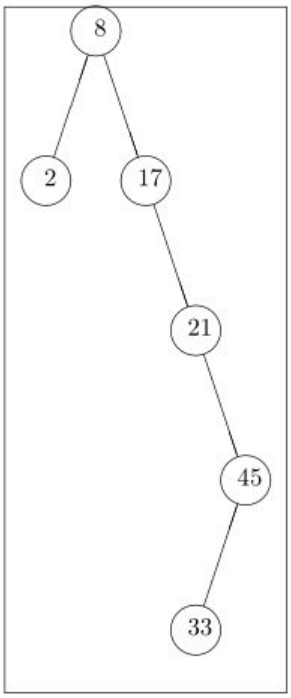
8	2	1
17	-1	3
2	-1	-1
21	-1	4
45	-1	-1

Inclusão do 33

8	2	1
17	-1	3
2	-1	-1
21	-1	4
45	5	-1
33	-1	-1

Eis como ficaria desenhada esta árvore

 ©75-21, Pedro Kantek



Acompanhe o algoritmo

```
import numpy as np
abp=np.array([[8,2,1],[17,-1,3],[2,-1,-1],
     [21,-1,4],[45,5,-1],[33,-1,-1]])
def buscaabp(k):
  onde=0 # nodo raiz
  while onde != -1:
    anterior=onde
    if k<abp[onde][0]:
      onde=abp[onde][1]
    else:
      if k>abp[onde][0]:
 onde=abp[onde][2]
      else:
 return anterior
  return -1
print(buscaabp(8))
```

Atualizando a tabela

método	custo	acessos para 1000 chaves
busca ABP	$O(\log_2 t)$	10

2.6 Primalidade

Define-se número primo como aquele inteiro que só é divisível pela unidade e por ele mesmo. A é divisível por B quando a divisão inteira de A por B deixa resto 0. Por exemplo, 5 é primo, pois seus únicos divisores são 1 e 5, enquanto 6 não é primo já que seus divisores são 1,2,3 e 6.

O contrário de um número primo é chamado número composto. Note-se que a unidade não é nem primo nem composto. Já o número 2 é o único primo que é par, já que todos os demais pares são divisíveis também por 2, logo compostos.

Uma característica interessante dos divisores de um número n é que eles sempre aparecem em pares, sendo que o produto dos elementos deste par reproduz o número n original. Esta regra só é quebrada quando n é um quadrado perfeito, mas para reestabelecer a regra basta tomar a raiz duas vezes. Por exemplo, os divisores de 100 são 1,2,4,5,10,20,25,50,100 e para que a regra acima continuasse válida eles deveriam ser 1,2,4,5,10,**10**,20,25,50,100.

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,...,p_r.

Façamos o conjunto $P = \{p_1, p_2, ..., p_r\}$. Imagine o número N, definido por $N = p_1 \times p_2 \times ... \times p_r$, 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 $M - N$ [1], 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

$$\zeta(x) = \frac{1}{1^x} + \frac{1}{2^x} + ...\frac{1}{n^x} + ...$$

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 rapido, 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

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + ...$$

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 (Satoy, 2004)* chegou a

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + ... = \frac{1}{6}\pi^2$$

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 $60 = 2\times2\times3\times5$ e daqui $\frac{1}{60} = \frac{1}{2}\times\frac{1}{2}\times\frac{1}{3}\times\frac{1}{5}\times = (\frac{1}{2})^2 \times \frac{1}{3} \times \frac{1}{5}$ e agora

$$\zeta(x) = (1 + \frac{1}{2^x} + \frac{1}{4^x} + ...) \times (1 + \frac{1}{3^x} + \frac{1}{9^x} + ...) \times ... \times (1 + \frac{1}{p^x} + \frac{1}{(p^2)^x} + ...) \times ...$$

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 Göttingen, quando Dirichlet foi trabalhar nessa cidade, 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.

[1] Exemplo: se 2 divide 8 e 2 divide 26, então 2 deve dividir a diferença entre 26 e 8 que é 18. Mais um exemplo, se 3 divide 15 e 3 divide 60 então 3 deve dividir a diferença $60 - 15 = 45$

 ©75-21, Pedro Kantek

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 raizes 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.

A próxima questão é programar se dado um número x ele é primo ou não. Vai-se usar esta tarefa para treinar a habilidade de refinamentos sucessivos na programação. Começando pela versão mais simples, derivada da definição de número primo.

```
def primo(x):
  d=2
  while d<x-1:
    if x%d==0:
      return False
    d=d+1
  return True
print(10,primo(10))
```

O próximo melhoramento é excluir o 2, depois os pares e depois crescer de 2 em 2. O desempenho cresce e o programa fica duas vezes mais rápido.

```
def primo(x):
  if x==2:
    return True
  if x%2==0:
    return False
  d=3
  while d<x-1:
    if x%d==0:
      return False
    d=d+2
  return True
print(11,primo(11))
print(12,primo(12))
```

A seguir, percebe-se que para detectar a primalidade, basta detectar um único fator, e da regra vista acima, sabe-se que eles vêm aos pares. Então a busca de fator pode parar ao se localizar a raiz quadrada do número. Fica

```
import math as mt
def primo(x):
  if x==2:
    return True
  if x%2==0:
    return False
  d=3
  while d<mt.sqrt(x):
    if x%d==0:
      return False
    d=d+2
  return True
print(11,primo(11))
```

Em resumo, um programa baseado no Crivo de Eratóstenes, pode ser assim obtido

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...)

Note que esta última regra acelera o algoritmo, já que o passo do while afora é $f = f + 6$. Eis o algoritmo

```
import math
def primo(n):
  if n==1:
     return False
  if n<4:
    return True
  if n%2==0:
    return False
  if n<9:
    return True
  if n%3==0:
    return False
  r=math.floor(math.sqrt(n))
  f=5
  while f<=r:
    if (n%f)==0 or (n%(f+2))==0:
      return False
    f=f+6
  return True
```

De posse de qualquer uma dessas funções, pode-se sair inventando novas funcionalidades. Seja agora uma função que recebe n e devolve o *n-ésimo* número primo. Obviamente, se a função $primo(x)$ já existe ela não vai ser reescrita apenas chamada. Eis como fica (o décimo número primo é 29).

```
def enesimop(n):
  can=2
  ctd=1
  while ctd<=n:
    if primo(can):
      ctd=ctd+1
    can=can+1
  return can-1
print("decimo",enesimop(10))
```

Seja agora o conceito de primo bom, que é Um número primo P_n é dito primo bom se e somente se $P_n^2 > P_{n-1} \times P_{n+1}$ para qualquer $1 \leq i \leq n-1$

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

$17^2 = 289 > P_6 \times P_8 \quad 13 \times 19 = 247$
$17^2 = 289 > P_5 \times P_9 \quad 11 \times 23 = 253$
$17^2 = 289 > P_4 \times P_{10} \quad 7 \times 29 = 203$
$17^2 = 289 > P_3 \times P_{11} \quad 5 \times 31 = 155$
$17^2 = 289 > P_2 \times P_{12} \quad 3 \times 37 = 111$
$17^2 = 289 > P_1 \times P_{13} \quad 2 \times 41 = 82$

Eis a programação de primo bom em C++

```
#include<iostream>
#include<cmath>
using namespace std;
   int primo(int x){
      //retorna 1 se x e primo e 0 senao
       int z,achei;
      achei = 1;
      z=2;
      if (x==2){
  return 1;
      }
```

 ©75-21, Pedro Kantek

```
      else {
      while ((z<(1+sqrt(x))) && (achei==1)) {
  if ((x%z)==0) {
      achei=0;
  }
  z++;
  }
  return achei;
      }
}

   int ordem(int x) {
      // retorna a ordem do numero primo x
      int me,md;
      me=1;
      md=2;
      while(md<=x) {
  if (primo(md)){
   me++;
  }
  md++;
      }
      return me-1;
   }

   int qual(int x) {
      // retorna o x-esimo numero primo
      int me,md;
      me=1;
      md=2;
      while(me<=x) {
  if (primo(md)){
   me++;
  }
  md++;
      }
return md-1;
   }

int main() {
   int n,g,incr,a;
   int quem;
   int resp;
   cin>>quem;

   if (primo(quem)==0){
      cout<<"nao e primo bom - nem e primo"<<endl;
   }
   cout<<"so e primo bom, se todos os r forem iguais a 1"<<endl;
   n=ordem(quem);
   cout<<"o numero dado "<<quem<<" e o primo de ordem "<<n<<endl;
   g=n;
   incr=1;
   while (n>1){
      a=(qual(g-incr))*(qual(g+incr));
      resp=(quem*quem)>a;
      cout<<(g-incr)<<" "<<(g+incr)<<" "<<(quem*quem)<<" "<<a<<" "<<resp<<endl;
      incr++;
      n--;
    }
}
```

2.7 Divisibilidade

Seja agora um conjunto de problemas sobre divisibilidade. Eis 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 7^o 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 python:

```
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=cnt+1
      i=i+1
print(t)
```

A implementação acima é muito demorada (estimei em umas 5h em uma CPU rápida)

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:

```
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 # correção quadrado perfeito
print(t)
```

Ainda assim, esta segunda estratégia continua muito demorada.

 ©75-21, Pedro Kantek

Terceira estratégia

Vamos nos socorrer de um pouco de matemática. Sabe-se que qualquer inteiro N pode ser expresso de maneira única como

$$N = p_1^{a1} \times p_2^{a2} \times p_3^{a3} \times ...$$

onde p_i é um número primo e a_n seu expoente. Por exemplo, $28 = 2^2 \times 7^1$. Daqui, o número de divisores de N, $D(N)$ de qualquer inteiro pode ser computado como

$$D(N) = (a_1 + 1) \times (a_2 + 1) \times ...$$

onde os a_n são os expoentes dos fatores primos que compõe o número N. No exemplo, os divisores de 28 são $D(28) = (2 + 1) \times (1 + 1) = 3 \times 2 = 6$. Uma tabela de primos é necessária para aplicar esta relação. Uma estratégia para criá-la pode ser (até 65500):

```
#def primo... já visto acima
def criatabprimos():
    pri=[2]
    i=1
    j=3
    while j<65500:
 if primo(j):
   pri.append(j)
   i=i+1
 j=j+2
    f=open("f:/p/.../primos.lis","w")
    f.write(str(a)[1:-1])
    f.close()
```

Depois, para ler essa tabela, usar:

```
def leiapri():
    f=open("f:/p/.../primos.lis","r")
    a=f.read()
    pri=list(map(int,a.split(", ")))
    print("feito")
```

Agora, o processamento

```
t=1
a=1
cnt=0
pri # lista de primos (acima...)
while cnt <= 500:
   cnt=1
   a=a+1
   t=t+a
   tt=t
   i=0
   while i < len(pri):
      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
print(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

$$D(t) = D(n/2) \times D(n+1) \; se \; n \; par$$

ou

$$D(t) = D(n) \times D((n+1)/2) \; se \; n+1 \; par$$

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:

```
n=3
Dn=2
cnt=0
f=open("f:/p/.../primos.lis","r")
a=f.read()
prix=list(map(int,a.split(", ")))
pri=prix[0:168] # lista primos até 1000
while cnt <= 500:
  n = n+1
  n1 = n
  if n1 % 2 == 0:
    n1 = n1//2
  Dn1 = 1
  i=0
  while i<len(pri):
    if pri[i]*pri[i] > n1:
      Dn1=2*Dn1
      break
    exponent=1
    while n1 % pri[i] == 0:
       exponent=exponent+1
       n1 = n1//pri[i]
    if exponent > 1:
       Dn1=Dn1*exponent
    if n1 == 1:
       break
    i=i+1
  cnt = Dn*Dn1
  Dn = Dn1
print (int((n*(n-1)/2)))
```

Esta implementação demorou menos de $\frac{1}{10}$ de segundo na mesma CPU lá do começo. Se a minha estimativa inicial da primeira estratégia estiver certa (5h de CPU) a quarta estratégia é $5 \times 60 \times 60 \times 10 = 180.000$ vezes mais eficiente que a primeira.

Se este ganho de eficiência pudesse ser aplicado no preço de um Big-Mac, ele passaria de $20R\$$ para 0.01 centavos de real. Com uma moedinha de 10 centavos você poderia comprar 9 Big-Macs. Veja-se como vale a pena estudar algoritmos.

2.7.1 Números perfeitos, imperfeitos e mais que perfeitos

Esta é uma classificação que advém da soma dos divisores próprios de um determinado número (estes são todos os divisores à exceção do próprio número. Por exemplo os divisores próprios de 6 são 1, 2 e 3). Dada esta definição os números podem ser divididos em

perfeitos: São os números iguais à soma dos seus divisores próprios. Por exemplo 6 ou 28.

imperfeitos: São os números maiores do que a soma de seus divisores próprios. Por exemplo, 4 ou então qualquer número primo.

 ©75-21, Pedro Kantek

mais do que perfeitos: São os números menores do que a soma de seus divisores próprios. Por exemplo 60. Seus divisores são 1,2,3,4,5,6,10,12,15,20 e 30 que somados perfazem 108, e $108 > 60$.

Seja agora uma função que recebe n e retorna 1,2 ou 3 em cada um dos 3 casos acima. Note que a codificação se inicia pela definição da função `sdpro(x)` que retorna a soma dos divisores próprios de um determinado número.

```
def sdpro(x):
  som=1
  for i in range(2,1+x//2):
    if x%i==0:
      som=som+i
  return som

def tiponum(x):
  s=sdpro(x)
  if x==s:
    return 1
  else:
    if x>s:
      return 2
  return 3
print(tiponum(28))
print(tiponum(4))
print(tiponum(60))
```

Ainda nessa seara, vamos estudar os números amigos: a é amigo de b se a é igual a soma dos divisores próprios de b e b é amigo de a se b é igual à soma dos divisores próprios de a. E, eles são amigos entre si se um é amigo do outro. Vamos estudar o código que recebe dois números e responde `True` se eles são amigos e `False` senão.

```
def amigos(a,b):
  sa=sdpro(a)
  sb=sdpro(b)
  return a==sb and b==sa
print(amigos(220,284))
print(amigos(110,162))
```

2.8 Geometria analítica 140a, 902

Este é um capítulo que ganha importância a cada dia. Começando lá na França no século XVII, na genial descoberta de René Descartes, que embora seja lembrado como filósofo (*Penso, logo existo*, a frase conhecida por sua forma em latim *Cogito, ergo sum*, é uma frase do filósofo francês René Descartes. A frase original foi escrita em francês *Je pense, donc je suis* e está no livro Discurso do Método, de 1637) foi um fantástico matemático: dele obtemos a Geometria Analítica.

Trata-se do tratamento algébrico para questões geométricas (ou aqui a maravilha da coisa: o tratamento geométrico para questões algébricas). O pulo do gato está em pensar um ponto e dar-lhe coordenadas x e y em um plano (muito apropriadamente conhecido como plano cartesiano). As equações – álgebra – do ponto x, y – geometria – dizem e fazem tudo. Contam os biógrafos que o tenente René Descartes estava deitado descansando em um acampamento militar, que não devia ser um lugar muito limpo, quando observou uma mosca andando sobre o teto do lugar. Ele pensou na mosca como um ponto e as laterais do teto como os eixos coordenados x e y e matutou em como especificar a mosca como funções de x e de y: pronto. Acabava de nascer a Geometria Análitica ou – muito merecidamente – Geometria Cartesiana.

E, porque este assunto ganha importância hoje ? Por causa da georreferência e geolocalização que são a moda da hora. Vejam-se o GPS, o Uber, o aluguel de patinetes e os aplicativos de paquera, só para ficar em alguns exemplos, que

funcionam localizando uma pessos (a mosca do René) sobre o planeta Terra (teto do René) e daí medindo distâncias e roteiros.

A abundância de sistemas de apoio à geolocalização (o GPS americano, o Galileu europeu, o Glonass russo ou o chinês Beidou) aliado as inúmeras vantagens da georreferência faz com que esta funcionalidade esteja mais ou menos presente em todo sistema de informações do século XXI. Afinal, uma característica fixa e importante de qualquer coisa que ocupe lugar no espaço é sua localização.

No caso do planeta Terra, os dois eixos ortogonais são o equador e o meridiano de Greenwich (um observatória da Marinha Britânica perto de Londres). Se olharmos o planeta com o equador na horizontal e o polo norte acima, tem-se que pontos ao norte do Equador terão latitude positiva. Pontos ao sul do Equador, têm latitude negativa. Curitiba, por exemplo tem latitude -25^o, ou também 25^o S.

O outro eixo, que seria o eixo das abcissas divide o planeta entre oriente (à direita, com valores negativos) e ocidente, à esquerda, com valores positivos. Curitiba têm longitude +49^o ou mais apropriadamente 49^o W.

A medida original era graus, minutos e segundos, mas a necessidade de maior precisão (como quebrar os segundos ? em décimos ?) fez com que todo o esquema fosse discutido e hoje é muito frequente a medida de graus e estes divididos em partes decimais ao infinito. Como exemplo, as duas maneiras de representar as coordenadas da cidade de Curitiba: Latitude: -25.4284, Longitude: -49.2733 ou em graus minutos e segundos: 25^o 25 42 Sul, 49^o 16 24 Oeste.

A seguir, algumas práticas:

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).

```
import numpy as np
import math
def ex1(p1,p2):
  return abs(p1[0]-p2[0])*abs(p1[1]-p2[1])

#print(ex1([2,5],[5,3]))
```

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.

```
def ex2(a,b,c):
  x=np.zeros((3,3))
  x[0,0]=a[0]
  x[0,1]=a[1]
  x[0,2]=1
  x[1,0]=b[0]
  x[1,1]=b[1]
  x[1,2]=1
  x[2,0]=c[0]
  x[2,1]=c[1]
  x[2,2]=1
  w1=x[0,0]*x[1,1]*x[2,2]+x[0,1]*x[1,2]*x[2,0]+x[1,0]*x[2,1]*x[0,2]
  w2=x[2,0]*x[1,1]*x[0,2]+x[0,0]*x[1,2]*x[2,1]+x[1,0]*x[0,1]*x[2,2]
  if 1==abs(w1-w2)%2:
    return -1
  return (abs(w1-w2))/2
#print(ex2([2,5],[2,-1],[8,1])  )
```

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 circulo é 125 u.a.

```
def ex3(a,b):
  r=(((a[0]-b[0])**2)+((a[1]-b[1])**2))**0.5
  return math.pi*r**2

#print(ex3([0,-3],[-6,-5]))
```

4. **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.

```
def ex4(a,b,c):
  tot=ex2(a,b,c)
  a1=ex2(a,[0,0],c)
  a2=ex2(a,b,[0,0])
  a3=ex2([0,0],b,c)
  if a1+a2+a3 != tot:
    return False
  else:
    return True
#print(ex4([-6,6],[-1,1],[-5,2]))
```

5. **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.

```
def ex5(x):
  y=np.array(x)
  r=y.reshape((10,2))
  print(r)
  q1=q2=q3=q4=0
  for i in range(10):
    if r[i,0]>=0 and r[i,1]>=0:
      q1=q1+1
    if r[i,0]<=0 and r[i,1]>=0:
      q2=q2+1
    if r[i,0]<=0 and r[i,1]<=0:
      q3=q3+1
    if r[i,0]>=0 and r[i,1]<=0:
      q4=q4+1
  return q1+q2+q3+q4
#print(ex5([-3,-7,-4,-4,0,-4,-6,8,6,-7,-9,-3,-5,9,0,0,7,2,-7,7]))
```

6. **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).

```
def ex6(x):
  y=np.array(x)
  r=y.reshape((5,2))
  dma=-9999999
  for i in range(5):
    d=((r[i,0]**2)+(r[i,1]**2))**0.5
    if d>dma:
      dma=d
      ima=i
  return ima+1
#print(ex6([5,-4,1,2,9,-7,-5,-9,-3,8]))
```

7. **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).

```
def ex7(x):
  y=np.array(x)
  r=y.reshape((6,2))
  dma=-9999999
  for i in range(1,6):
    d=(((r[i,0]-r[0,0])**2)+((r[i,1]-r[0,1])**2))**0.5
    if d>dma:
      dma=d
      ima=i
  return ima+1
#print(ex7([-1,-9,4,1,-6,-5,-4,8,0,-7,7,5]))
```

2.9 Dígitos Verificadores

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 atribuia 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 MPAR
- 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.

 ©75-21, Pedro Kantek

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 a direita. A seqüência 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 divivir 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.

CNPJ (antigo CGC) a) São 3 DVs.
b) O DV1 ocupa a 8.posição do código e é um Módulo 10 dos 7 dígitos iniciais.
c) O DV2 ocupa a 13.posição e é um Módulo 11 de todos os anteriores com o vetor de *pesos* = 543298765432. d) O DV3 ocupa a 14.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.

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.

Boleto Bancário

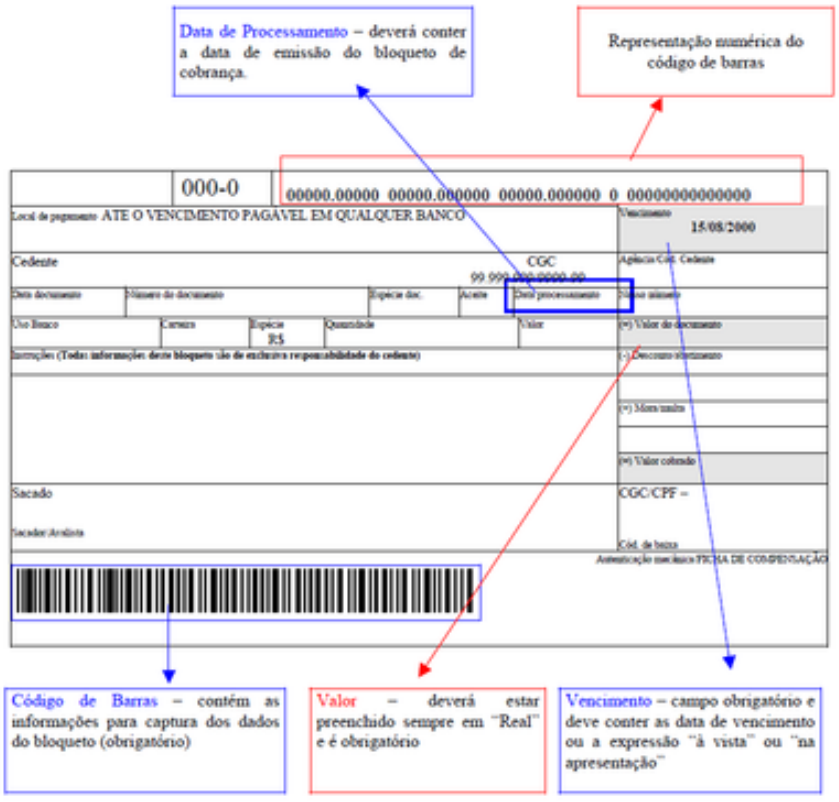
O sistema bancário brasileiro é muito eficiente. Resultado direto dos muitos anos em que convivemos com uma inflação renitente e da cultura brasileira de transfere aos bancos inúmeros procedimentos de cobrança aliado ao tamanho imenso de nosso país o resultado é um conjunto de bancos e de práticas eficientes e modernos.

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.

Existe uma norma do Banco Central do Brasil (Carta Circular 2926 de 25/07/2000) e uma orientação da FEBRABAN (Federação Brasileira de Bancos) que padroniza o bloqueto e elas vão ser resumidas aqui. Existe uma lista de informações que obrigatoriamente devem estar impressas no bloqueto e são:

- 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.

Eis uma imagem do bloqueto com algumas orientações



 ©75-21, Pedro Kantek

A parte que é capturada no pagamento é formada pelas seguintes informações:

tamanho	conteúdo
3	identificação do banco (BBB)
1	moeda (9=real) (M)
4	fator de vencimento (1000=3/7/2000) (FFFF)
10	valor (incluídas 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:

tamanho	conteúdo
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)

O fator de vencimento é o número de dias transcorridos desde 07/10/1997 (na verdade desde 03/07/2000 - 1000d). Existe uma limitação aqui, já que esta numeração só aguenta até 21/02/2025. A conta dos 1000 dias é necessária já que quando a primeira posição do fator de vencimento é igual a zero, isso significa que o vencimento é "contra apresentação". Eis algumas datas e seus fatores de vencimento:

fator	vencimento
1000	03/07/2000
1002	05/07/2000
1667	01/05/2002
4789	17/11/2010
9999	21/02/2025

O cálculo do DV: Dispõe-se os dados do bloqueto como segue

```
BBBMFFFFVVVVVVVVVVLLLLLLLLLLLLLLLLLLLLLLLL
```

e multiplica-se cada dígito por

```
43298765432987654329876543298765432987654 32
```

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='34194897000004450017500000564138355700070 00'
 é 8
```

```
DV='059725578998338320406689183396264233650824 7'
 é 3
DV='304042776831780789165422074364974541796755 3'
 é 9
DV1='341917500' é 9
DV2='0005641383' é 4 e
DV3='5570007000' é 0
DV2='6361840992' é 2
DV1='405100184' é 2
```

Uma solução bem simples

```
def cpf(x):
  z=x[8]*2+x[7]*3+x[6]*4+x[5]*5+x[4]*6+x[3]*7+x[2]*8+x[1]*9+x[0]*10
  y=11-(z%11)
  if y == 10 or y == 11:
    d1 = 0
  else:
    d1 = y
  z=d1*2+x[8]*3+x[7]*4+x[6]*5+x[5]*6+x[4]*7+x[3]*8+x[2]*9+x[1]*10+x[0]*11
  y=11-(z%11)
  if y == 10 or y == 11:
    d2 = 0
  else:
    d2 = y
  return([d1,d2])
  # para chamar faça: aa=cpf([1,1,1,2,2,2,3,3,3])
```

Uma solução um pouco mais compacta e sofisticada (atenção: é um script apenas)

```
c=[2,5,3,0,1,3,0,6,9]
p=[10,9,8,7,6,5,4,3,2]
s=0
for i in range(9):
  s=s+(c[i]*p[i])
d=11-(s%11)
if d == 10 or d==11:
  d=0
c=c+[d] #pode ser c.append(d)
p=[11]+p
s=0
for i in range(10):
  s=s+(c[i]*p[i])
d2=11-(s%11)
if d2 == 10 or d2==11:
  d2=0
print([d,d2])
```

2.10 Calendários

Os calendários antigos tinham que conciliar 2 ciclos: o da lua e do sol. O da lua deu origem ao 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.

Notem-se as frações a seguir (o número certo é 365,242199):

$$365,2425 = 365 + \frac{1}{4} - \frac{1}{100} + \frac{1}{400}$$

$$365,242197 = 365 + \frac{1}{4} - \frac{1}{100} + \frac{1}{400} - \frac{1}{3300}$$

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)

 ©75-21, Pedro Kantek

Janeiro em homenagem a Janius, deus romano que cuidava das portas e das janelas

Fevereiro em homeganem às Februas (festival de purificação)

Março em homenagem a Marte deus da guerra

Abril não há consenso, mas historiadores acham que deriva de apripe (latim antigo = abrir) as flores

Maio maiores mês dos velhos

Junho juniores mês dos jovens

Quintilio

Sextilio

Setembro septilio

Outubro outilio

Novembro novitilio

Dezembro decitilio

Julio Cezar resolveu mudar o quintílio para Julho, e depois dele, Cezar Augusto, mudou o sextílio para Agosto.
os bispos reunidos no Primeiro Concílio de Nicéia, realizado em 325 determinaram que a Páscoa ocorreria sempre:

- num domingo
- depois da primeira lua cheia que
- se seguisse ao equinócio de primavera. (o sol sobre o equador)

O ano juliano tinha 11 min e 14 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.

calendário republicano francês

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, fructidor, brumaire...). Este calendário durou pouco, foi abolido por Napoleão em 1805.

PROPOSTAS NOVO CALENDÁRIO

- Da ONU, aprovada em 1954. O ano é de 52 semanas e 364 dias. O dia 365 é no final do ano, (DIA DE FIM DE ANO) não tem número nem nome na semana. Nos anos bissextos este dia cai após a 26ª semana. O primeiro mês do quadrimestre tem 31 dias e os demais tem 30.
- Calendário Internacional Fixo. 13 meses de 28 dias, mais 1 dia que não pertence a nada no fim do ano. O dia do bissexto vem após 28 junho. A vantagem é que as segundas sempre são dias 1, 8, 15 e 22. O novo mês é o mês SOL entre junho e julho.
- Calendário do congresso americano => orientado a business Finalmente, há uma nova proposta de estabelecer um tempo mundial, (desvinculado do sol), orientado a negócios e com divisões decimais. Aos poucos é este sistema que vai sendo implementado subrepticiamente. Por exemplo, no tráfego aéreo mundial.

Caso prático: Calendários

Em 1347 a peste negra devastou a Europa. Esta doença é na verdade uma pneumonia que causa bubões (inchaços nas axilas e virilhas), sendo também chamada de peste bubônica. Os bubões eram caldos de cultura da bactéria causadora, que também era transmitida por pulgas dos ratos. O nome negra, vem do fato de que (supostamente) a carne das vítimas enegrecia pouco antes da morte. A história começa quando uma tribo Mogol de nome Kipchak resolve atacar um posto comercial genovês no Mar Negro. Usaram para isso uma das primeiras armas biológicas que a história registra: cadáveres humanos contaminados com a doença eram atirados através de catapultas para dentro da cidade. Esta embora armada e provisionada para resistir ao cerco, ao ver-se impotente para enfrentar esta arma resolveu fugir de navio de volta a Gênova, abandonando o posto comercial. Junto com eles, foi a doença. Durante os 4 anos seguintes a peste foi para a Sicília, África, Itália, Espanha, França, Inglaterra e depois toda a Europa. Vinte e cinco milhões de pessoas, um quarto

da população européia, morreram. Dois séculos passariam antes que a população retornasse ao número de 100 milhões. A devastação da peste negra encerrou um ciclo na história da humanidade. Levantes sociais e políticos espoucaram, a mão de obra escasseou, o campo foi abandonado, os alimentos rarearam. Até aqui, o universo era governado por regras estabelecidas por duas autoridades: Aristóteles (384-322 aC) e o egípcio Cláudio Ptolomeu (100-170 dC). No milênio anterior, a Igreja havia mesclado essas regras à sua visão de mundo, resultando um bloco homogêneo: Deus havia criado a Terra no centro do Universo. Estrelas e planetas giravam em órbitas circulares em volta da terra. Oito esferas concêntricas feitas de material imutável e eterno continham a Lua, o Sol, Marte, Mercúrio, Júpiter, Vênus e Saturno. A última esfera continha as estrelas. Apenas na terra a matéria se decompunha e morria. Nas esferas tudo era eterno. Veja-se o reflexo disso nos nomes dos dias de semana, nos principais idiomas ocidentais:

Corpo Celeste	Inglês	Francês	Italiano	Espanhol
Sol	Sunday	dimanche	domenica	domingo
Lua	Monday	lundi	lunedi	lunes
Marte	Tuesday	mardi	martedi	martes
Mercúrio	Wednesday	mercredi	mercoledi	miércoles
Júpiter	Thursday	jeudi	giovedi	jueves
Vênus	Friday	vendredi	venerdi	viernes
Saturno	Saturday	samedi	sabato	sábado

A reação à peste foi um novo cenário que buscou enterrar e esquecer aquele o mais rápido possível: a renascença. A Itália, pela localização central iniciou o movimento do comércio entre oriente e ocidente. Nasceram aqui os primeiros sistemas administrativos, o conhecimento financeiro e os bancos. A ciência (a matemática) começou a ser usada: na arquitetura, no comércio, na cartografia. Sabemos hoje que a Terra demora 365 dias, 5 horas, 48 minutos e 46 segundos para uma volta completa ao redor do sol. Egípcios haviam estimado este valor em 365,25 dias (ou seja 365 dias e 6 horas. Ficou uma diferença de 11 minutos e 14 segundos). Com a adoção do calendário egípcio por Júlio Cézar no século I dC, esta disparidade foi se acumulando ano após ano, afastando as datas do calendário das estações. Em meados do século XV ja havia dez dias de atraso. Em 1475 o papa Sisto IV pedira um estudo para determinar a causa do erro. Não houve quem conseguisse compatibilizar Aristóteles e Ptolomeu com o calendário. Podiam garantir estabilidade política, mas estavam cada vez mais incapazes de calcular a data da Páscoa corretamente.

Copérnico (1473-1543) começou a desenvolver em 1506 um sistema astronômico baseado em suas próprias observações e cálculos. Na tentativa de tirar a Terra do centro do universo e colocar aí o Sol, a dificuldade era explicar como as coisas não "caíam"em direção ao sol. Achava-se na época, que a matéria era naturalmente atraída para o centro do universo (embaixo da terra). Quando recrutado pelo secretário do papa, em 1514 para resolver o problema do calendário, Copérnico viu-se num dilema: ou reafirmava a teoria que a vaidade, o medo e a Bíblia haviam montado (estamos no centro do universo) e não resolvia o problema, ou chutava o pau da barraca para propor um novo calendário.

Ele, que bobo não era, recusou o convite, embora continuando a estudar o problema em segredo. Convencido pela correção de seus cálculos e encorajado por amigos, esboçou um rascunho de suas idéias em 1530. Este escrito provocou reações mistas. Finalmente, em 1543, autorizou a publicação de *De Revolutionibus Orbium Coelestium*, comumentemente conhecido como As Revoluções. Copérnico recebeu o primeiro exemplar em 24 de maio de 1543 e morreu poucas horas depois.

O próximo personagem desta história é Galileu Galilei. Pulamos Giordano Bruno, queimado na fogueira no Campo di Fiori em 17 de fevereiro de 1600. Galileu teria deixado cair duas bolas de pesos diferentes da Torre de Pisa para provar que ambas chegariam juntas, contrariando Aristóteles que afirmara chegar a mais pesada antes. Há dúvidas se isso de fato ocorreu. Mais modernamente supõe-se que tenha sido uma experiência intelectual apenas. O argumento intelectual é notável[2]Este é o melhor jeito de fazer física, à la Einstein...

Galileu freqüentou diversas universidades na Itália, sempre conflitando com os demais professores. Escreveu inúmeros livros, "inventou" o telescópio [3] Ao usar os telescópios, viu quatro satélites orbitando Júpiter e os anéis de Saturno e relatou isso no livro *Sidereus Nuncius*(O Mensageiro Celeste). Escreveu de leve, ainda sem adotar completamente o modelo copernicano. A Igreja já começou a enviar mensagens de que não concordava com as idéias desse livro. Em 21 de dezembro de 1614, o padre Tomás Caccini, em Florença criticou Galileu aformando que se Deus parou o sol a pedido de Josué para que os israelitas derrotassem os amoritas, como o sol poderia ser o centro do universo ? O padre foi mais longe: acusou Galileu, a matemática e todos os matemáticos de hereges políticos e religiosos. A defesa de Galileu é perfeita:

> Não me sinto na obrigação de acreditar que o mesmo Deus que nos dotou de sentidos, razão e intelecto, tencionava descartar o uso destes e por algum outro meio nos dar o conhecimento que com eles podemos obter [...] A intenção do Espírito Santo é ensinar-nos como se vai para o céu e não como o céu funciona.

Em 5 de março de 1616 o cardeal Belarmino da Santa Inquisição decretou que o sistema copernicano era "falso e errôneo" afirmando que Deus fixou a Terra em seus alicerces para jamais ser movida. Galileu nunca se conformou com este decreto, buscando incessantemente a sua revogação. Em 1624, obteve do papa uma autorização para escrever seu

[2]Suponha jogar as 2 bolas e suponha que Aristóteles estava certo: a mais pesada chega antes. Agora suponha as mesmas duas bolas porém ligadas por fio. Por um lado pode-se argumentar que a mais leve "segura" a mais pesada e esta demora mais a chegar. Por outro lado, pode-se supor que o fio transforma as duas massas em uma só, que passa a ser mais pesada que a bola anterior. O resultado é que agora as bolas chegam antes do que chegavam. Para que ambos os raciocínios estejam certos e estão, a única possibilidade é elas chegarem juntas.

[3]Ele é assim considerado pois construiu os maiores telescópios em uso na Europa, ultrapassando seus modelos anteriores por diversas vezes.

 ©75-21, Pedro Kantek

livro mais famoso *Diálogo sobre os dois máximos sistemas do mundo*, que foi publicado em 1632 e recebido com louvores por acadêmicos de toda a Europa. Pouco depois, foi acusado pela inquisição de herege. Não pode ver as acusações ou as provas. Ficou no dilema: ou se retratava ou morria como Giordano Bruno. Em decisão que alguns criticaram como prejudicial a ciência, ele resolveu pela vida. Em 22 de junho de 1633, fez uma longa retratação. Diz a lenda que depois de terminada a leitura, ao se erguer da posição de joelhos onde estava,Galileu teria dito *"E pur, si muove"* . Só em 1757 a Igreja retirou a proibição sobre a obra. Em 1992, o papa João Paulo II reconheceu formalmente o erro da Igreja.

Depois vem Isaac Newton, um dos 5 maiores cientistas que a Humanidade produziu (decomposição da luz, cálculo diferencial e integral, gravitação entre outros). Ao escrever sobre a lei universal da gravitação, ele acabou de consolidar o modelo copernicano, que já havia sido engordado com as leis de Kepler sobre o deslocamento dos planetas.

Calendários Juliano e Ptolomeico

A encrenca do calendário está em que ele lida com 3 ciclos distintos: solar: Alterna as estações e depende do sol. Mede os anos; lunar: Alterna as luas (cheia, nova ...) e depende da lua. Mede os meses; semanal: Mede os dias da semana e tem origem religiosa. Indica o sabath.

A dificuldade e que estes 3 ciclos não são múltiplos entre si.

Em 45 aC, Júlio Cezar criou o calendário juliano, baseado no ptolomeico. Neste o ano tinha 365d e um quarto de dia. Os meses foram batizados de Janius (portas e janelas), Februas (festa da purificação), Mars (guerra), Apripe (abertura das flores), Maiores, Juniores, Quintílio, Sextílio, Septílio, Octílio, Novitílio e Decitílio. Logo depois, o próprio JC mudou o nome do Quintílio para Julius. Seu sucessor, Cezar Augusto não deixou por menos, mudando o próximo para Augustus.

Em 1582, o papa Gregório mudou o calendário, que com a reforma passou a ser conhecido como calendário gregoriano. Sumiram 10 dias em março desse ano. Adotou-se a regra dos anos bissextos. A adoção deste calendário demorou. Na Inglaterra ele só foi aceito em 1752. Em 1793, a Revolução Francesa mudou novamente o calendário. Nele há 12 meses de 30 dias e 5 feriados nacionais. Os meses: germinal, floral, thermidor, fructidor, brumaire. Napoleão o aboliu em 1805.

Regra do bissexto

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.

Observação: o tema calendários está tratado no livro de algoritmos.

Calendário Republicano Francês

Os anos sempre começavam no equinócio do outono. Tinha 12 meses de 30 dias cada. Cada mês se divide em 3 décadas de 10 dias. Desaparecem as semanas. Os nomes dos meses adotam fenômenos naturais ou agrícolas: Outono (nome terminado em aire) Vendémiaire, (vendimiário), a partir do 22, 23 ou 24 de setembro Brumaire, (brumário), a partir do 22, 23 ou 24 de outubro Frimaire, do francês frima (geada), a partir do 21, 22 ou 23 de novembro

Inverno (terminado em -ôse): Nivôse (nevoso), a partir do 21, 22 ou 23 de dezembro Pluviôse (chuvoso), a partir do 20, 21 ou 22 de janeiro Ventôse (ventoso), a partir do 19, 20 ou 21 de fevereiro

Primavera (terminado -al): Germinal (semeal, de semear), a partir de 20 ou 21 de março Floréal (floral), a partir de 20 ou 21 de abril Prairial (praderal), a partir de 20 ou 21 de maio

Verão (terminado em -idor): Messidor (messe, colheita), a partir de 19 ou 20 de junho Thermidor (calor), a partir de 19 ou 20 de julho Fructidor (frutador), a partir de 18 ou 19 de agosto

As décadas são : primidi, duodi, tridi, quartidi, quintidi, sextidi, septidi, octidi, nonidi, décadi.

Os 5 dias sobrantes (6 em anos bissextos) aparecem no fim do ano. No começo eram conhecidos como *Sans-culottides*, mas depois viraram festas:

Fête de la Vertu "Festa da Virtude"dia 17 ou 18 de setembro Fête du Génie "Festa do Talento"no 18 ou 19 de setembro Fête du Travail "Festa do Trabalho"dia 19 ou 20 de setembro Fête de l'Opinion "Festa da Opinião"dia 20 ou 21 de setembro Fête des Récompenses "Festa das Recompensas"no 21 ou 22 de septembro Fête de la Révolution "Festa da Revolução"dia 22 ou 23 de setembro (em anos bissextos)

Os dias do ano Em vez de um santo (como no calendário católico), cada dia ganha um nome (planta, animal ou ferramenta). Há uma lista de todos os dias, dos quais retirei algumas datas:
26/09 - dia do cavalo (cheval)
06/10 - dia do asno (âne)
19/12 - dia da oliva
25/12 - dia do cachorro (chien)
14/01 - dia do gato (chat)
12/04 - dia da castanheira (marronnier)
04/07 - dia do tabaco (tabac)
...
A seguir, um script bem simples para achar o dia da semana

```
import math
dia=16
mes=4
ano=1722
A=math.floor((12-mes)/10)
B=ano-A
C=mes+(12*A)
D=math.floor(B/100)
E=math.floor(D/4)
F=E+2-D
G=math.floor(365.25*B)
H=math.floor(30.6001*(C+1))
I=F+G+H+dia+5
R=I%7
print(R)
```

Para achar a Páscoa de um ano qualquer

```
import math
ano=2016
A=ano%19
B=math.floor(ano/100)
C=ano%100
D=math.floor(B/4)
E=B%4
F=math.floor((B+8)/25)
G=math.floor((1+B-F)/3)
H=((19*A)+B+15-(D+G))%30
I=math.floor(C/4)
K=C%4
L=(32+(2*E)+(2*I)-(H+K))%7
M=math.floor((A+(11*H)+(22*L))/451)
P=math.floor((H+L+114-(7*M))/31)
Q=(H+L+114-(7*M))%31
print(Q+1,P)
```

Finalmente, um código completo:

```
import numpy as np
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
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
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
```

 ©75-21, Pedro Kantek

```
    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+l+114-(7*m))//31)
    q=(h+l+114-(7*m)) % 31
    print ("pascoa ", (q+1),p)
    if (bissexto(ano)==1):
 dm[1]=29;
    soma=0
    i=1
    while i<12:
 dp[i]=dp[i-1]+dm[i-1]
 i=i+1
    ordi=dp[p-1]+q+1
    carna=ordi-47
    ss=ordi-2
    cc=ordi+60
    c=ordi+60;
    i=0
    while dp[i]<carna:
       i=i+1
    print("terca de carnaval ",(carna-dp[i-1]),i )
    i=0
    while dp[i]<ss:
       i=i+1
    print("sexta santa ", (ss-dp[i-1]), i)
    i=0
    while dp[i]<cc:
 i=i+1
    print("Corpus Christi ",(cc-dp[i-1]),i )

pascoa(2019)
```

2.11 Diagonais, treliças ... (035c)

Exponenciação eficiente

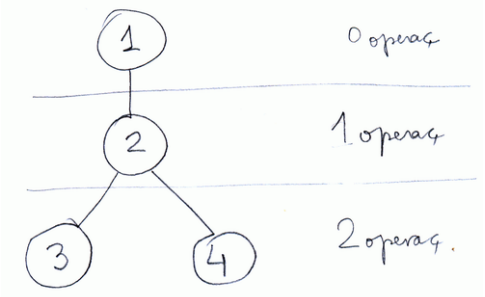
Problema retirado do maravilhoso **eulerproject.net** onde é o problema 122. Para calcular x^n, deve-se começar com $n = 1$. Aqui não há o que fazer e a resposta está calculada, pelo que o número de operações é zero.

Depois, $n = 2$. Também não há o que pensar. Tem-se que fazer $x \times x$. O número de operações é 1.

Quando $n = 3$, deve-se calcular primeiro n^2 (1 operação) e depois multiplicar este por n. Total 2 operações.

Quando $n = 4$, deve-se calcular primeiro n^2 (1 operação) e depois multiplicar este por ele mesmo. Total 2 operações.

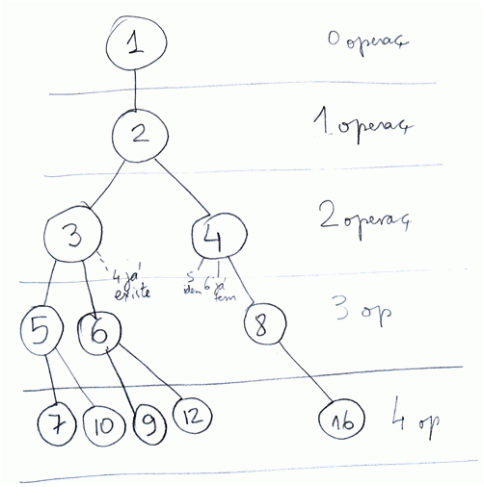
Neste ponto, consegue-se desenhar a seguinte estrutura de solução, que por acaso é uma árvore.



A regra para continuar a construir a árvore é: **Escolha a folha mais à esquerda. Combine-a com todos os nodos dela mesma, até a raiz. Se o resultado for inédito, desenhe-o como filho deste nodo. Se já tiver**

aparecido antes, despreze-o

Seguindo esta regra, desenha-se o próximo nivel.



Apenas para você ver como fica no programa eis um pedaço da árvore

```
Num    N  pai ops ok   --------------parciais-----------------------
  1-   1   -1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2-   2    1   1  88   1   0   0   0   0   0   0   0   0   0   0   0
  3-   3    2   2  88   2   1   0   0   0   0   0   0   0   0   0   2
  4-   4    2   2  88   2   1   0   0   0   0   0   0   0   0   0   1
  5-   5    3   3  88   3   2   1   0   0   0   0   0   0   0   0   2
  6-   6    3   3  88   3   2   1   0   0   0   0   0   0   0   0   1
  7-   7    5   4  88   5   3   2   1   0   0   0   0   0   0   0   3
  8-   8    4   3  88   4   2   1   0   0   0   0   0   0   0   0   1
  9-   9    6   4  88   6   3   2   1   0   0   0   0   0   0   0   2
 10-  10    5   4  88   5   3   2   1   0   0   0   0   0   0   0   1
 11-  11    9   5  88  10   5   3   2   1   0   0   0   0   0   0   5
 12-  12    6   4  88   6   3   2   1   0   0   0   0   0   0   0   1
 13-  13    9   5  88  10   5   3   2   1   0   0   0   0   0   0   3
 14-  14    8   5  88   7   5   3   2   1   0   0   0   0   0   0   1
 15-  15    9   5  88  10   5   3   2   1   0   0   0   0   0   0   2
 16-  16    7   4  88   8   4   2   1   0   0   0   0   0   0   0   1
 17-  17   12   5  88  16   8   4   2   1   0   0   0   0   0   0   5
 18-  18   10   5  88   9   6   3   2   1   0   0   0   0   0   0   1
 19-  19   13   6  88  14   7   5   3   2   1   0   0   0   0   0   3
 20-  20    9   5  88  10   5   3   2   1   0   0   0   0   0   0   1
 21-  21   13   6  88  14   7   5   3   2   1   0   0   0   0   0   2
 22-  22   14   6  88  11  10   5   3   2   1   0   0   0   0   0   1
 23-  23   15   6  88  13  10   5   3   2   1   0   0   0   0   0   2
...
```

Observação: esta árvore não está desenhada na ordem em que ela foi criada. Se você acompanhar o desenho, verá – por exemplo – que o nodo 16 é criado muito antes do 15 ou do 14. Mas, a árvore acima foi **ordenada** antes de ser impressa...

Diagonais

É o problema 28 do `projecteuler.net`. Diz:

 ©75-21, Pedro Kantek

Project Euler.net

About | Archives | Recent | News | Register | Sign In

Number spiral diagonals

Problem 28

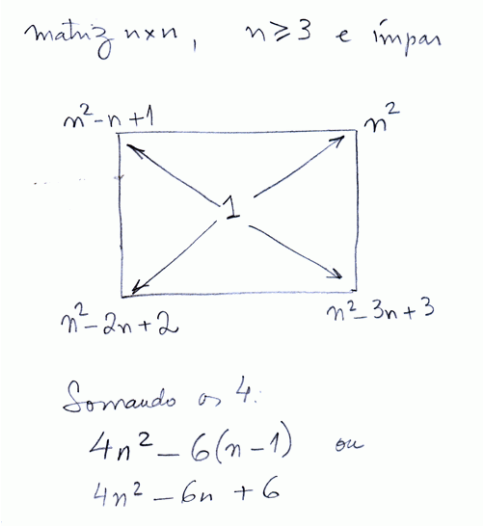
Starting with the number 1 and moving to the right in a clockwise direction a 5 by 5 spiral is formed as follows:

```
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
```

It can be verified that the sum of the numbers on the diagonals is 101.

What is the sum of the numbers on the diagonals in a 1001 by 1001 spiral formed in the same way?

Olhando a matriz que está escrita na folha, e estudando as 4 pontas em diagonal a partir do centro, tem-se



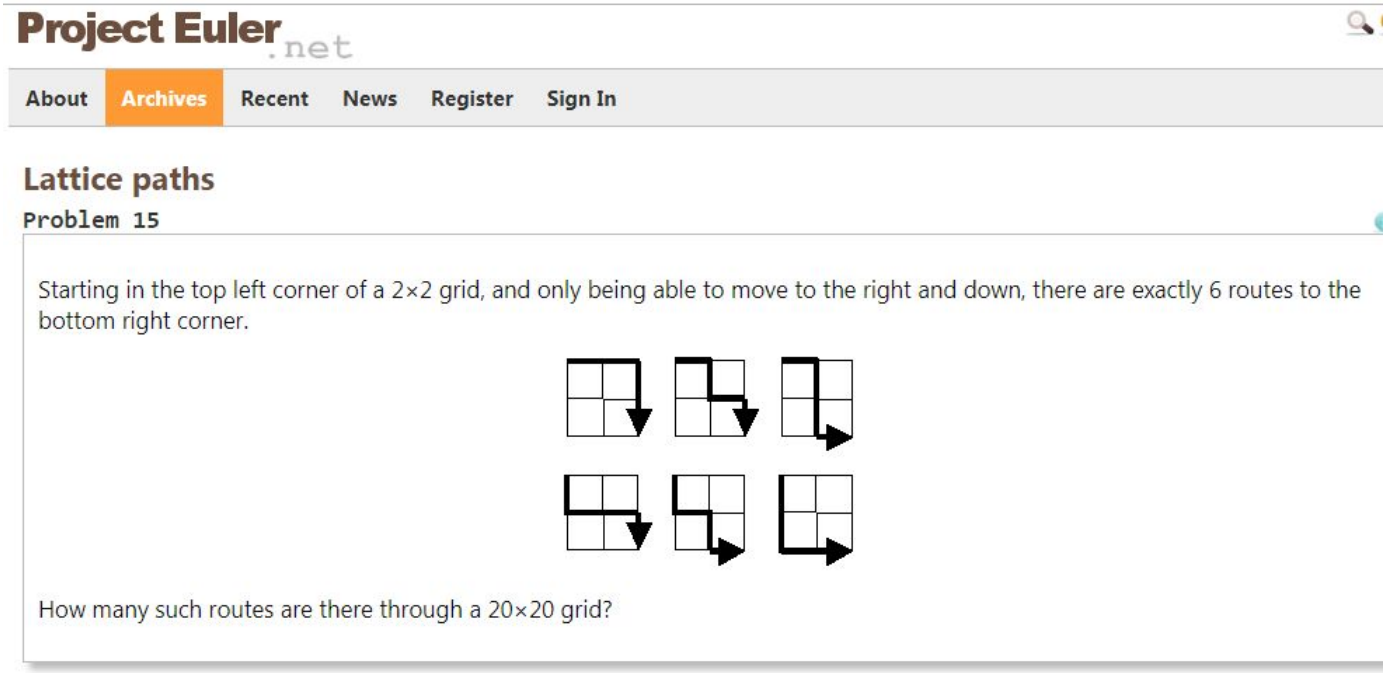
Para fazer isto via programa:

```
#include<stdio.h>

long int eulr(long int k) {
   if (k>2) {
  return ((4*k*k) - (6*(k-1)) + eulr(k-2));
   }
   else {
      return 1;
   }
}

long int euli(long int k) {
   int j, s=1;
   for (j=3;j<=k;j=j+2) {
      s=s+(4*j*j)-(6*(j-1));
   }
   return s;
}
main() {
   long int n,s=0;
   printf("Informe o tamanho da matriz (deve ser impar e maior que 2)\n");
   scanf("%ld",&n);
   printf("%ld \n",eulr(n)); // recursivo
```

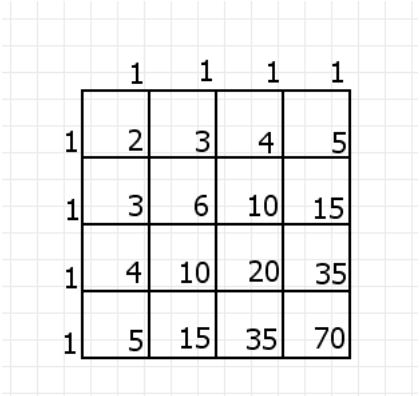
```
   printf("%ld \n",euli(n)); // iterativo, ambos dao o mesmo valor
}

Em PYTHON-----------------------------------------
# c35 - resolvedor da matriz na folha 035c
def eulr(k):
  if k>2:
    return ((k*k*4) - 6*(k-1)) + eulr(k-2)
  else:
    return 1

def euli(k):
  s=1
  for j in range(3,k+1,2):
    s=s+(4*j*j)-(6*(j-1))
  return s

def main():
  n=int(input("informe o tamanho da matriz (impar e >2) "))
  r=eulr(n)
  i=euli(n)
  print("recursivo e interativo " +str(r)+"  "+str(i))
```

Caminho na treliça

É o problema 15 do `projecteuler.net`. Diz:



Considere o diagrama:



 ©75-21, Pedro Kantek

Cada número nesta treliça representa o número de rotas até aquele nodo ou ponto (que vem a ser a intersecção na treliça).

Por exemplo, considere o nodo que tem 15 rotas a ele na linha mais embaixo. Há 10 rotas para o nodo acima dele e há 5 rotas para o nodo à esquerda dele. Então há 10+5=15 rotas para ele no total. Desse mesmo jeito, cada um dos outros valores foi calculado e fica evidente que o número de rotas diferentes em uma treliça 4×4 é 70.

Programado e ajeitado em 29/04/14, por P. Kantek

```
def trelica(n):
    import numpy
    t=numpy.zeros((n,n))
    for i in range(n):
 for j in range(n):
   if i==0:
      t[i][j]=1
   if j==0:
       t[i][j]=1
    for i in range(1,n):
 for j in range(1,n):
   t[i][j]=t[i-1][j]+t[i][j-1]
#    print(int(t[n-1][n-1]))
    print(t)
```

2.12 Movimentos no 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, sugriu 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 ? O xadrez já foi chamado de *drosóphila melanogaster* da IA, similar à importância da mosca doméstica na genética.

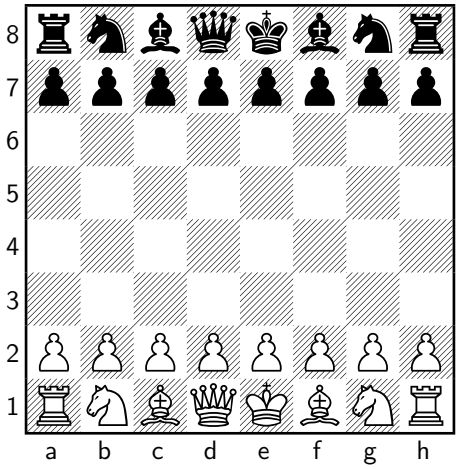
Abstraindo a computação, ainda assim o xadrez é um mundo maravilhoso. Aqui no Brasil, não temos grande tradição nele, mas em uma boa parte do mundo, as crianças aprendem a jogar e jogam durante toda sua vida escolar e por conseguinte muitos adultos seguem jogando vida afora. Isto, desde o século XII, que registra as primeiras manifestações do jogo, que por sinal é muito mais antigo do que isso.

Ele representa 2 exércitos que se opõe 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 raínha 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.



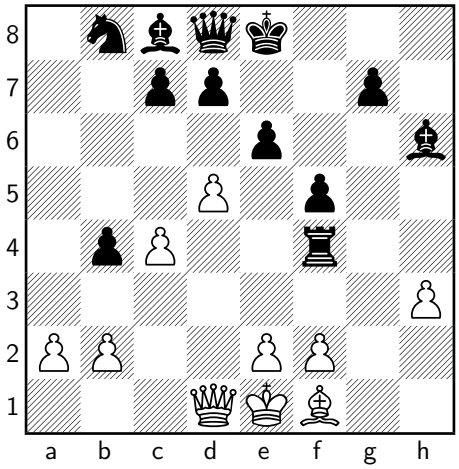
Tal disposição é boa para humanos, mas computadores precisam outra coisa: uma matriz numérica de 8×8, com casa brancas e negras alternadas sendo que que a casa embaixo, mais à direita, é branca. Por ser numérica, a matriz usa números para identificar as peças. Primeiro, a cor. Peças brancas são positivas e negras, negativas. Os valores: 1=peão, 2=torre, 3=cavalo, 4=bispo, 5=dama e 6=rei. 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.

 ©75-21, Pedro Kantek

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.

2.13 Sistemas Lineares e matriz inversa

O sistema

$$\left\{\begin{array}{lcl} a_{11}\ x_1 + a_{12}\ x_2 + ... + a_{1n}\ x_n & = & b_1 \\ a_{21}\ x_1 + a_{22}\ x_2 + ... + a_{2n}\ x_n & = & b_2 \\ ... & & \\ a_{m1}\ x_1 + a_{m2}\ x_2 + ... + a_{mn}\ x_n & = & b_m \end{array}\right.$$

é equivalente à seguinte equação matricial

$$\left(\begin{array}{l} a_{11}\ a_{12}\ ...\ a_{1n} \\ a_{21}\ a_{22}\ ...\ a_{2n} \\ ... \\ a_{m1}\ a_{m2}\ ...\ a_{mn} \end{array}\right) \left(\begin{array}{c} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{array}\right) = \left(\begin{array}{c} b_1 \\ b_2 \\ . \\ . \\ . \\ b_m \end{array}\right)$$

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 unidade.

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 $\left(\begin{array}{cc} 3 & 5 \\ 2 & 3 \end{array}\right)$

Uma possibilidade é querer encontrar escalares x, y, z e w para os quais
$\left(\begin{array}{cc} 3 & 5 \\ 2 & 3 \end{array}\right)\left(\begin{array}{cc} x & y \\ z & w \end{array}\right) = \left(\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right)$ ou
$\left(\begin{array}{cc} 3x+5z & 3y+5w \\ 2x+3z & 2y+3w \end{array}\right) = \left(\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right)$
ou ainda equações que satisfaçam $\left\{\begin{array}{l} 3x+5z=1 \\ 2x+3z=0 \end{array}\right.$ e $\left\{\begin{array}{l} 3y+5w=0 \\ 2y+3w=1 \end{array}\right.$

 ©75-21, Pedro Kantek

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 2×2 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 3×3 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).

$[AI] = \begin{vmatrix} a_{11} \ ... \ a_{1n} & 1 \ 0 \ ... \ 0 \\ a_{21} \ ... \ a_{2n} & 0 \ 1 \ ... \ 0 \\ ... & ... \\ a_{n1} \ ... \ a_{nn} & 0 \ 0 \ ... \ 1 \end{vmatrix}$ Aplica-se agora a este arranjo as técnicas de eliminação de Gauss, sobre o conjunto todo,

de maneira a ficar com o seguinte arranjo $\begin{vmatrix} 1 \ 0 \ ... \ 0 & b_{11} \ ... \ b_{1n} \\ 0 \ 1 \ ... \ 0 & b_{21} \ ... \ b_{2n} \\ ... & ... \\ 0 \ 0 \ ... \ 1 & b_{n1} \ ... \ b_{nn} \end{vmatrix}$

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 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 $A^{-1} \times B = x$. Vamos ver um exemplo do que se diz aqui. Seja um sistema, dado por

$$\left\{ \begin{array}{lll} 3x + 2y + z & = & \\ x - y + z & = & \\ 5x + z & = & \end{array} \right\}$$

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 $A^{-1} \times B$ 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:
$a[1][] = a[1][] + a[0] \times (-a[1][0]/a[0][0])$ onde a notação $a[1][]$ deve ser entendida como a aplicação da operaçao à toda a linha [1] da matriz a. Mas, atenção: isto não vale no compilador C++. 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 $a[2][] = a[2][] + a[0] \times (-a[2][0]/a[0][0])$.

Para a segunda coluna, começa-se achando o zero na primeira linha. O comando agora é
$a[0][] = a[0][] + a[1][] \times (-a[0][1]/a[1][1])$ e depois
$a[2][] = a[2][] + a[1][] \times (-a[2][1]/a[1][1])$. A terceira coluna é
$a[0][] = a[0][] + a[2][] \times (-a[0][2]/a[2][2])$ e
$a[1][] = a[1][] + a[2][] \times (-a[1][2]/a[2][2])$.
Com estas operações, os zeros foram alcançados. Agora basta transformar os elementos da diagonal principal na unidade. As operações são:
$a[0][] = a[0][]/a[0][0]$, $a[1][] = a[1][]/a[1][1]$ 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.

2.14 Truques variados de programação

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. A 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++ ou o Python fazem.

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.

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.

```
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. 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))**0.5)
    i=i+1
```

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).

```
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
```

 ©75-21, Pedro Kantek

```
  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 ; ctfl=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):
      ctfl=ctfl+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 = ",ctfl)
  print("Quadra = ",ctqu)
  print("Full hand = ",ctfh)
  print("Sequencia = ",ctse)
  print("Trinca = ",cttr)
```

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)
```

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)
```

 ©75-21, Pedro Kantek

```
    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.

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)}
```

Números crescentes Vamos chamar de crescente um número natural $n = d_1 d_2 ... d_k$ 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

```
digs=[] # conterá 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)
```

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.
```

```
import numpy as np
def divs(x):
  soma=1
```

 ©75-21, Pedro Kantek

```
  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.

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 que contém a quantidade de habitantes da cidade em cada quarteirão. Assim, se a posição 2,2 da matriz tem 50, significa que no quarteirao 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)
```

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.

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.

 ©75-21, Pedro Kantek

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 líquefeita, 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 $hectometro^2$, 1 hm=100m) em hm, em relação ao nivel 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 as 15h00. Deve imprimir também qual a altura mínima da lava nesta hora, em relação ao nivel 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
```

```
  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.

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
```

```
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)
```

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.

...

 ©75-21, Pedro Kantek

Truques: O primeiro é o conceito de conjunto que 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)
```

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.

```
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).

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.

 ©75-21, Pedro Kantek

```
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]=-9999999999999.0
      for jj in range(10):
 if raios[jj]>maximo[ii]:
 maximo[ii]=raios[jj]
    menor=9999999999999.0
    for ii in range(10):
      if maximo[ii]<menor:
 menor=maximo[ii]
 imenor=ii
    ctd=ctd+imenor+1
    i=i+1
  print("So= ",ctd)
```

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.

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 é 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  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
```

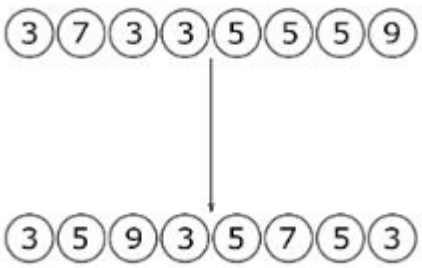
```
  t=np.zeros((8,8),int)
  i=spb=spn=0
  while i<100:
    for jj in range(8):
```

 ©75-21, Pedro Kantek

```
      x=f.readline()
      nums=x.split()
      for kk in range(8):
 t[jj,kk]=int(nums[kk])
    pb=pn=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)
```

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.

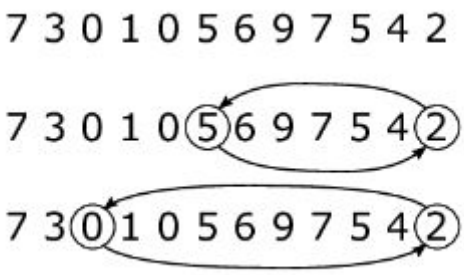
Saída Imprima uma linha contendo o caractere "S"se for possível trocar bolas de posição e obter a sequência sem bolas vizinhas com o mesmo número; ou o caracter "N"se não for possível.

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
```

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 730102697545, 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.

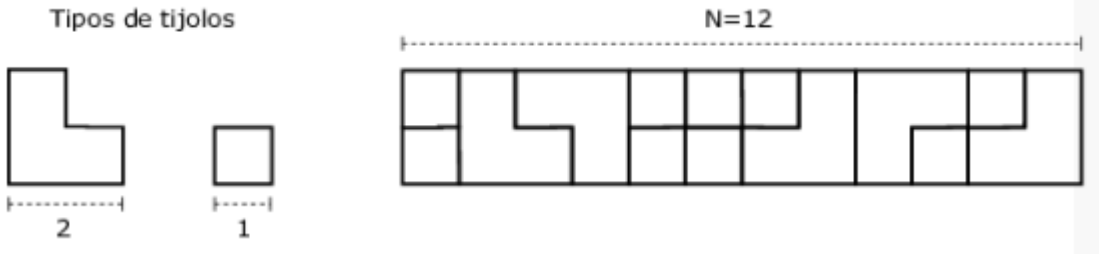
Entrada A primeira linha da entrada contém um inteiro N, indicando quantos dígitos decimais tem o número não divisível por 5. A segunda linha contém N inteiros $D_i, 1 \leq i \leq N$, representando os dígitos decimais do número em questão.

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.

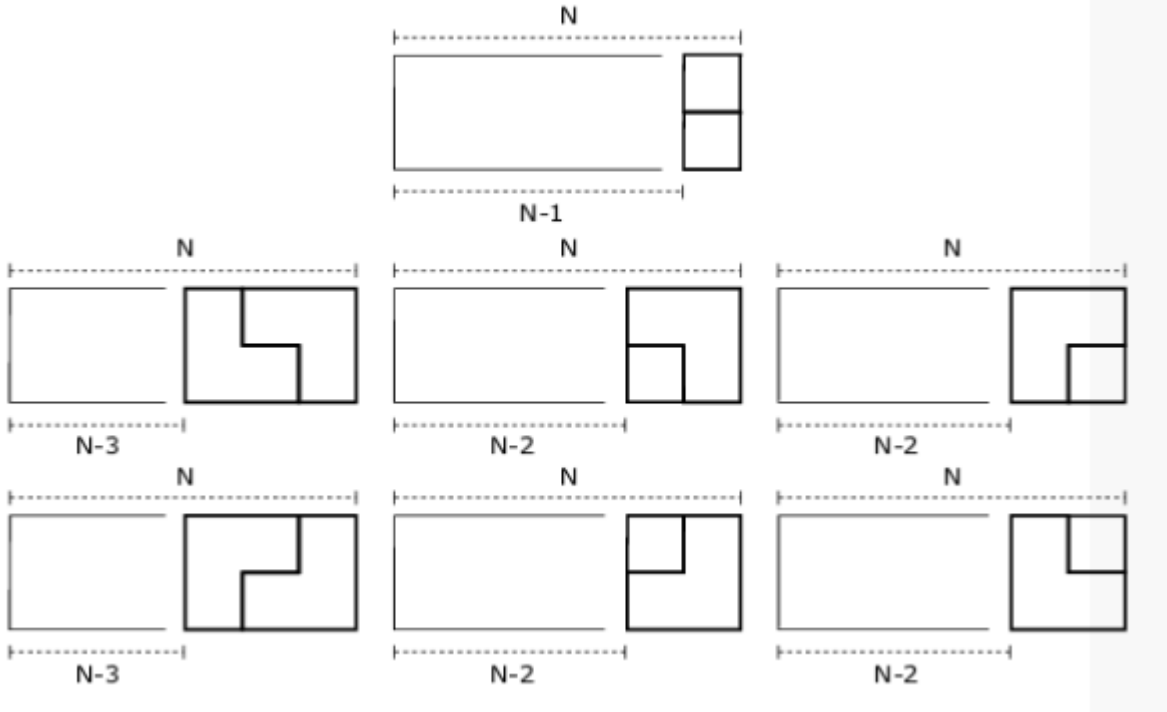
Restrições $2 \leq N \leq 1000$ e D_i é um inteiro entre 0 e 9, inclusive.

```
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
```

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
5
Entrada
11
Saída
36543
Entrada
6
Saída
241
Entrada
0
Saída
1
Entrada
8712
Saída
```

```
844673301
```

```
d={0:1, 1:1, 2:5}
def muro(n):
  aa=d.get(n)
  if aa is None:
    d[n]=(muro(n-1)+4*muro(n-2)+2*muro(n-3))%(10**9+7)
    return d[n]
  else:
    return aa
print(muro(61))
```

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:

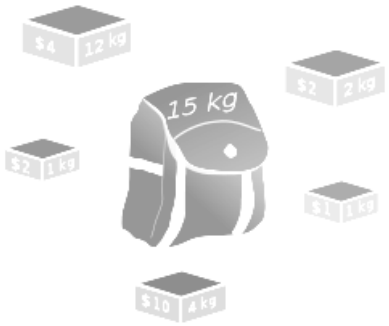
$$F_N = F_{N-1} + 4 \times F_{N-2} + 2 \times F_{N-3}$$

. 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.

2.15 Mochila 240a

O problema da mochila (em inglês, Knapsack problem) é um problema de otimização combinatória. O nome é devido ao modelo de uma situação em que é necessário preencher uma mochila com objetos de diferentes pesos e valores. O objetivo é que se preencha a mochila com o maior valor possível, não ultrapassando o peso máximo.

O problema da mochila é um dos 21 problemas NP-completos de Richard Karp, exposto em 1972. A formulação do problema é extremamente simples, porém sua solução é mais complexa.



Normalmente este problema é resolvido com programação dinâmica, obtendo então a resolução exata do problema. Foi possivelmente reportado pela primeira vez na literatura por Dantzig (1957) e constitui um marco das técnicas de programação inteira, otimização combinatória e programação dinâmica.

Podemos definir o problema matematicamente da seguinte forma:

Dados vetores $x[1..n]$ e $w[1..n]$, vamos denotar por $x.w$ o produto escalar de x por w:

$$x.w = x[1]w[1] + ... + x[n]w[n]$$

Suponha dado um número positivo ou nulo W e vetores $w[1..n]$ e $v[1..n]$ com componentes positivos ou nulos. Uma mochila é qualquer vetor $x[1..n]$ tal que

$x.w \leq W$ e $0 \leq x[i] \leq 1$ para todo i

O valor de uma mochila é o número $x \times v$. Uma mochila é ótima se tem valor máximo.

Imagine que você possui um contêiner de certo tamanho e vários produtos com seus valores para serem colocados dentro dele. Porém os produtos devem ser colocados de um jeito que o contêiner tenha o maior valor.

Suponha que você tenha em sua máquina uma pasta cheia de arquivos e deseja gravá-los em CD, só que você já sabe de antemão que não vai caber tudo num CD só. Podem ser dois, três... dez CD's. Como proceder para encaixar o máximo de arquivos em cada CD desperdiçando o mínimo espaço em cada disco?

Desses exemplos podemos entender que a motivação do problema da mochila é colocar dentro de um compartimento uma quantidade de objetos com determinadas importâncias para que esse compartimento tenha a maior importância possível.

O modelo em si pode ser aplicado, além nos exemplos citados acima, em casos como: Investimento de capital, corte e empacotamento, carregamento de veículos, orçamento.

Foi também utilizado para a construção do algoritmo de Criptografia de chave pública, onde no contexto do problema da mochila a chave pública seria o peso total que o mochila pode carregar e a partir dai, por essa palavra a encriptação é gerada.

Por ser um problema combinatório é possivel resolve-lo por enumeração exaustiva, ou seja tentar todas as soluções possiveis e comparando-as para identificar a melhor solução. Porém, obter todos os subconjuntos de um conjunto dado tem complexidade de $O(2^n)$. Esta abordagem pode tornar completamente inviável a solução uma vez que, mesmo em pequenas dimensões como um problema com apenas 20 itens, haveria um numero enorme de respostas. Em um problema com mais de 80 itens, o algoritmo levaria bilhões de anos para ser concluido. Assim, o método de resolução por enumeração exaustiva é de utilidade muito reduzida, senão mesmo nula, para problemas reais.

Outras estratégias são o *backtracking* (enumeração inteligente através da construção de uma árvore), o *método guloso* (é um macete que sempre escolhe a melhor alternativa. É rápido, mas nem sempre dá certo). A estratégia a ser usada aqui é a da programação dinâmica.

É um método aprimorado de usar recursividade que ao invés de chamar uma função várias vezes ele, na primeira vez que é chamado, armazena o resultado para que cada vez que a função for chamada novamente volte o resultado e não uma requisição para ser resolvida.

Exemplo: Suponha $W = 50$ e $n = 4$. A figura abaixo dá os valores de $w[1..4]$ e $v[1..4]$. Mais abaixo, temos uma solução $x[1..4]$ do problema fracionário e uma solução $x'[1..4]$ do problema booleano. O valor da mochila fracionária é $x.v = 1040$. O valor da mochila booleana é $x'.v = 1000$.

w	40	30	20	10
v	840	600	400	100
x	1	0.333	0	0
x'	0	1	1	0

Algoritmo O problema da mochila booleana pode ser resolvido por um algoritmo de programação dinâmica [CLR 17.2-1] que consome $O(nW)$ unidades de tempo. Mas isso só é possível se W e $w[1...n]$ são todos inteiros não-negativos.

O consumo $O(nW)$ é bem melhor que o do algoritmo recursivo, mas não é nenhuma maravilha, porque é proporcional a W.

A tabela t contém as soluções das instâncias do problema. A tabela é definida assim:

$t[i, Y]$ é o valor máximo da expressão $x[1..i].v[1..i]$ sujeita à restrição $x[1..i].w[1..i] \leq Y$.

Em português de gente, t[i][j] contém o valor da mochila composta pelos melhores i primeiros produtos, cujo peso é menor ou igual a j.

Os possíveis valores de Y são $0, 1, ..., W$ (É claro que isso só funciona porque estamos supondo W inteiro) e os possíveis valores de i são $0, 1, ..., n$. É importante que 0 seja um possível valor de Y e de i. Por exemplo, se w[1]=0 então t[1,0]=v[1].

Pode-se ver que t[0,Y]=0 para todo Y. Se $i > 0$ então $t[i, Y] = A$ se $w[i] > Y$ e $t[i, Y] = max(A, B)$ se $w[i] \leq Y$,

onde $A = t[i - 1, Y]$ e $B = t[i - 1, Y - w[i]] + v[i]$. Traduzindo para o português:

Se a mochila ótima para 1..i não usa i então ela é também uma mochila ótima para 1..i-1. Se a mochila ótima para 1..i usa i então ela é o resultado de juntar i com uma mochila ótima para 1..i-1.

A partir dessa recorrência é possível escrever um algoritmo de programação dinâmica para determinar t e o vetor x:

```
1: Mochila-Booleana (w, v, W)
2: n ← dimensão de w
3: for Y ← 0 até W do
4:   t[0][Y] ← 0
5:   for i ← 1 até n do
6:     A ← t[i-1][Y]
7:     if w[i] > Y then
8:       B ← 0
9:     else
10:      B ← t[i-1][Y-w[i]]+v[i]
11:    end if
12:    t[i][Y] ← max(A,B)
13:  end for
14: end for
15: Y ← W
16: for i ← n decrescendo até 1 do
17:   if t[i][Y] = t[i-1][Y] then
```

```
18:     x[i] ← 0
19:   else
20:     x[i] ← 1
21:     Y ← Y-w[i]
22:   end if
23: end for
24: devolva x, t[n][W]
```

Exemplos:

```
w=1 5 1 1 6 6 8 8;v=2 21 3 13 13 11 19 4;W=14
x=0 1 0 1 0 0 1 0;Y=53

w=1 3 4 2 4 2 3 3;v=2 14 13 17 21 22 22 9;W=13
x=1 0 0 1 1 1 1 0;Y=84

w=7 4 7 8 4 3 4 4;v=3 5 5 17 7 16 23 4;W=15
x=0 0 0 1 0 1 1 0;Y=56
```

Uma possível solução

```
def f240(w,v,W):
    import numpy
    print('--------------------------------------')
    print('w= ',w)
    print('v= ',v)
    print('W= ',W)
    w=[0]+w
    v=[0]+v
    t=numpy.zeros((len(w),100))
    x=[-1]*len(w)
    Y=0
    while Y<=W:
 t[0][Y]=0
 i=1
 while i<len(w):
   A=t[i-1][Y]
   if w[i]>Y:
       B=0
   else:
       B=t[i-1][Y-w[i]]+v[i]
   if A>=B:
       t[i][Y]=A
   else:
       t[i][Y]=B
   i=i+1
 Y=Y+1
    Y=W
    i=len(w)-1
    while i>=1:
 if t[i][Y]==t[i-1][Y]:
   x[i]=0
 else:
   x[i]=1
   Y=Y-w[i]
 i=i-1
    print(x[1:],t[len(w)-1][W])
    print('Conteudo da mochila =     Peso')
    for i in range(1,len(x)):
 if x[i]==1:
   print(w[i],end=' ')

    print('\nConteudo da mochila =     valor')
    for i in range(1,len(x)):
 if x[i]==1:
```


©75-21, Pedro Kantek

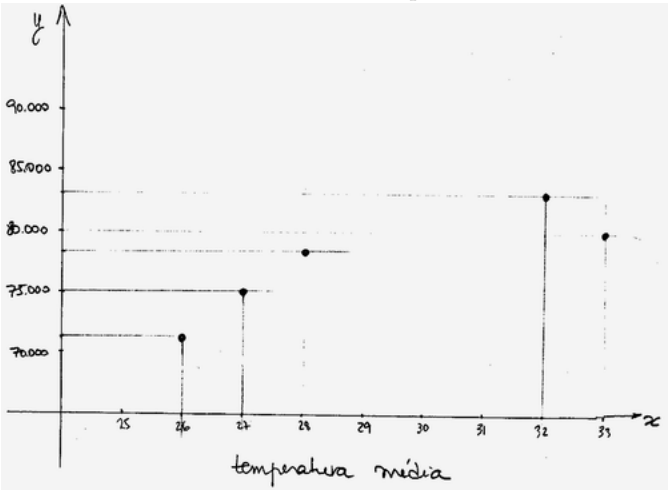
```
   print(v[i],end=' ')
    print('\n')
w=[1,5,1,1,6,6,8,8]
v=[2,21,3,13,13,11,19,4]
W=14
f240(w,v,W)
w=[1,3,4,2,4,2,3,3]
v=[2,14,13,17,21,22,22,9]
W=13
f240(w,v,W)
w=[7,4,7,8,4,3,4,4]
v=[3,5,5,17,7,16,23,4]
W=15
f240(w,v,W)
```

2.16 Mínimos quadrados

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(^{o}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

$$na + (\sum_{i=1}^{n} x_i)b = \sum_{i=1}^{n} y_i$$

e

$$(\sum_{i=1}^{n} x_i)a + (\sum_{i=1}^{n} x_i^2)b = \sum_{i=1}^{n} x_i y_i$$

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

$$5a + 146b = 387$$

$$146a + 4302b = 11351$$

Da primeira equação fica $a = 77,4 - 29.2b$. Levando este valor na segunda equação, fica

$$38,8b = 50,6$$

ou

$$b = 1,30$$

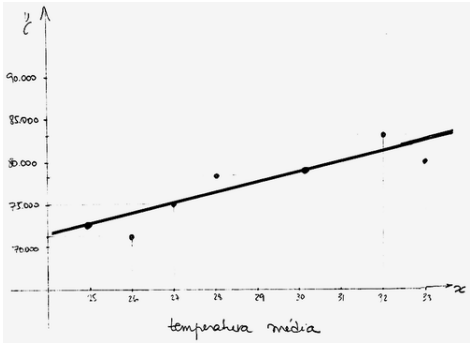
. Retornando à primeira equação para obter o valor de a, tem-se $a = 39,44$.

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

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:

$Na + Jb = K$ (eq. 1) e

$Ja + Lb = M$ (eq. 2)

Multiplicando a eq.1 por J e a eq.2 por N, fica:

$JNa + J^2b = KJ$ (eq. 3) e

$JNa + LNb = NM$ (eq. 4)

multiplicando a eq.4 por -1 e somando as duas, fica:

$(J^2 - LN)b = KJ - NM$ (eq.5) ou

$b = \frac{KJ-NM}{J^2-LN}$ (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) obtem-se o valor de a, o que permite reconstruir a equação da reta procurada. Agora pode-se extrapolar quaisquer 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: while X ≠ -1 do
8:    CT ← CT + 1
9:    VX[CT] ← X
10:   escreva ("digite o par de ",X)
```

 ©75-21, Pedro Kantek

11: leia (Y)
12: VY[CT] ← Y
13: escreva ("digite mais um X ou -1 para encerrar")
14: leia (X)
15: **end while**
16: J ← K ← L ← M ← N ← 0
17: **for** IND=1; IND < CT; IND++ **do**
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: **end for**
24: B ← $\frac{KJ-NM}{J^2-LN}$
25: A ← $\frac{K-JB}{N}$
26: leia (X)
27: **while** X ≠ -1 **do**
28: Y ← $A + B \times X$
29: escreva (Y)
30: leia (X)
31: **end while**

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
```

2.17 Recursividade: passeio cavalo, caixeiro viajante, hanoi

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:

- $h[x, y] = 0$ Indicará que a célula x, y ainda não foi ocupada.
- $h[x, y] = i$ Indicará 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: repeat
13:    RESP ← FALSO
14:    LP ← LCAV + DLIN [K]
15:    CP ← CCAV + DCOL [K]
16:    if (LP ≥ 1) ∧ (LP ≤ n) ∧ (CP ≥ 1) ∧ (CP ≤ n) then
17:       if MATRIZ [LP] [CP] = 0 then
18:          MATRIZ [LP] [CP] ← NUMMOV
19:          if NUMMOV < (n^2) then
20:             RESP ← ANDACAVALO (NUMMOV+1, LP, CP)
21:             if (RESP = FALSO) then
22:                MATRIZ [LP] [CP] ← 0
23:             end if
24:          else
25:             RESP ← VERDADEIRO
26:          end if
27:       end if
28:    end if
29:    K++
30: until (K > 8) ∨ (RESP = VERDADEIRO))
31: devolva (RESP)
32: fim função
33: MATRIZ ← 0
34: MATRIZ [linha inicial] [coluna inicial] ← 1
35: if (ANDACAVALO (2, linha inicial, coluna inicial)) then
36:    imprima MATRIZ
37: else
38:    imprima "Não existe caminho possível"
39: end if
40: fim algoritmo
```

 ©75-21, Pedro Kantek

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.

Em python,

```
import time
import numpy as np
global mat
mat=np.zeros((8,8),int)
global ct
ct=0
def andacavalo(nummov, lcav, ccav):
  global ct
#  print(mat,nummov)
#  print('----------------------------------')
  dlin=[-2,-1,1,2,2,1,-1,-2]
  dcol=[1,2,2,1,-1,-2,-2,-1]
  k=0
  while 1==1:

    resp=0
    lp=lcav+dlin[k]
    cp=ccav+dcol[k]
    if lp>=0 and lp<8 and cp>=0 and cp<8:
      if mat[lp,cp]==0:
 mat[lp,cp]=nummov
 if nummov<62:
 ct=ct+1
 resp=andacavalo(nummov+1,lp,cp)
 if resp==0:
   mat[lp,cp]=0
 else:
 resp=1
    k=k+1
    if k>7 or resp!=0:
      break
  return resp

mat[4,4]=1
ini=time.time()
if andacavalo(2,4,4)==1:
  print(ct)
  print(mat)
fim=time.time()
print(fim-ini)
```

Dados reais A seguir, uma configuração real, com dados de processamento em C++ e em Python, até para haver comparação. O resultado numérico – por óbvio – é igual, mas os dados de desempenho não são. Ambas as soluções foram:

```
[[53 34 55 30 51 32 15 18]
 [56 49 52 33 16 19  6 13]
 [35 54 29 50 31 14 17  4]
 [48 57 36 41 20  5 12  7]
 [37 28 47 58 43 22  3 64]
 [46 59 42 21 40 11  8 23]
 [27 38 61 44 25  2 63 10]
 [60 45 26 39 62  9 24  1]]
```

Em ambas houve 27.241.111 chamadas recursivas. A diferença está no tempo de execução. C++ demorou 2.75 segundos enquanto Python demorou 272.1 segundos. Praticamente 100 vezes mais. É obvio, mas vale informar: é o mesmo computador, a mesma memória e tanto quanto possível o mesmo ambiente para ambos.

Algoritmo de Strassen O algoritmo de Strassen preve 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)
```

 ©75-21, Pedro Kantek

```
    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 reais, o algoritmo de Strassen (com $n > 4$) demorou 13.8 seg e o algoritmo canônico 18.5 seg. Já para matrizes de 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 $O(n.logn)$ 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 $tam(esquerdo) > k$, então a mediana está na lista esquerda. Aplicar $QS(esquerdo)$.
4. subtrair de k o tamanho da lista esquerdo
5. Calcular o número de duplicações da mediana, fazendo $tam(lista) - (tam(esquerdo) + tam(direito))$
   - 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[pivot],arr[begin]=arr[begin],arr[pivot]
  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
```

2.18 A*

Relembrando o que se viu atrás, o problema fundamental na informática é: *dada uma chave recuperar os dados associados a essa chave*. Em miúdos: dado um CPF recuperar as transações deste CPF em algum contexto. Aplicando métricas, suponhamos que há n chaves e s espaço de armazenamento. Antes de prosseguir, declare-se que nos últimos anos (décadas), o tamanho tanto de n quanto de s explodiram. Está aí o espaço de armazenamento da Internet que não deixa mentir. Mesmo em instalações pessoais, há não muito tempo a unidade de armazenamento era o Megabyte. Passamos ao Gigabyte (1024 vezes maior) e ao Terabyte (mais 1024 vezes maior). Acho que ainda veremos o Petabyte (mais 1024 maior) por aí.

Mas, o ponto agora é ainda maior do que isso. Há espaços de armazenamento que extrapolam esses números, mesmo que eles cresçam explosivamente. Vamos a um exemplo real. Suponhamos um jogador de xadrez. A chave k aqui é uma situação de tabuleiro e o retorno esperado é qual a melhor resposta a esta situação. Supondo uma tabela com todas as possibilidades de jogo, a busca e recuperação seria – em tese – bem simples. A questão é: **qual seria o tamanho dessa tabela ?**. Ninguém sabe ao certo, mas há uma estimativa (em [RUS03]) de que tal tabela teria 10^{120}

itens de tamanho. Mostrado dessa maneira não parece muito impressionante, mas por extenso o número é mais forte: 100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000, ou seja, um '1' seguido de 120 zeros.

Apenas para comparar, vamos a alguns outros conceitos: Comecemos com o número de Avogadro que é a quantidade de moléculas em um mol de qualquer substância. Ou seja 18 gramas (ou 18 ml) de água, ou 2 gramas de Hidrogêneo ou 16 gramas de Oxigênio. Ele indica a quantidade de átomos nos 2 últimos casos. Ou de moléculas no primeiro. É de 6.02×10^{23} ou por extenso 602000000000000000000000. Agora imagine-se o universo conhecido: bilhões de galaxias e em cada galaxia bilhões de estrelas e seus sistemas planetários e neles os planetas. Agora imagine quantos átomos haverá nesse universo. E, depois quantos eletrons haverá aí. Pois este número de elétrons existentes no nosso Universo está estimado em 10^{80} eletrons ou 100000000000000000000000000000000000000000000000000000000000000000000000000000000.

Então, se cada eletron existente no universo fosse usado para armazenar uma posição no xadrez (como ?) ainda assim haveria necessidade de 10^{40} universos distintos (ou 10000000000000000000000000000000000000000 universos) para poder representar as posições do jogo.

Claramente, há uma impossibilidade de escrever toda a tabela. Há que navegar pelo jogo, sem representar tudo. Aqui entram os algoritmos "inteligentes" de busca, dos quais o mais conhecido e simples é o A^*.

A* O algoritmo A*, criação de Hart, Nilsson e Raphael apareceu em 1968. Um artigo de 1972, de Hart, corrigiu alguns pequenos equívocos da apresentação original. A idéia é juntar na mesma busca, vantagens da busca gulosa (sempre a opção mais apetitosa) e da busca em largura. A primeira é eficiente, mas pode chegar a um beco sem saída. A segunda sempre chega à resposta, mas pode ser tremendamente ineficiente. Usando A*, a função de avaliação do próximo candidato a expandir na árvore de busca é

$$\boxed{f(n) = k_1 \times p(n) + k_2 \times h'(n)}$$

Sem perda de generalidade, pode-se inicialmente, considerar $k_1 = k_2 = 1$ e com isso a fórmula com que trabalharemos, por enquanto, é:

$$\boxed{f(n) = p(n) + h'(n)}$$

Trocando em miúdos, para cada nodo na árvore, dever-se-á calcular sua profundidade, e também a heurística do nodo. O campeão (ou seja, o nodo que vai ser expandido) é aquele que tiver o menor valor de $f(n)$.

$p(n)$ calcula o valor do custo necessário para ir do nodo inicial até o nodo em questão. Já $h'(n)$ estima o custo necessário para ir do nodo atual até um estado meta. Como o objetivo é minimizar o caminho total, deve-se expandir o nodo que tiver a menor soma de ambas parcelas. Nesse sentido, $f(n)$ pode ser entendido como uma estimativa do menor custo para ir do estado inicial até um estado meta, passando por n.

Se se conseguir impor uma pequena restrição à função $h'(n)$, pode-se afirmar que a busca A* é *completa* e *ótima*. A restrição é de que deve-se escolher uma função h' que nunca sobreestime o esforço necessário para alcançar a meta. Ou seja, a função deve ser otimista: pode errar para menos no esforço que será necessário, mas nunca poderá errar para mais.

Um bom exemplo para esta função, no caso de um problema que visa determinar qual o melhor caminho rodoviário entre duas cidades, poderia ser a distância linear entre cada ponto parcial e o destino. Perceba que esta medida erra para menos, mas nunca pode errar para mais.

Comecemos com um exemplo simples: Este exercício vai oferecer uma matriz quadrada numérica e solicitar qual a menor soma de vizinhos contíguos saindo do elemento mais alto à esquerda até o mais baixo à direita. Ao fazer a soma, pode-se andar (somar) para qualquer um dos quatro vizinhos da célula atual. Este exercício está baseado no problema 83 do projeto Euler (projecteuler.net). Assim, por exemplo, na 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
```

O caminho mais barato é: 131 + 201 + 96 + 342 + 234 + 103 + 18 + 150 + 111 + 422 + 121 + 37 + 331 = 2297. O referido caminho fica mais visível em

O fato do caminho poder voltar à esquerda inviabiliza o algoritmo da programação dinâmica.

Embora existam diversos algoritmos para calcular este caminho vai se usar aqui o algoritmo A* ligeiramente modificado.

Relembrando, o algoritmo A* está baseado na idéia de custo de um nodo, que sempre é calculado fazendo-se

$$C = F + H$$

onde C é o custo de um nodo, F é o valor real desde o início até o nodo em análise e H é uma heurística estimadora do caminho que falta para chegar até o final.

Neste caso, F é o valor real da soma dos valores da matriz desde a casa inicial (1,1) até a casa que se está examinando. Quanto a H, usaremos como função heurística o menor valor de custos da matriz original multiplicado pela distância manhattan da casa atual até o final.

Para entender este conceito, examine a matriz H do exemplo acima:

```
162 144 126 108  90
144 126 108  90  72
126 108  90  72  54
108  90  72  54  36
 90  72  54  36  18
```

Note que o menor valor da matriz original é 18, e este valor foi sendo multiplicado pela quantidade de casas necessárias a percorrer até o final da matriz. Eis a variavel LA do exemplo acima

```
    lin col  som      custo   pai
    --- --- ----  ----------- ---
 1-   1   1   131  10000000293  -1
 2-   1   2   804  10000000948   1
 3-   2   1   332  10000000476   1
 4-   2   2   428  10000000554   3
 5-   3   1   962  10000001088   3
 6-   2   3   770  10000000878   4
 7-   3   2  1231  10000001339   4
 8-   1   3  1004  10000001130   6
 9-   2   4  1735  10000001825   6
10-   3   3  1516  10000001606   6
11-   4   1  1499  10000001607   5
12-   1   4  1107  10000001215   8
13-   1   5  1125  10000001215  12
14-   2   5  1275  1000000l347  13
15-   4   2  1930  2020    7
16-   3   5  1386  10000001440  14
17-   4   5  2342  2378   16
18-   3   4  1808  10000001880  16
19-   4   3  2013  2085   10
20-   5   1  2304  2394   11
21-   4   4  1929  10000001983  18
22-   5   4  1966  10000002002  21
23-   5   5  2297  2315   22
```

Mais um exemplo

Seja uma matriz de ordem 14 a saber:

```
      1   2   3   4   5   6   7   8   9  10  11  12  13  14
    --- --- --- --- --- --- --- --- --- --- --- --- --- ---
 1|   70 149 101 892 457 539 404 230 254 321 833 902 962 592
 2|  929 159 415 466 507  24 231 222 493 630 543 951 215 290
 3|  729 517 916  72 261 688 316 980 354 842  91 333 613 471
 4|  165  77 713 484 525 172  27 812 766 167 410 998 933  95
 5|  509 881 977 282 402 990  93 955 198 979  45 885 889  79
 6|  548 105 569 320 558 423 452  46 367 875 153 285 584 618
 7|  446 417 658 606 249  53 935 473 264  23 771 148 827 770
 8|  783 697   9 343 381 942 210  88 182 536  13 667 874 538
 9|  531  47 819 345 331 125 992 600  69 194 954 604 246 563
10|  969 835 708 499 436 756 409 693 628  61 475 869 219 635
11|  828 769 786 442 138 679  73 186 145  96 128 137 247 431
12|  975 274 903 777   8  26 734 614 508 549 272 946 792 896
13|  831 386  97 595 853 256 774 414 968 801 991 545 883 656
14|  166 258 989 799 152   5 155 554 681 912   7 356 820 529
```


©75-21, Pedro Kantek

Nesta matriz o menor custo é 7036, com 27 casas percorridas, a saber: 14 14, 13 14, 12 14, 11 14, 11 13, 11 12, 11 11, 11 10, 10 10, 9 10, 9 9, 8 9, 8 8, 7 8, 6 8, 6 7, 5 7, 4 7, 4 6, 4 5, 3 5, 3 4, 2 4, 2 3, 1 3, 1 2, 1 1. Eis este algoritmo em Python:

```
def vivo831():
  t=50
  ICA = []
  f=open("c:/p/n/831/EXEA00000",'r')
  matrix = [[0 for i in range(t)] for j in range(t)]
  for i in range(t):
    ll = f.readline()
    matrix[i] = [int(x) for x in ll.split()]
  inf = 1000000000
  men = min(min(matrix))
  h = [[men for col in range(t)] for row in range(t)]
  i=t
  while (i>=1):
    j=t
    while (j>=1):
      h[i-1][j-1]=h[i-1][j-1]*((-1)+(t-(i-1))+(t-(j-1)))
      j=j-1
    i=i-1
  la=[]
  zica=[]
  proximo=1
  la.append([0,0,matrix[0][0],(matrix[0][0]+h[0][0]),-1])
  zica.append(0)
  while (1==1):
    men=9999999999999999
    imen=0
    i=0
    while (i<len(la)):
      if la[i][3]<men:
 men=la[i][3]
 imen=i
      i=i+1
    la[imen][3]=la[imen][3]+inf
# processamento do filho 1
    f1=[la[imen][0]-1,la[imen][1]]
    if min(f1)>=0 and max(f1)<=t-1 and f1[0]*1000+f1[1] not in zica:
      zica.append(f1[0]*1000+f1[1])
      conta=la[imen][2]+matrix[f1[0]][f1[1]]
      la.append([f1[0],f1[1],conta,conta+h[f1[0]][f1[1]],imen])
      proximo=proximo+1
      if f1[0]==t-1 and f1[1]==t-1:
 break
# processamento do filho 2
    f2=[la[imen][0],la[imen][1]+1]
    if min(f2)>=0 and max(f2)<=t-1 and f2[0]*1000+f2[1] not in zica:
      zica.append(f2[0]*1000+f2[1])
      conta=la[imen][2]+matrix[f2[0]][f2[1]]
      la.append([f2[0],f2[1],conta,conta+h[f2[0]][f2[1]],imen])
      proximo=proximo+1
      if f2[0]==t-1 and f2[1]==t-1:
 break
# processamento do filho 3
    f3=[la[imen][0]+1,la[imen][1]]
    if min(f3)>=0 and max(f3)<=t-1 and f3[0]*1000+f3[1] not in zica:
      zica.append(f3[0]*1000+f3[1])
      conta=la[imen][2]+matrix[f3[0]][f3[1]]
      la.append([f3[0],f3[1],conta,conta+h[f3[0]][f3[1]],imen])
      proximo=proximo+1
      if f3[0]==t-1 and f3[1]==t-1:
 break
```

```
# processamento do filho 4
    f4=[la[imen][0],la[imen][1]-1]
    if min(f4)>=0 and max(f4)<=t-1 and f4[0]*1000+f4[1] not in zica:
      zica.append(f4[0]*1000+f4[1])
      conta=la[imen][2]+matrix[f4[0]][f4[1]]
      la.append([f4[0],f4[1],conta,conta+h[f4[0]][f4[1]],imen])
      proximo=proximo+1
      if f4[0]==t-1 and f4[1]==t-1:
 break
  ultimo=proximo-1
  print("Custo deste caso = "+str(la[ultimo][2]))
  vica=0
  while ultimo != -1:
#    print(la[ultimo][0]+1,la[ultimo][1]+1)
    ICA.append([la[ultimo][0]+1,la[ultimo][1]+1])
    ultimo=la[ultimo][4]
    vica=vica+1
  print("Numero de trechos "+str(vica))
# fazendo o caminho gráfico em CAM
  CAM = [[" " for i in range(t)] for j in range(t)]
  i=0
  while i<len(ICA)-1:
    j = ICA[i]
    k = ICA[i+1]
    if j[0]==k[0] and j[1]>k[1]:
      m=[k[0],k[1]]
      CAM[m[0]-1][m[1]-1]='>'
    if j[0]==k[0] and j[1]<k[1]:
      m=[k[0],k[1]]
      CAM[m[0]-1][m[1]-1]='<'
    if j[0]<k[0] and j[1]==k[1]:
      m=[k[0],k[1]]
      CAM[m[0]-1][m[1]-1]='^'
    if j[0]>k[0] and j[1]==k[1]:
      m=[k[0],k[1]]
      CAM[m[0]-1][m[1]-1]='v'
    i=i+1
  for i in range(1,len(CAM)-1):
    for j in range(1,len(CAM[i])-1):
      print(CAM[i][j], end=' ')
    print()
  print("acabou")

vivo831()
```

2.19 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

Apenas para reforçar o conceito de ordem, em espanhol, um computador é *un ordenador* e em francês é *un ordinateur*.

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 estabecer 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 $O(n \times log_2 n)$, ambas as expressões – como sempre – a menos das constantes multiplicativas.

O estudo da primeira família se justifica pelas seguintes razões:
1. Constantes multiplicativas eventualmente menores
2. Abundância de recursos de hardware
3. Irrelevância do desempenho para pequenas instâncias
4. 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 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". os algoritmos em C++:

```
#include<iostream>
using namespace std;

void bolha(int v[], int tam){
    int i,j;
    i=tam-1;
    while (i>=1){
       j=1;
       while(j<=i){
   if (v[j-1]>v[j]){
      swap(v[j-1],v[j]);
      }
   j++;
   }
       i--;
    }
}

void bolhaest(int v[], int tam){
    int i,j,mudou;
    i=tam-1;
    mudou=1;
    while ((i>=1)&&mudou){  // mudou==1
       j=1;
       mudou=0;
       while(j<=i){
   if (v[j-1]>v[j]){
      swap(v[j-1],v[j]);
```

```
      mudou=1;
      }
   j++;
   }
       i--;
    }
}

void inser(int v[], int n)
{
   int i, aux, j;
   for (i = 1; i < n; i++) {
       aux = v[i];
       j = i-1;
       while (j >= 0 && v[j] > aux) {
    v[j+1] = v[j];
    j = j-1;
       }
       v[j+1] = aux;
   }
}

void selec (int v[], int tam) {
   int i,j,cor,inui;
   i=0;
   while (i<tam){
      cor=v[i];
      inui=i;
      j=i+1;
      while (j<tam){
  if (v[j]<cor){
      cor=v[j];
      inui=j;
  }
  j++;
      }
      swap(v[i],v[inui]);
      i++;
   }
}

void shell(int v[], int tam){
   int k,h,i,j,x,m;
   int decr[3]={5,3,1};  // se quiser, pode alterar os tamanhos
   int tam2;
   tam2=sizeof(decr)/sizeof(decr[0]);
   for (k=0;k<tam2;k++){
      h=decr[k];
      i=h;
      while (i<tam){
  x=v[i];
  j=i;
  while ((j>=h) && (v[j-h]>x)){
     v[j]=v[j-h];
     j=j-h;
  }
  v[j]=x;
  i++;
      }
   }
}

void criaheap(int v[], int n, int i) {
```

 ©75-21, Pedro Kantek

```
    int maior = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && v[l] > v[maior]) {
 maior = l;
    }
    if (r < n && v[r] > v[maior]) {
 maior = r;
    }
    if (maior != i) {
 swap(v[i], v[maior]);
 criaheap(v, n, maior);
    }
}
void heapsort(int v[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--){
 criaheap(v, n, i);
    }
    for (int i=n-1; i>=0; i--) {
 swap(v[0], v[i]);
 criaheap(v, i, 0);
    }
}

void quick(int v[], int i, int f) {
   int ant, dep, mei;
   ant=i;
   dep=f;
   mei=v[(i+f)/2];
   do{
      while(v[ant]<mei){
  ant++;
  }
      while (v[dep]>mei){
  dep--;
  }
      if (ant<=dep) {
  swap(v[ant],v[dep]);
  ant++;
  dep--;
  }
   } while (ant<=dep);
   if (i<dep){
      quick(v, i,dep);
      }
   if (ant<f){
      quick(v,ant,f);
      }
   }

int main(){
   int ve1[]={7,3,1,9,11,23,6,6,19,10,33,44,55,23,24,25,26};
   int i,tam;
   tam = sizeof(ve1)/sizeof(ve1[0]);
   cout<<"bolha ";
   bolha(ve1,tam);
   for (i=0;i<tam;i++){
       cout<<ve1[i]<<' ';
   }
   int ve2[]={7,3,1,9,11,55,88,66,29,100,23,6,6,19,10};
   tam = sizeof(ve2)/sizeof(ve2[0]);
   cout<<endl<<"bolhaestrela ";
   bolhaest(ve2,tam);
```

```
   for (i=0;i<tam;i++){
      cout<<ve2[i]<<' ';
   }
   int ve3[]={5,7,1,3,2,4,8,6,9,57,68,79,54,33};
   tam = sizeof(ve3)/sizeof(ve3[0]);
   cout<<endl<<"insercao ";
   inser(ve3,tam);
   for (i=0;i<tam;i++){
      cout<<ve3[i]<<' ';
   }
   int ve4[]={22,23,7,3,1,9,11,23,6,6,19,10};
   tam = sizeof(ve4)/sizeof(ve4[0]);
   cout<<endl<<"selecao ";
   selec(ve4,tam);
   for (i=0;i<tam;i++){
      cout<<ve4[i]<<' ';
   }
   int ve5[]={7,3,1,9,11,23,6,6,19,10,8,88,888};
   tam = sizeof(ve5)/sizeof(ve5[0]);
   cout<<endl<<"shell ";
   shell(ve5,tam);
   for (i=0;i<tam;i++){
      cout<<ve5[i]<<' ';
   }
   int ve6[]={7,3,1,88,888,2,22,21,17,19,12,13,11};
   tam = sizeof(ve6)/sizeof(ve6[0]);
   cout<<endl<<"heap ";
   heapsort(ve6,tam);
   for (i=0;i<tam;i++){
      cout<<ve6[i]<<' ';
   }
   int ve7[]={2,22,26,29,7,3,1,91,9,11,23,6,6,19,10};
   tam = sizeof(ve7)/sizeof(ve7[0]);
   cout<<endl<<"quick ";
   quick(ve7,0,tam);
   for (i=0;i<tam;i++){
      cout<<ve7[i]<<' ';
   }
}
```

os mesmos algoritmos em Python:

```
# ordenações
def bolha(v):
  i=len(v)-1
  while i>=1:
    j=1
    while j<=i:
      if v[j-1]>v[j]:
 v[j-1],v[j]=v[j],v[j-1]
      j=j+1
    i=i-1
  return(v)

def bolhaest(v):
  i=len(v)-1
  mudou=True
  while i>=1 and mudou:
    j=1
    mudou=False
    while j<=i:
      if v[j-1]>v[j]:
 v[j-1],v[j]=v[j],v[j-1]
```

 ©75-21, Pedro Kantek

```
 mudou=True
      j=j+1
    i=i-1
  return(v)

def inser(v):
  v=v+[0]
  for i in range(1,len(v)):
    aux=v[i]
    j=i-1
    while j>=0 and v[j]>aux:
      v[j+1]=v[j]
      j=j-1
    v[j+1]=aux
  return v[1:]

def selec(v):
  i=0
  while i<len(v):
    cor=v[i]
    inui=i
    j=i+1
    while j<len(v):
      if v[j]<cor:
 cor=v[j]
 inui=j
      j=j+1
    v[i],v[inui]=v[inui],v[i]
    i=i+1
  return v

def shell(v):
  decr=[5,3,1]
  for k in range(len(decr)):
    i=h=decr[k]
    while i<len(v):
      x=v[i]
      j=i
      while j>=h and v[j-h]>x:
 v[j]=v[j-h]
 j=j-h
      v[j]=x
      i=i+1
  return v

def criaheap(v,n,i):
  maior=i
  l=2*i+1
  r=2*i+2
  if l<n and v[l]>v[maior]:
    maior=l
  if r<n and v[r]>v[maior]:
    maior=r
  if maior != i:
    v[i],v[maior]=v[maior],v[i]
    criaheap(v,n,maior)
  return v

def heapsort(v,n):
  i=(n//2)-1
  while i>=0:
    v=criaheap(v,n,i)
    i=i-1
```

```
  i=n-1
  while i>=0:
    v[0],v[i]=v[i],v[0]
    v=criaheap(v,i,0)
    i=i-1
  return v

def quick(v,i,f):
  ant=i
  dep=f
  mei=v[(i+f)//2]
  while 1==1:
    while v[ant]<mei:
      ant=ant+1
    while v[dep]>mei:
      dep=dep-1
    if ant<=dep:
      v[ant],v[dep]=v[dep],v[ant]
      ant=ant+1
      dep=dep-1
    if ant>dep:
      break
  if i<dep:
    quick(v,i,dep)
  if ant<f:
    quick(v,ant,f)
  return v


vv=[1,7,3,22,19,18,17,162,16,7,51,59,58,67,87]
print(bolha(vv))
print(bolhaest(vv))
print(inser(vv))
print(selec(vv))
print(shell(vv))
print(heapsort(vv,len(vv)))
print(quick(vv,0,len(vv)-1))
```

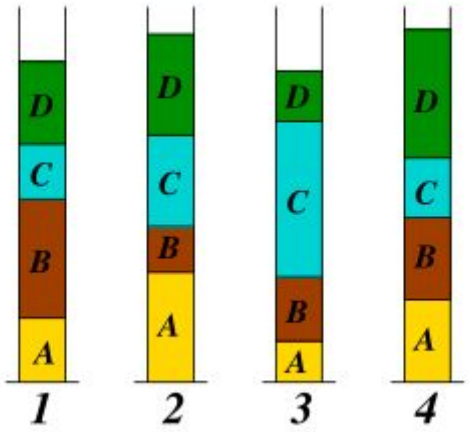
2.20 Sistemas lineares na prática

1. Materiais Particulados Suponha 4 tipos de materiais particulados distribuídos em 4 provetas e em cada proveta os materais estão dispostos em camadas, não misturadas, de modo que é possível medir o volume de cada material em cada uma das provetas. Chamando os materiais de A, B, C e D e suas densidades de ρ_A, ρ_B, ρ_C e ρ_D estas são as incógnitas que queremos achar. Os dados disponíveis são:

incógnita	significado
m_1	massa da proveta 1
m_2	massa da proveta 2
m_3	massa da proveta 3
m_4	massa da proveta 4
v_{im}	volume material m na proveta i

Veja-se uma visualização do que se fala

 ©75-21, Pedro Kantek



Deve-se lembrar que a massa de material A na proveta 1 é $v_{1A} \times \rho_A$, e assim por diante para todos os materiais. Assim, a massa total na proveta 1 é

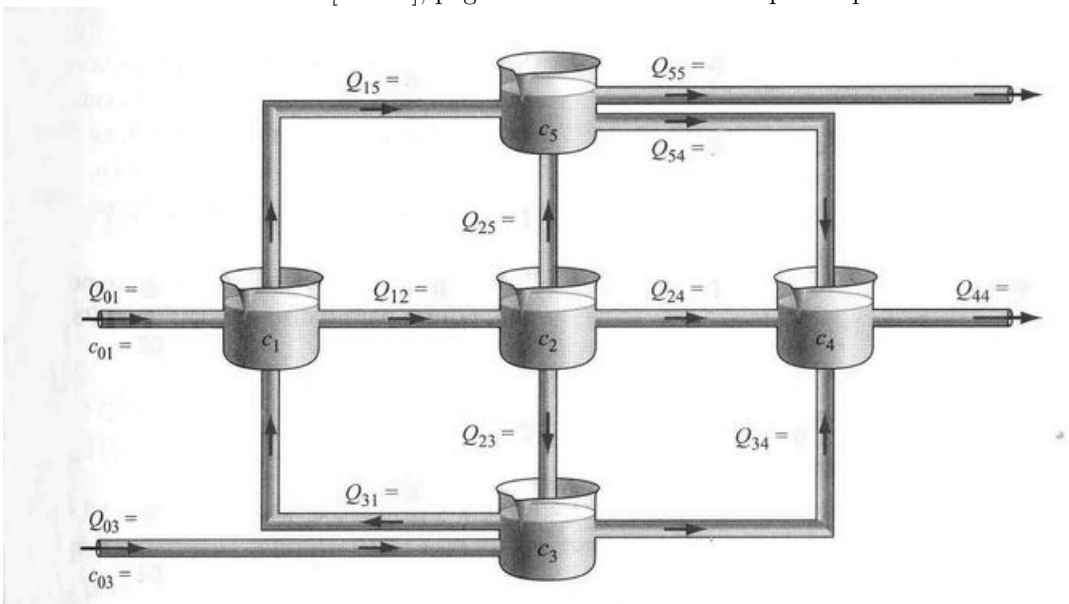
$$v_{1A}.\rho_A + v_{1B}.\rho_B + v_{1C}.\rho_C + v_{1D}.\rho_D = m_1$$

Considerando as 4 provetas tem-se 4 equações a 4 incógnitas.

2. Transportando máquinas Extraído de [Fra07], pág. 166. Uma transportadora possui 5 tipos de caminhões, que são equipados para transportar cinco tipos diferentes de máquinas A, B, C, D e E. Na tabela a seguir, o que cada tipo de caminhão pode levar a carga plena. Se a linha (1), significando caminhão do tipo 1, tiver A=1, B=1, C=1, D=0, E=2 significa que quando totalmente carregado ele pode levar essas quantidades de cada máquina sem sobrar espaço.

A pergunta é quantos caminhões de cada tipo deve-se usar para transportar exatamente as seguintes quantidades de cada tipo de máquina a carga plena.

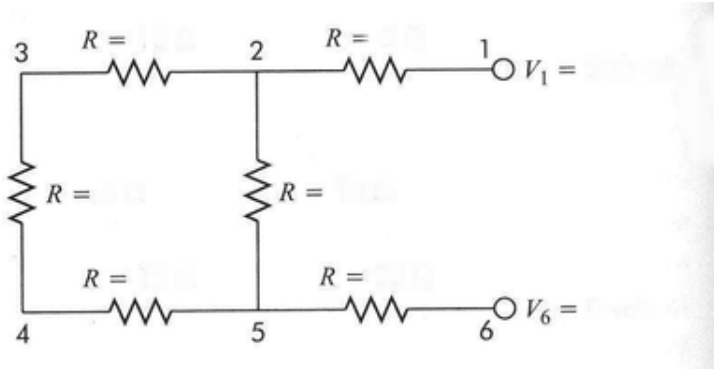
3. Reatores Extraído de [Cha13], pág. 227. Cinco reatores acoplados por tubos estão ilustrados na figura abaixo



Seja uma instância do problema na qual os valores são $Q_{15} = 5$, $Q_{55} = 4$, $Q_{54} = 2$, $Q_{25} = 1$, $Q_{01} = 6$, $Q_{12} = 4$, $Q_{24} = 1$, $Q_{44} = 9$, $Q_{23} = 2$, $Q_{34} = 6$, $Q_{31} = 3$, $Q_{03} = 7$, $C_{01} = 20$ e $C_{03} = 50$. A taxa de de fluxo de massa através de cada tubo é calculada como o produto de fluxo (Q) pela concentração (c). Em regime permanente ou seja, estacionário, o fluxo de massa para dentro e para fora de cada reator deve ser igual. Neste exemplo, os balanços de massa podem ser

escritos como
reator 1 $6 \times 20 + 3 \times r_3 = 5 \times r_1 + 4 \times r_1$
reator 2 $4 \times r_1 = r_2 + r_2 + 2 \times r_2 = 4 \times r_2$
reator 3 $2 \times r_2 + 7 \times 50 = 3 \times r_3 + 6 \times r_3$
reator 4 $r_2 + 2 \times r_5 + 6 \times r_3 = 9 \times r_4$
reator 5 $r_2 + 5 \times r_1 = 4 \times r_5 + 2 \times r_5$
Daqui:
Para o reator 1: $120 + 3.r_3 = 9.r_5$
Reator 2: $r_1 = r_2$
Reator 3: $2.r_2 + 350 = 9.r_3$
Reator 4: $r_2 + 2.r_5 + 6.r_3 = 9.r_4$
Reator 5: $r_2 + 5.r_1 = 6.r_5$
E daqui:
Reator 1: $9r_1 + 0r_2 - 3r_3 + 0r_4 + 0r_5 = 120$
Reator 2: $r_1 - r_2 + 0r_3 + 0r_4 + 0r_5 = 0$
Reator 3: $0r_1 + 2r_2 - 9r_3 + 0r_4 + 0r_5 = -350$
Reator 4: $0r_1 + 1r_2 + 6r_3 - 9r_4 + 2r_5 = 0$
Reator 5: $5r_1 + r_2 + 0r_3 + 0r_4 - 6r_5 = 0$
E resolvendo obtem-se $r_1 = 28$, $r_2 = 28$, $r_3 = 45.2$, $r_4 = 42.7$ e $r_5 = 42.6$ que é a resposta procurada. Simplificando e resolvendo, deve-se notar que o sistema é esparso (muitos zeros) o que eventualmente impede de usar o algoritmo ingênuo de Gauss. Se for o caso, use as ferramentas estudadas (APL, Maple, Freemat, pacote Numpy do Python ...)

4. Circuito Elétrico Extraído de [Cha13], pág. 224. A lei das tensões especifica que a soma algébrica das quedas de tensão (ou das diferenças de potencial) em qualquer caminho fechado deve ser igual a zero. Usando também a lei de Ohm ($V = ri$) obtém-se um sistema de equações algébricas lineares simultâneas porque os vários laços do circuito estão interconectados.



Seja uma instância do problema onde: $R_{12} = 5\Omega$, $R_{23} = 10\Omega$, $R_{34} = 5\Omega$, $R_{45} = 15\Omega$, $R_{25} = 10\Omega$ e $R_{56} = 20\Omega$ e $V_1 = 200V$ e $V_6 = 0V$.

Assumindo os sentidos positivos como $3 \to 2$, $4 \to 3$, $5 \to 4$, $6 \to 5$, $5 \to 2$ e $1 \to 2$ (se o resultado der negativo, basta inverter o sentido). Aplicando a lei de Kirchhoff tem-se
nodo 2: $i_{12} + i_{52} + i_{32} = 0$
nodo 5: $i_{65} - i_{52} - i_{54} = 0$
nodo 3: $i_{43} - i_{32} = 0$
nodo 4: $i_{54} - i_{43} = 0$
Aplicando a lei das tensões a cada um dos laços, tem-se
$-i_{54}.R_{54} - i_{43}.R_{43} - i_{32}.R_{32} + i_{52}.R_{52} = 0$
$-i_{65}.R_{65} - i_{52}.R_{52} + i_{12}.R_{12} - 200 = 0$
substituindo as resistências e trazendo as constantes para o lado direito, fica:
$-15.i_{54} - 5.i_{43} - 10.i_{32} + 10.i_{52} = 0$
$-20.i_{65} - 10.i_{52} + 5.i_{12} = 200$
Agora as 6 correntes desconhecidas podem ser representadas na forma matricial como

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 10 & -10 & 0 & -15 & -5 \\ 5 & -10 & 0 & -20 & 0 & 0 \end{bmatrix} \begin{Bmatrix} i_{12} \\ i_{52} \\ i_{32} \\ i_{65} \\ i_{54} \\ i_{43} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 200 \end{Bmatrix}$$

Resolvendo este sistema encontra-se a solução: 6.15, -4.6, -1.53, -6.15, -1.53 e -1.53.

 ©75-21, Pedro Kantek

5. Cascalho na construção civil Extraído de [Cha13], pág. 252. Um engenheiro civil envolvido em uma construção precisa de areia (A), cascalho fino (F) e cascalho grosso (G) para um projeto de uma obra. Existem 3 minas de onde esses materiais podem ser obtidos. A composição de cada mina é dada. Quantos metros cúbicos devem ser minerados de cada mina para as necessidades do engenheiro ?

Seja um exemplo: $A = 4800m^3$, $F = 5800m^3$ e $G = 5700m^3$, com as seguintes proporções de cada mina:

	Areia	C. fino	C. grosso
Mina1	52	30	18
Mina2	20	50	30
Mina3	25	20	55

Para este caso, as respostas são: $M_1 = 4005m^3$, $M_2 = 7131m^3$ e $M_3 = 5162m^3$.

Uma possíve solução em Python

```
# folha 744 de exemplo
def f744():
    import numpy as np
    print('---------------- Exercicio 1----------------')
    a=np.array([[2.8,4.5, 3.4,3.7],[5.2,10.6,5.4,3.9],
 [6.5,2.3,1.4,3.1],[1.8,8.2,5.6,1.5]],float)
    b=np.array([26.70,53.92,23.40,37.92],float)
    print(a)
    print(b)
    print(np.linalg.solve(a,b))
    print('---------------- Exercicio 2----------------')
    a=np.array([[6,4,0,7,0],[5,0,1,7,9],[7,0,7,9,9],[6,8,2,8,3],
  [7,9,5,1,8]],float)
    b=np.array([173,165,219,210,147],float)
    print(a)
    print(b)
    print(np.linalg.solve(a,b))
    print('---------------- Exercicio 5----------------')
    a=np.array([[0.35,0.34,0.33],[0.29,0.26,0.43],
  [0.36,0.40,0.24]],float)
    b=np.array([7519,7599,7066],float)
    print(a)
    print(b)
    print(np.linalg.solve(a,b))
f744()
```

2.21 Tomografia computadorizada

Para encerrar a discussão de sistemas lineares com uma aplicação robusta e real, vai-se trabalhar com a reconstituição de imagens que é feita pela máquina de TCA (tomografia computadorizada axial).

Começa-se a discussão pela presença muito comum de um sistema de m equações a n incógnitas estudando-se agora o que acontece com os diversos valores de m e n.

Quando $m = n$, tem-se o caso estudado até aqui nas aulas anteriores, já que se as m equações são linearmente independentes (nenhuma é múltipla de outra ou combinação linear de outras) o sistema tem uma única solução.

Se $m < n$ o sistema só admite solução se forem arbitrados valores **ad-hoc** para $m - n$ incógnitas.

O caso que vai ser estudado agora é $m > n$ que é o caso mais comum na realidade. Em uma rápida análise, comecemos com um sistema $m = n$ e acrescentemos ao sistema uma nova equação linearmente independente.

Para simplificar usando um caso real suponhamos 3 equações e 3 incógnitas: o sistema tem uma solução x_1, x_2 e x_3. Ao acrescentar a quarta equação o sistema deixa de ter solução exata. Mas, se pegarmos 3 dessas 4 equações (por exemplo as equações 1, 2 e 4) ele voltará a ter solução. Vamos chamá-la de x_1, x_2 e x_4. Se pegarmos as equações 1, 3 e 4, nova solução. Ao resolver o sistema com as 4 equações quer-se o ponto do espaço que minimiza a distância (mínimos quadrados) das soluções de 4 equações tomadas 3 a 3: a saber 1,2,3; 1,2,4; 1,3,4 e 2,3,4.

Esta estratégia faz todo o sentido, já que na vida real não existem soluções exatas, por diversas razões, sendo a principal a omnipresença do ruído, aqui entendido como erro (de leitura, de processamento, de transcriçao, do universo contra mim...).

Não vamos trabalhar com este algoritmo ele é bastante sofisticado, mas vão-se usar os pacotes à nossa disposição (APL, Freemat, Matlab, Python, ...). Neste último não se usa `numpy.linalg.solve` já que este exige matrizes quadradas, mas `numpy.linalg.lstsq`. Este exercício pede que você resolva um sistema com 170 equações a 64 incógnitas.

Tomografia Computadorizada Axial Trata-se de uma técnica que usa um computador e uma máquina de Raio X para reconstituir imagens de maneira transversal de um corpo. Originalmente na pesquisa do seu inventor, o programador inglês Godfrey Hounsfield, era apenas o cérebro humano já que este – por estar dentro de um capacete de matéria óssea: o crânio – é impermeável ao Raio-X exceto nos casos de rachaduras ou quebra. Mais tarde a técnica deu tão certo que acabou sendo adaptada a outras partes do corpo, mesmo de animais e também em alfândegas, presídios e similares para examinar o interior de qualquer coisa. A propósito até onde sei, o Hounsfield foi o único não médico a ganhar o Nobel de Medicina até hoje.

Feixes de raio X são disparados sobre o corpo em análise e ao invés de serem registrados em fotografias são entregues a um computador (a você...) para processamento. A máquina toda gira ao redor do corpo tirando muitos tomogramas do mesmo objeto. A variação angular é que vai permitir a reconstituição posterior.

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 cm^{-1}, e da água 0,19 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	35:55
Pulmão	-900:-400	Coágulo	80
Gordura	-110:-65	Músculo	40:60
gua	0	Fígado	50:85
Rim	30	Osso	130:250

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 voxeis 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. (Estas últimas informações foram retiradas em http://www.if.ufrgs.br/ast/med/imagens/imagens.htm por Kepler de Souza Oliveira Filho).

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.

A 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:

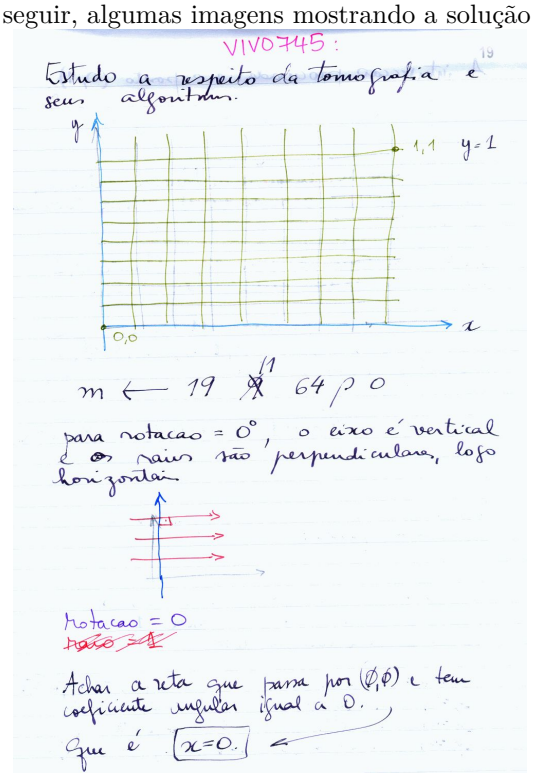
$$P(\theta, t) = \int_R f(x, y)\delta(x \sin\theta - y\cos\theta - t)dxdy$$

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. Em python

 ©75-21, Pedro Kantek

```
import numpy as np
def leitcoef():
    coef=np.zeros((680,16),float)
    ref=open("f:/n/k37/f745coef.myd","r")
    lin=ref.readline()
    i=0
    while (0!=len(lin)):
 lin=lin.split()
 for j in range(0,16):
     coef[i,j]=float(lin[j])
 i=i+1
 lin=ref.readline()
    coef.resize([170,64])
    return(coef)

def leiti():
    ti=np.zeros((170),float)
    ref=open("f:/n/k37/f745EXE1.myd","r")
    lin=ref.readline()
    i=0
    while (0!=len(lin)):
 ti[i]=float(lin)
 i=i+1
 lin=ref.readline()
    return(ti)
print(leitcoef())
a=leitcoef()
b=leiti()
c=np.linalg.lstsq(a,b)
ima=np.zeros((8,8),float)
k=0
xx=c[0]
for i in range(0,8):
    for j in range(0,8):
 ima[i,j]=abs(float("%8.3f" % xx[k]))
 k=k+1

print(ima)
```

A seguir, algumas imagens mostrando a solução matemática da tomografia computadorizada



A intersecção procurada é o ponto (0, 1).
└ entre a reta $x=0$ e $y=1$ é ↑

Agora deve-se achar a distância entre o ponto (0,0) e o ponto (0, 1). Essa distância é 1.

Divide-se esse 1 por 10, obtendo 0,1 ~~era~~ de separação entre cada raio.

Agora vamos procurar 11 retas que passam por (0,0), (0, 0.1), (0, 0.2), ... (0,0.9) e (0, 1) e são perpendiculares a $x=0$

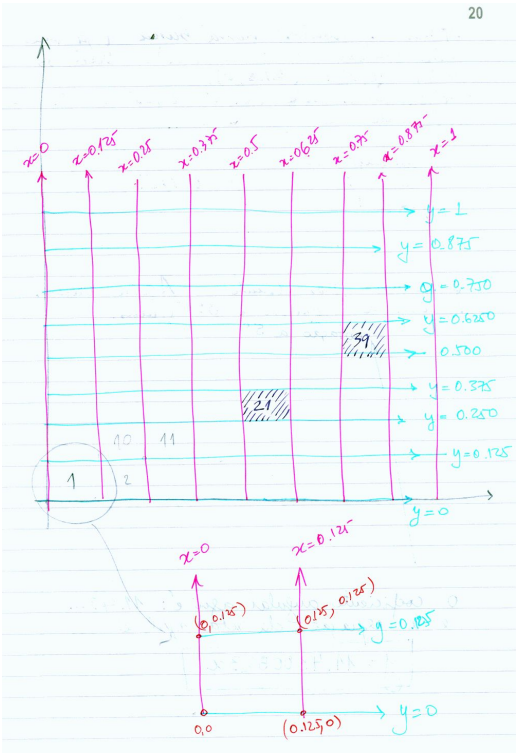
São elas: $y=0$ $y=0.1$, $y=0.2$, ... $y=0.9$ e $y=1$.

Agora vamos varrer os 64 pixels e para cada pixel:

analisar a reta $y=0$ com as 4 retas delimitadoras do pixel 1.
Como se pode ver no desenho ao lado

reta $(y=0)$ intersecção com a reta $(y=0)$ => infinitos
$(y=0)$ com $(y=0.125)$ => nenhum ponto
$(y=0)$ com a reta $(x=0)$ => só 1 ponto 0,0
$(y=0)$ com a reta $(x=0.125)$ só 1 ponto $(0.125, 0)$

©75-21, Pedro Kantek


20
x=0
x=0.125
x=0.25
x=0.375
x=0.5
x=0.625
x=0.75
x=0.875
x=1
y=1
y=0.875
y=0.750
y=0.6250
0.500
y=0.375
y=0.250
y=0.125
y=0
39
21
10
11
1
2
x=0
x=0.125
(0,0.125)
(0.125,0.125)
y=0.125
y=0
0,0
(0.125,0)

Achei 2 pontos nessa busca (já que neste contexto ∞ = nenhum). São os pontos (0,0) e (0.125, 0).
A distância entre eles é 0.125 e este é o coeficiente de

rotacao 0
raio 1
pixel 1

} 0.125.

e assim por diante. Agora vamos simular a rotação a 5° graus.

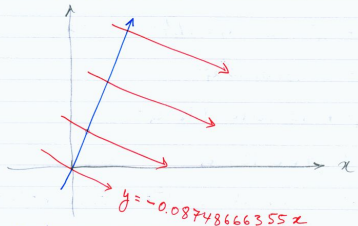
O coeficiente angular aqui é: 11.43...
e a equação da reta azul é

$y = 11.4300523x$

Agora vai-se procurar a interseccao da reta $y = 1$ com a reta $y = 11.4300523x$.
Acha-se o ponto (0.08748866353, 1)

A distância entre o ponto (0,0) e o ponto (0.08748..., 1) é de 1.003819838

Divide-se este valor por 10 obtendo-se 0.1003819838



Então, a questão agora é a reta que passa por 0,0 e ~~tem~~ é perpendicular a $y = 11.4300523x$ que é

$y = -0.08748866355x$

A segunda reta é a reta que passa pelo ponto que está na reta $y = 11.4300523x$ e está à distância $2 * 0.1003819838 = 0.2.00763967676$ do ponto (0.0) e é perpendicular a $y = 11.4300523x$.

 ©75-21, Pedro Kantek

Agora precisa-se achar o ponto sob a reta $y = 11.4300523x$ e esta à uma distância conhecida de 0,0.

então dada uma reta, um ponto e uma distância quer-se o outro ponto.

da formula pitagórica

$d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$ e sabendo-se que $x_1 = 0$ e $y_1 = 0$ fica

$d^2 = x_1^2 + y_1^2$ e $y = 11.4300523x$

$d^2 = x_1^2 + 11.4300523^2 x_1^2$

$d^2 = x_1^2 + 130.6460956\, x_1^2$

$d^2 = 131.6460956\, x_1^2$

$d = 0.20076396$ e $d^2 = 0.04030617069$

e $x_1^2 = 0.04030617069 / 131.6460956$

e $x_1 = 0.01749773276$ e $y_1 = 0.2000000$

A questão agora é a reta perpendicular a $y = 11.4300523x$ e que passa por $x_1 = 0.01749773276$ e $y_1 = 0.2$

A dito o coeficiente angular desa nova reta perpendicular é -0.08748866355

A reta procurada é

$y - y_1 = m(x - x_1)$ onde x_1 e y_1 são os pontos onde ela passa.

A reta procurada é

$y - 0.2 = -0.08748866355\,(x - 0.01749773276)$

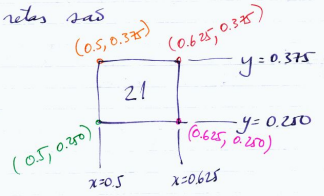
ou

$y = -0.08748866355x + 0.2015308533$

Agora deve-se examinar esta reta vis-à-vis as 4 retas delimitadoras do pixel em análise.

Considerando o pixel 21 (por hipótese)

Suas retas são



Vamos às intersecções:

1. da reta $x = 0.5$ com a reta $y = -0.08x + 0.201$ que dá o ponto: $(0.5, 0.161)$
2. da reta $x = 0.625$ com a reta $y = 0.08x + 0.201$ que dá o ponto: $(0.625, 0.151)$
3. da reta $y = 0.25$ com a reta $y = -0.08x + 0.201$ que dá:
   $y = -0.08x + 0.201$ e $y = 0.25$

   $0.25 = -0.08x + 0.201$

   $0.25 - 0.201 = -0.08x$

   $0.049 = -0.08x$

   $x = -0.6125$ é o ponto procurado

   é $(-0.6125, 0.25)$

4. da reta $y = 0.375$ com a reta $y = 0.08x + 0.201$ que dá $(-2.175, 0.375)$

graficamente este estado de coisas fica assim

O ponto é: as coordenadas dos pontos estão dentro (ainda que no limite) dentro do pixel?

Vejamos:

Ponto 1:

coord. x $\Rightarrow$ $0.5 \leq -2.17 \leq 0.625$ ✗ E

coord y $\Rightarrow$ $0.25 \leq 0.375 \leq 0.375$ ✓ ?

Resposta: não

Ponto 2: $0.5 \leq 0.625 \leq 0.625$ ✓ E

$0.25 \leq 0.151 \leq$ ~~0.625~~ 0.375 ✗ $\Rightarrow$ NÃO

Ponto 3: $0.5 \leq -0.6125 \leq 0.625$ ✗ E

$0.25 \leq 0.25 \leq 0.375$ ✓ $\Rightarrow$ NÃO

Ponto 4: $0.5 \leq -2.175 \leq 0.625$ ✗ E

$0.25 \leq 0.375 \leq 0.375$ ✓ $\Rightarrow$ NÃO

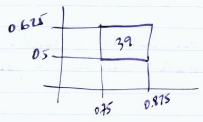
©75-21, Pedro Kantek

Como a quantidade de pontos incluídos no pixel é ϕ (zero), o coeficiente da rotação 5

raio 1

pixel 21 é zero.

Para terminar esta redação matemática, vamos pegar a rotação de 35°, raio 6 e pixel 39.



$35° \Rightarrow y = 1.428148007x$ e a reta perpendicular a esta é $y = -0.700x$

Como é o raio 6, tem-se:

a intersecção da reta $y = 1,428148007x$ e a reta $y = 1$

nos dá o ponto

$1 = 1.428148007x$

$x = 0.700207533$ e

portanto o ponto é $(0.709, 1)$

24

A distância entre $(0,0)$ e $(0.700, 1)$ é

$d^2 = (0.700)^2 + 1^2$

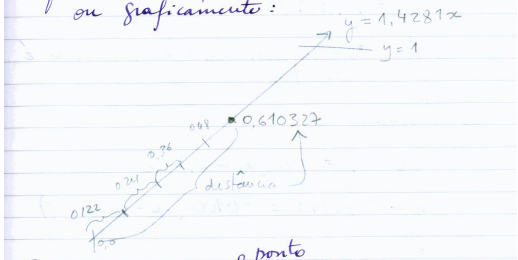
$= 0.49 + 1$

$d^2 = 1.49$

$d = \sqrt{1.49} = 1.220655562$

que dividido por 10 dá = 0.122065...

ou graficamente:



Precisa-se achar o ponto ~~a reta~~ que ~~passa~~ por esta em $y = 1.4281x$ e que está a 0.610327 da origem.

$d^2 = x_1^2 + 1.4281x^2$

$0.610327^2 = x_1^2 + (1.4281^2) \cdot x_1^2$

$0.372499 = x_1^2 + 2.0394\, x_1^2$

$0.372499 = 3.0394\, x_1^2$

$x_1^2 = \frac{0.372499}{3.0394} = 0.122554$

$x_1 = \sqrt{0.122554} = 0.35007$

retornando à equação da reta original acha-se o ponto que é

$(0.35007,\ 0.49993)$

Agora, a reta que passa por $(0.35007, 0.49993)$ e é perpendicular à reta

$y = 1.428148\,x$.

O coeficiente angular da reta procurada é -0.700.

a fórmula é

$y - y_1 = m(x - x_1)$ e fica:

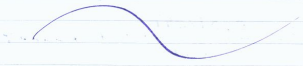
$y - 0.49993 = -0700\,(x - 0.35007)$

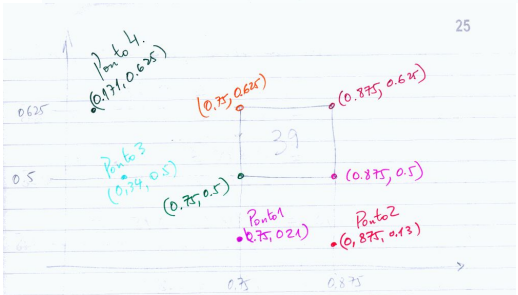
ou

$y - 0.49993 = -0.700x + 0.245049$

$y = -0.700x + 0.744979$ que é a reta procurada

agora vamos de novo à analise gráfica:





a reta é $y = -0.700x + 0.744979$

Análise dos 4 pontos de intersecção:

da reta $y = -0.700x + 0.7449$ com a reta $x = 0.75$ ⟶ dá o ponto $(0.75, 0.2199)$.

com a reta 0.875 ⟶ dá o ponto $(0.875, 0.1319)$

com a reta $y = 0.5$, tem

$0.5 = -0.700x + 0.7449$ ou

$0.5 - 0.7449 = -0.700x$

$-0.2449 = -0.700x$ e $x = \frac{0.2449}{0.700}$

$x = 0.3498$,

e o ponto é $(0.3498, 0.5)$

com a reta $y = 0.625$, tem

$0.625 = -0.700x + 0.7449$ ou

$-0.1199 = -0.700x$ e $x = \frac{-0.1199}{-0.7}$

$x = 0.171285$ e o ponto é $(0.171285, 0.625)$

 ©75-21, Pedro Kantek

Agora a análise final para os 4 pontos:

$P_1 - x$: $0.75 \leq 0.75 \leq 0.875$ E
$P_1 - y$: $0.5 \leq 0.21 \leq 0.625$ — NÃO

$P_2 - x$: $0.75 \leq 0.875 \leq 0.875$ E
$P_2 - y$ $0.5 \leq 0.13 \leq 0.625$ — NÃO

$P_3 - x$: $0.75 \leq 0.34 \leq 0.875$ E
$P_3 - y$ $0.5 \leq 0.5 \leq 0.625$ — NÃO

$P_4 - x$: $0.75 \leq 0.17 \leq 0.875$ E
$P_4 - y$: $0.5 \leq 0.625 \leq 0.625$ — NÃO

Logo: o coeficiente buscado para
rotação 35°
raio 6
e pixel 39 é Zero

Se, ao contrário 2 pontos tivessem caído dentro do pixel, haveria que:

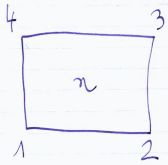
a) calcular a distância entre esses 2 pontos

b) o coeficiente buscado para uma rotação nesse raio e nesse pixel, seria o valor a) acima.

26

para preencher a matriz pontos
pontos ← 64 4 2 ρ ∅.

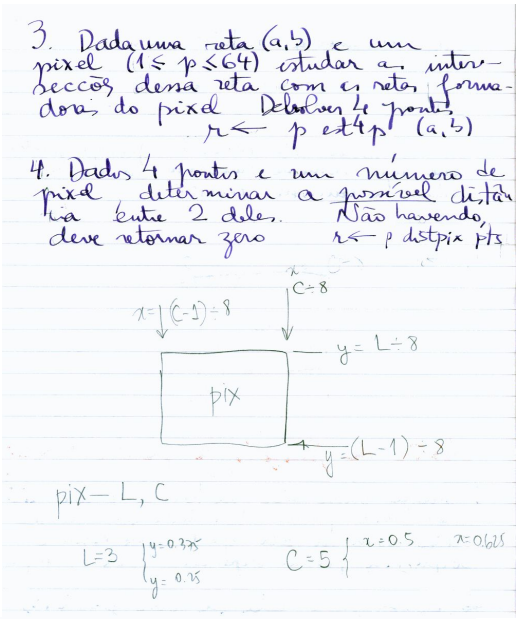
pontos [28]
criei a função APL chamada preenchepontos que aparentemente funcionou muito bem criando a variável PONTOS

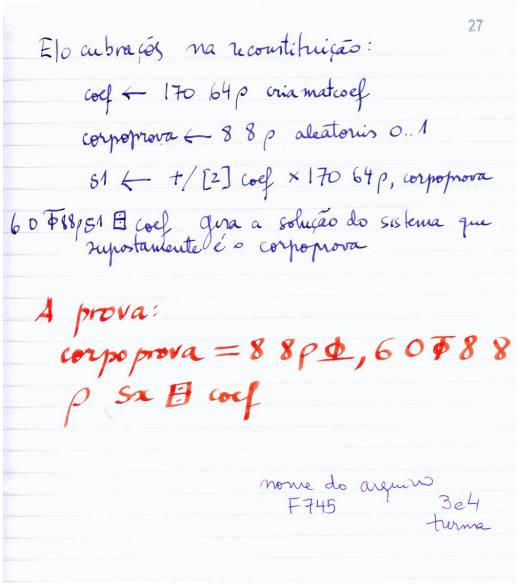


Lista de funções a desenvolver:

1. Dada uma reta na forma $y = ax$ (ié: a) devolver o ponto de intersecção dessa reta com a reta $y = 1$ que limita o corpo na posição superior r ← retapi a

2. Dada uma reta (a em $y = ax + b$) já que esta reta passa pela origem e uma distância devolver a reta (a, b) perpendicular à reta dada que passa pelo ponto especificado
r ← d reper a





Embora não seja objetivo deste texto, não resisto a incluir as soluções em APL, já que as soluções em Python (acima incluídas) estão muito prét-a-porter, ou seja estão prontinhas...

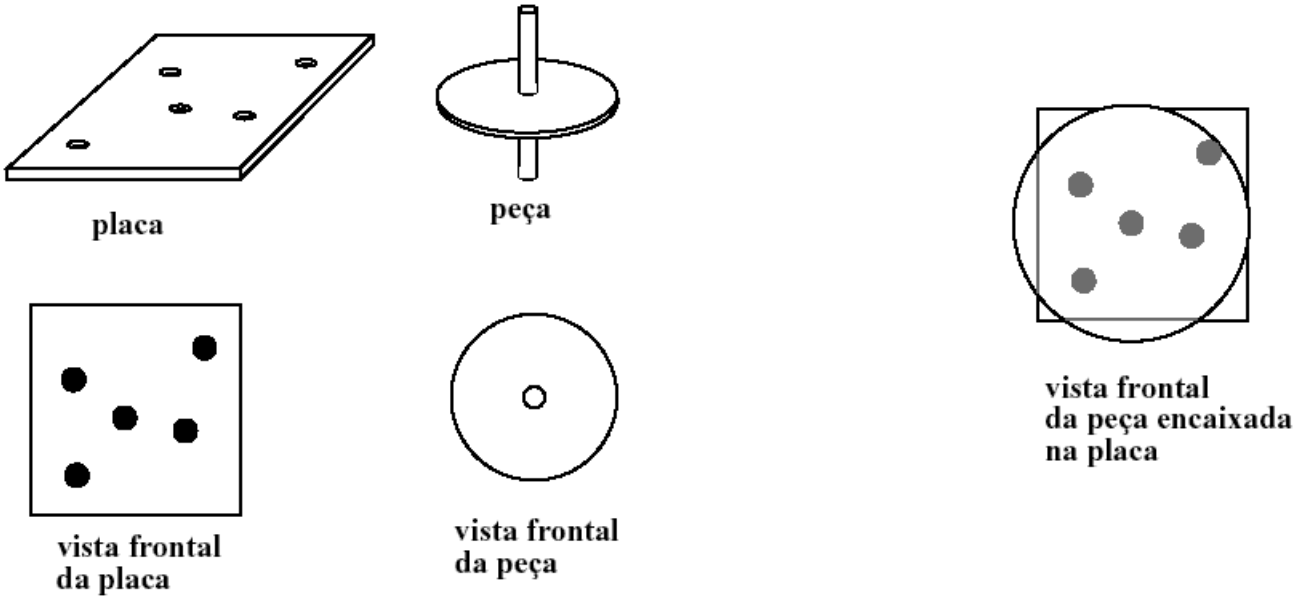
```
    ∇
 [0] r←criamatcoef;distancia;idi;ipi;irot;m;rotacao;rr;rr1;vetdis
 [1] ⍝ cria a matriz de coeficientes para o sistema linear da tomografia
 [2] ⍝ 17 x 9 x 64 = 19 rotacoes, 9 raios e 64 pixels na imagem.
 [3] m←17 10 64⍴0
 [4] irot←1
 [5] t1:→(irot>17)/fim
 [6]
 [7] ⍝------------------inicio do ciclo ----------------------
 [8] rotacao←3○(○90-irot×5)÷180
 [9] rr←retapi rotacao
[10] distancia←(rr,1)dp 0,0
[11] vetdis←distancia÷⍳10
[12] idi←1
[13] t2:→(idi>10)/oba
[14] rr1←vetdis[idi]reper rotacao
[15] ipi←1
[16] t3:→(ipi>64)/oga
[17] m[irot;idi;ipi]←ipi distpix ipi est4p rr1
[18] ipi←ipi+1
[19] →t3
[20] oga:idi←idi+1
```

 ©75-21, Pedro Kantek

```
[21] →t2
[22] oba:irot←irot+1
[23] →t1
[24] fim:r←m
     ∇

     ∇
 [0] r←retapi a
 [1] ⍝ devolve o ponto de interseccao da reta y=ax com a reta y=1
 [2] r←÷a
     ∇

     ∇
 [0] r←a dp b
 [1] ⍝ devolve a distancia pitagorica entre o ponto a e o ponto b
 [2] r←(((a[1]-b[1])*2)+((a[2]-b[2])*2))*0.5
     ∇

     ∇
 [0] r←d reper a;x;y
 [1] x←((d*2)÷(1+a*2))*0.5
 [2] y←x×a
 [3] r←x,y
     ∇
 [0] r←p est4p ab;lc;p1;p2;p3;p4;l;c
 [1] lc←achalc p
 [2] l←lc[1]
 [3] c←lc[2]
 [4] ⍝neste ponto reta superior→y=l÷8, inferior→y=(l-1)÷8, esq→x=(c-1)÷8 dir→x=c÷8
 [5] p1←((((l-1)÷8)-ab[2])÷ab[1]),(l-1)÷8  ⍝ mantem y
 [6] p2←(((l÷8)-ab[2])÷ab[1]),l÷8   ⍝ mantem y
 [7] p3←((c-1)÷8),(ab[1]×((c-1)÷8))+ab[2] ⍝ mantem x
 [8] p4←(c÷8),(ab[1]×c÷8)+ab[2]       ⍝mantem x
 [9] r←4 2⍴p1,p2,p3,p4
     ∇

     ∇
 [0] r←p distpix pts;l;c;lc;p1;p2;p3;p4;ap1;ap2;ap3;ap4;quais
 [1] ⍝ acha uma possivel distancia dentro do pixel (coeficiente do sistlin)
 [2] lc←achalc p
 [3] l←lc[1]
 [4] c←lc[2]
 [5] p1←pts[1;]
 [6] p2←pts[2;]
 [7] p3←pts[3;]
 [8] p4←pts[4;]
 [9] ⍝analise de p1:
[10] ap1←((p1[1]≥((c-1)÷8))^(p1[1]≤(c÷8)))^((p1[2]≥((l-1)÷8))^(p1[2]≤l÷8))
[11] ap2←((p2[1]≥((c-1)÷8))^(p2[1]≤(c÷8)))^((p2[2]≥((l-1)÷8))^(p2[2]≤l÷8))
[12] ap3←((p3[1]≥((c-1)÷8))^(p3[1]≤(c÷8)))^((p3[2]≥((l-1)÷8))^(p3[2]≤l÷8))
[13] ap4←((p4[1]≥((c-1)÷8))^(p4[1]≤(c÷8)))^((p4[2]≥((l-1)÷8))^(p4[2]≤l÷8))
[14] quais←(ap1,ap2,ap3,ap4)/⍳4
[15] →(2≠⍴quais)/zero
[16] r←pts[quais[1];]dp pts[quais[2];]
[17] →0
[18] zero:r←0
     ∇
```

2.22 Cubra os furos

Uma placa de aço retangular contém N furos circulares de 5 mm de diâmetro, localizados em pontos distintos, não sobrepostos – ou seja, o centro de cada furo está a uma distância maior ou igual a 5 mm do centro de todos os outros furos. Uma peça de forma circular, tendo em seu centro um eixo de 5 mm de diâmetro, deve ser colocada sobre a placa, de modo que o eixo encaixe-se em um de seus furos.



Tarefa Você deve escrever um programa para determinar o diâmetro mínimo que a peça deve ter de tal forma que, com seu eixo encaixado em um dos furos da placa, a parte circular cubra completamente todos os outros furos da placa.

Entrada A entrada é composta de vários conjuntos de teste. Cada linha de um conjunto de teste contém N pares de valores inteiros X e Y, separados por um espaço em branco, que descrevem a posição do centro de um furo (-10000 $\leq$ X $\leq$ 10000 e -10000 $\leq$ Y $\leq$ 10000). Os pares são separados por um ";". A unidade de medida das coordenadas dos furos é 1 mm. A seguir um exemplo

```
20 25 ; 10 5 ; 10 10  e  0 5 ; 10 0 ; 0 10
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir o diâmetro mínimo que a peça deve ter, como um número real com 2 casas decimais. A saída para o exemplo acima é 41.05 e 27.33

Em python:

```
def furos():
    import numpy as np
    x=np.array([[76,58],[20,33],[35,27],[24,28],[30,63]])
    d=np.zeros((5,5))
    t=len(x)
    mx=np.zeros((t))
    for i in range(t):
 for j in range(t):
     d[i][j]=2*(2.5+(((x[i][0]-x[j][0])**2)+((x[i][1]-x[j][1])**2))**0.5)
    print(d)
    for i in range(t):
 maximo=-9999
 for j in range(t):
     if d[i][j]>maximo:
  maximo=d[i][j]
 mx[i]=maximo
    mn=99999
    for i in range(t):
 if mx[i]<mn:
     mn=mx[i]
    print(mn)

furos()
```

 ©75-21, Pedro Kantek

micróbio Um micróbio 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.

Uma solução em Python:

```
def microbio(z,n,k):
    import random
    pop=z
    horas=0
    while pop<1000:
       horas=horas+n
       pop1=pop
       if pop1 < 1:
  print("zerou a populacao")
  return
#      print(pop1)
       for j in range(pop1):
  a=random.randint(0,1)
  if a==1:
     pop=pop-1
  else:
     pop=pop+k
   dias=horas/24
    return dias

def muitos():
    ct=0
    dd=0
    z=int(input("informe população inicial (z) "))
    n=int(input("informe a quantidade horas de ciclo (n) "))
    k=int(input("informe filhotes a cada divisão (k) "))
    while ct<1000:
       dd=dd+microbio(z,n,k)
       ct=ct+1
    print(dd/1000)

muitos()
```

Para gerar um (pseudo-) aleatório em Python, pode fazer:

```
from random import randint
a=randint(0,1) # a=0:cara, 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.

Troco 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)

```
def moedas(compra,nota):
  moedas=[0]*6
  compra=int(compra*100)
  nota=nota*100
  troco=nota-compra
  while troco>=100:
    moedas[0]=moedas[0]+1
    troco=troco-100
  while troco>=50:
    moedas[1]=moedas[1]+1
    troco=troco-50
  while troco>=25:
    moedas[2]=moedas[2]+1
    troco=troco-25
  while troco>=10:
    moedas[3]=moedas[3]+1
    troco=troco-10
  while troco>=5:
    moedas[4]=moedas[4]+1
    troco=troco-5
  moedas[5]=int(troco)
  print(sum(moedas))

# aqui o python deu uma engrossada: tente fazer 8.88,10 e
# deixe de fora o int(compra*100). Ele faz **rda.
```

egípcios 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

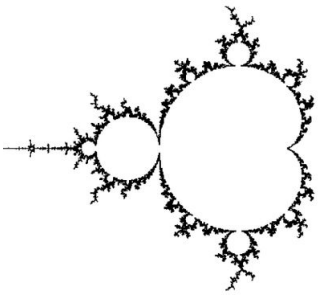
número	n	d
3.141592654 c/ 2 dígitos	22	7
3.141592654 c/ 3 dígitos	355	113
2.718281828 c/ 2 dígitos	87	32
2.718281828 c/ 3 dígitos	878	323
5.67891234 c/ 2 dígitos	91	16
5.67891234 c/ 3 dígitos	619	109

```
def egipcio(nr,lim): # lim deve ser 100 ou 1000
  minimo=+999999
  i=1
  while i<lim:
    j=1
    while j<lim:
      ca=i/j
      if (abs(nr-ca))<minimo:
 minimo=abs(nr-ca)
```

 ©75-21, Pedro Kantek

```
 mini=i
 minj=j
      j=j+1
    i=i+1
  return [mini,minj]

print (egipcio(3.14159265,100))
print (egipcio(3.14159265,1000))
```

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:
$(3 + 4i) + (2 + i) = 5 + 5i$
$i + (2 + i) = 2 + 2i$
$2 + 3 = 5$
$(3 + I) \times (2 - I) = 7 - I$

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 $z = z^2 + c$, 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 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 seqüência de iterações. Se ao final dessa seqüência, 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 iteracoes	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) Uma solução em Python

```
#mandelbroot
def mandelb(c):
  i=1
  zn=0+0j
  while i<=200:
    i=i+1
    zp=zn**2+c
    print(zp)
    hip=abs(zp)
    if hip>2:
      return i
    zn=zp
  return -1
print(mandelb(0.5861+0.5224j))
print(mandelb(0.4163+-.5544j))
```

OBI 2018: fase 1 - Álbum 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
$1 \leq N \leq 100$
$1 \leq M \leq 300$
$1 \leq X \leq N$

Exemplos

```
10 3 5 8 3
7

5 6 3 3 2 3 3 3
3

3 4 2 1 3 3
0
```

Solução em Python

```
# figurinhas
def figura(x):
  z=[0]*x[0] # tamanho do album
  i=1
  while i<=x[1]: #quantas figurinhas
    z[x[i+1]]=1
    i=i+1
  ct=0
  i=0
  while i<x[0]:
    if z[i]==0:
      ct=ct+1
    i=i+1
  return ct

print(figura([5,6,3,3,2,3,3,3]))
```

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.

		dia													
cápsula	ciclo	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 C_i, para $1 \leq i \leq N$, 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

$1 \leq N \leq 10^5$
$1 \leq F \leq 10^9$ e
$1 \leq C_i \leq 10^6$

Em todos os casos de teste, a resposta é sempre menor ou igual a 10^8 dias; Em todos os casos de teste, o número de moedas produzido, no total, após 10^8 dias, é sempre menor ou igual a 10^9.

Exemplos

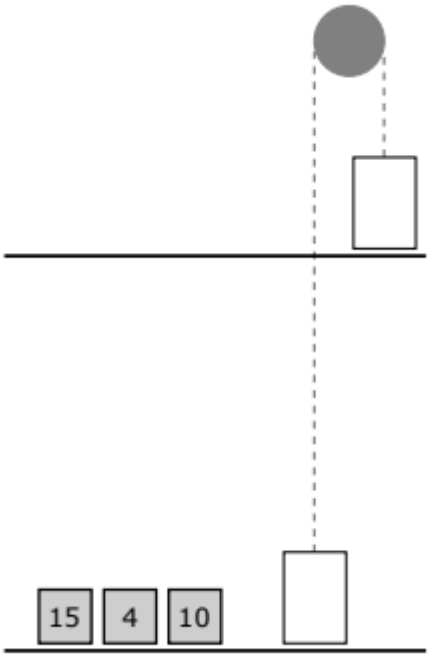
```
3 12 3 7 2
14
```

```
10 100 17 13 20 10 12 16 10 13 13 10
130
```

Solução em Python

```
# moedas
def moedas(x):
  n=x[0]
  f=x[1]
  i=1
  ct=0
  while 1==1:
    j=1
    while j<=n:
      if i%x[j+1]==0:
        ct=ct+1
        if ct>=f:
          return i
      j=j+1
    i=i+1
print(moedas([3,12,3,7,2]))
print(moedas([10,100,17,13,20,10,12,16,10,13,13,10]))
```

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, $P - Q \leq 8$, 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
$1 \leq N \leq 10^4$
O peso das caixas está entre 1 e 10^5 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
```

Solução em Python

```
# pesos
def pesos(x):
  i=x[0]-1
  while i>2:
    if x[i]-x[i-1]>8:
      return 'N'
    i=i-1
  if x[1]>8:
    return 'N'
  return 'S'
print(pesos([3,4,10,15]))
print(pesos([8,2,6,15,20,25,35,35,40]))
print(pesos([4,10,14,20,23]))
print(pesos([1,8]))
```

Enigma A máquina Enigma

```
# maquina enigma
a= "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
r1="CDZRMGHXAWQVUFSBPLTOEIYNKJ"
r2="VEWOASXDHRCYQJNPTKBLMFUZGI"
r3="IPRNFMWOZJQCXGTDBKELYVAHSU"
r4="DQNTSXZROJBAKCGEYPUIHFLMVW"
r5="BKQSLPMDCANYHGIRZVJFTUXEOW"
rf="OKYXVGFSTQBWNMAUJZHIPELDCR"
to=list(a)
l=[]
```

```
l.append(list(r1))
l.append(list(r2))
l.append(list(r3))
l.append(list(r4))
l.append(list(r5))

for i in range(1,7):
  t1=input("entre a tomada (informe as 2 letras) - %d " % i)
  t1=t1.upper()
  i1=to.index(t1[0])
  i2=to.index(t1[1])
  to[i1]=t1[1]
  to[i2]=t1[0]

for m in range(3):
  ra=int(input("qual o número do rotor ? "))
  ini=input("inicializado com ? ")
  ini=ini.upper()
  i1 = a.index(ini)
  ratx=[]
  for i in range(i1,i1+26):
    j = i%26
    ratx.append(l[ra-1][j])
  if m == 0:
    rat1=ratx
  elif m == 1:
    rat2=ratx
  elif m==2:
    rat3=ratx
print (rat1,rat2,rat3)
# ciclo de criptografia
print("agora vai-se iniciar o ciclo de criptografia...")
palavra=input("qual a palavra a criptografar ? ")
xuxa=[]
for zica in range(len(palavra)):
  l1 = palavra[zica]
  l1=l1.upper()
  indice1 = a.index(l1)
  letra1=to[indice1]
  indice2=a.index(letra1)
  letra2=rat1[indice2]
  indice3=a.index(letra2)
  print('entra ',letra2)
  letra3=rat2[indice3]
  print ('sai ',letra3)
  indice4=a.index(letra3)
  letra4=rat3[indice4]
  indice5=a.index(letra4)
  letra5=rf[indice5]
  indice6=rat3.index(letra5)
  letra6=a[indice6]
  indice7=rat2.index(letra6)
  letra7=a[indice7]
  indice8=rat1.index(letra7)
  letra8=a[indice8]
  indice9=to.index(letra8)
  letra9=a[indice9]
  xuxa.append(letra9)
#  print(["sequencia desta letra ",l1,letra1,letra2,letra3,letra4,letra5,letra6,letra7,letra8,
  nrat=[]
  for i in range(1,27):
    if zica ==26: #era i==26:
      nrat2=[]
```

 ©75-21, Pedro Kantek

```
      for k in range(1,27):  #so faz o rotate de rat3 e de rat2 -- para nao complicar demais
        z=k%26
        nrat2.append(rat2[z])
      rat2=nrat2
    j=i%26
    nrat.append(rat3[j])
  rat3=nrat
print("criptografia da palavra ",''.join(xuxa))
aa=input('tecle algo')
```

FITS Um programa que manuseia uma imagem guardada no padrão FITS

```
import numpy as np
import matplotlib.pyplot as plt
from astropy.utils.data import download_file
from astropy.io import fits
image_file = download_file('http://data.astropy.org/tutorials/FITS-images/HorseHead.fits', cac
hdu_list = fits.open(image_file)
print(hdu_list.info())
image_data = hdu_list[0].data
print(type(image_data))
print(image_data.shape)
hdu_list.close()
image_data = fits.getdata(image_file)
print(type(image_data))
print(image_data.shape)
plt.imshow(image_data, cmap='gray')
print('Min:', np.min(image_data))
print('Max:', np.max(image_data))
print('Mean:', np.mean(image_data))
print('Stdev:', np.std(image_data))
print(type(image_data.flat))
NBINS = 1000
histogram = plt.hist(image_data.flat, NBINS)
plt.show()
from matplotlib.colors import LogNorm
plt.imshow(image_data, cmap='gray', norm=LogNorm())
```

Criptoaritmética Imagine uma conta como

```
 SEND
+MORE
-----
MONEY
```

A brincadeira está em atribuir um dígito de 0..9 a cada letra de modo que a conta dê certo, sendo que letras iguais têm o mesmo valor e letras diferentes têm valor diferente. Uma possível resposta poderia ser

```
 9567
+1085
-----
10652
```

O programa que acha isto, (fazendo todas as combinações e trabalhando de maneira recursiva) é

```
def troca(depara,x): # depara="P1B6U9H2F7S5J8", x e a conta a fazer
  vet1=list(x[0])
  vet2=list(x[1])
  vetr=list(x[2])
  j=0
  while j<len(vet1):
    if vet1[j]!=' ':
      vet1[j]=depara[1+depara.index(vet1[j])]
```

```
    j=j+1
  j=0
  while j<len(vet2):
    if vet2[j]!=' ':
      vet2[j]=depara[1+depara.index(vet2[j])]
    j=j+1
  j=0
  while j<len(vetr):
    if vetr[j]!=' ':
      vetr[j]=depara[1+depara.index(vetr[j])]
    j=j+1
  return [vet1,vet2,vetr]

def menos(a,b): # a="1234567890" b="23489" devolve "15670" (o que falta)
  r=''
  for i in range(len(a)):
    if a[i] not in b:
      r=r+a[i]
  return r

def sol(ja,ain): # funcao recursiva que testa todas as possibilidades...
  global conta,qt
  qt=qt+1
#  if (qt%1000000==0):
  print("sendo chamado..." , ja,ain,str(qt))
  if len(ain)!=0:
    quais=menos('0123456789', ja)
    i=0 # era 1 deixei 0
    while i<len(quais):
      ja2=ja+ain[0]+quais[i]
      ain2=ain[1:len(ain)]
      sol(ja2,ain2)
      i=i+1
    return
  y=troca(ja,conta)
  t1=int(''.join(str(e) for e in y[0]))
  t2=int(''.join(str(e) for e in y[1]))
  tr=int(''.join(str(e) for e in y[2]))
  if t1+t2==tr:
    print("achamos uma resposta em ",y)
    aa=input("tecle algo")
  return

def criptarit(): #acerta conta (global) e o que já foi alocado (ou '') e manda ver
  global conta,qt
  qt=0
#  conta=['SEND','MORE','MONEY']  # agora deu...
#  sol('M1','OSENDRY')
#  conta=['BUHP','FUHS','PJSJU']
#  sol('P1','BUHFSJ')
# conta=['DRNRDHRMAD','NGUNGNNUHG','JDGHUMHUJH']
#  sol('','DRNHMAGUJ')
  conta=['SEND','MORE','MONEY'] #este aqui da...
  sol('M1S9O0','ENDRY')

criptarit()
```

Turtle Um pacote interessante de Python é o **turtle** que implementa a tartaruga de Wally Feurzig e Seymour Papert em 1966 através da linguagem Logo. É um ótimo ambiente para ensinar programação para crianças.

```
import turtle
t=turtle.Pen()
turtle.bgcolor("black")
```

 ©75-21, Pedro Kantek

```
colors = ["red", "yellow", "blue", "green"]
for x in range(300):
  t.pencolor(colors[x%4])
  t.forward(x)
  t.left(91)
```

euler001

```
# euler 1
soma=0
for i in range(1,1000):
  if i%3==0 or i%5==0:
    soma=soma+i
print(soma)
```

euler020

```
# euler 002
a=1
b=1
t=0
s=0
while (t<4000000):
  t=a+b
  b=a
  a=t
  print(a,b,t)
  if t%2==0:
    s=s+t
print(s)
```

euler003

```
# euler003
import math
def primo(x):
  if x%2==0 and x!=2:
    return False
  lim=math.ceil(x**0.5)
  for j in range (3,lim,2):
    if x%j==0:
       return False
  return True

s=600851475143
maior=-9999999

# s**0.5 = 775147
for i in range(1,775147):
  if s%i==0:
    c1 = s//i
    c2 = s//c1
    if primo(c1):
      if c1>maior:
        maior=c1
    if primo(c2):
      if c2>maior:
        maior=c2
print(maior)

# implementação alternativa mais demorada
# for i in range (1,600 851 475 143):
```

```
#   if s%i==0 and primo(i) and i>maior:
#     maior=i
# print(maior)
# esta implementaçao demorou 20 minutos para i chegar em 3 633 531 810...
```

euler003alt

```
import math
def primo(x):
  pri=True
  lim=math.ceil(x**0.5)
  for j in range (2,lim):
    if x%j==0:
      pri=False
  return pri

s=600851475143
maior=-9999999


# implementação alternativa mais demorada
for i in range (1,600851475143):
  if s%i==0 and primo(i) and i>maior:
    maior=i
print(maior)
```

euler004

```
#euler 004
def pali(x):
  a=list(x)
  b=list(reversed(a))
  return a==b
x=999
mai=-9999
z=999
while x>100:
  if pali(str(x*z)):
    if mai<x*z:
      mai=x*z
      mx=x
      mz=z
  z=z-1
  if z<100:
    x=x-1
    z=999
print(mai,mx,mz)
```

euler007

```
def prim2(n):
  import math
  if n==1:
    return False
  if n<4:
    return True
  if n%2==0:
    return False
  if n<9:
    return True
  if n%3==0:
    return False
  r=math.floor(math.sqrt(n))
```

 ©75-21, Pedro Kantek

```
  f=5
  while f<=r:
    if n%f==0:
      return False
    if (n%(f+2))==0:
      return False
    f=f+6
  return True

def enesprim(qual):
  count=1
  candidato=1
  while count<qual:
    candidato=candidato+2
    if prim2(candidato):
      count=count+1
  print(candidato)
```

euler008

```
def euler008():
  s='73167176531330624919225119674426574742355349194934'
  s=s+'96983520312774506326239578318016984801869478851843'
  s=s+'85861560789112949495459501737958331952853208805511'
  s=s+'12540698747158523863050715693290963295227443043557'
  s=s+'66896648950445244523161731856403098711121722383113'
  s=s+'62229893423380308135336276614282806444486645238749'
  s=s+'30358907296290491560440772390713810515859307960866'
  s=s+'70172427121883998797908792274921901699720888093776'
  s=s+'65727333001053367881220235421809751254540594752243'
  s=s+'52584907711670556013604839586446706324415722155397'
  s=s+'53697817977846174064955149290862569321978468622482'
  s=s+'83972241375657056057490261407972968652414535100474'
  s=s+'82166370484403199890008895243450658541227588666881'
  s=s+'16427171479924442928230863465674813919123162824586'
  s=s+'17866458359124566529476545682848912883142607690042'
  s=s+'24219022671055626321111109370544217506941658960408'
  s=s+'07198403850962455444362981230987879927244284909188'
  s=s+'84580156166097919133875499200524063689912560717606'
  s=s+'05886116467109405077541002256983155200055935729725'
  s=s+'71636269561882670428252483600823257530420752963450'
  maior=-9999999
  i=0
  while i<987:
      prod=1
      j=0
      while j<13:
          prod=prod*int(s[i+j])
          j=j+1
      if prod>maior:
          maior=prod
          maii=i
          maij=j
      i=i+1
  print (maior,maii,maij)



  print(len(s))
```

Euler 13

```
# euler13
```

```
def eul13():
  t=0
  with open("f:/p/up2017/ofidiario/arq_p13.txt") as f:
    for line in f:
      a = int(line)
      t=t+a
  print(t)


eul13()
```

Euler 14

```
# eul14
def eul14():
  sem=1
  mai=0
  while sem < 1000000:
    temp=sem
    ctt=0
    while temp != 1:
      if temp%2==0:
        temp=temp//2
      else:
        temp=(3*temp)+1
      ctt=ctt+1
    if ctt > mai:
      mai = ctt
      qual = sem
    sem=sem+1
  return qual

print(eul14())
```

Euler 15

```
# eul15
def eul15():
  import numpy
  a=numpy.zeros((21,21))
  for i in range(21):
    a[i][0]=1
    a[0][i]=1
  i=1
  while i < 21:
    j=1
    while j<21:
      a[i][j]=a[i-1][j]+a[i][j-1]
      j=j+1
    i=i+1
  return a[20][20]

print(eul15())
```

Uma rede neural - 845

```
import numpy as np

def rn845(c1,c2,en,sa,tn,fa):
  # c1=neuronios da camada 1
  # c2=neuronios camada 2
```

 ©75-21, Pedro Kantek

```
# en=entradas reais
# sa=saidas esperadas
# tn=tipos de neuronio (1=linear, 2=sigmoidal)
# fa=fator de aprendizado

# neuronio 1
print("neuronios camada 1",a)
print("neuronios camada 2",b)
n1=(c1[0][0]*1)+(c1[0][1]*en[0])+(c1[0][2]*en[1])
if tn[0]==2:
  n1=np.tanh(n1)
print("neuronio 1 = ",n1)
# neuronio 2
n2=(c1[1][0]*1)+(c1[1][1]*en[0])+(c1[1][2]*en[1])
if tn[1]==2:
  n2=np.tanh(n2)
print(n2)
# neuronio 3
n3=(c1[2][0]*1)+(c1[2][1]*en[0])+(c1[2][2]*en[1])
if tn[2]==2:
  n3=np.tanh(n3)
print(n3)
# neuronio 4
n4=(c2[0][0]*1)+(c2[0][1]*n1)+(c2[0][2]*n2)+(c2[0][3]*n3)
if tn[3]==2:
  n4=np.tanh(n4)
print(n4)
# neuronio 4
n5=(c2[1][0]*1)+(c2[1][1]*n1)+(c2[1][2]*n2)+(c2[1][3]*n3)
if tn[4]==2:
  n5=np.tanh(n5)
print(n5)
# note que n4=saida1 e n5=saida2
# retropropagação no n4
r4=sa[0]-n4
if tn[3]==2:
  r4=r4*1-(r4**2)
print("erro em n4 = ",r4)
# retropropagacao de n5
r5=sa[1]-n5
if tn[4]==2:
  r5=r5*(1-n5**2)
print("erro em n5 = ",r5)
# retropropagação de n3  R3←(R4×c2[1;4])+(R5×c2[2;4])
r3=(r4*c2[0][3])+(r5*c2[1][3])
if tn[2]==2:
  r3=r3*(1-n3**2)
print("erro em n3 = ",r3)
# retropropagacao em n2  R2←(R4×c2[1;3])+(R5×c2[2;3])
r2=(r4*c2[0][2])+(r5*c2[1][2])
if tn[1]==2:
  r2=r2*(1-n2**2)
print("erro em n2 = ",r2)
# retropropagacao de n1   R1←(R4×c2[1;2])+(R5×c2[2;2])
r1=(r4*c2[0][1])+(r5*c2[1][1])
if tn[0]==2:
  r1=r1*(1-n1**2)
print("erro em n1 = ",r1)
# calculo dos novos pesos
w11=c1[0][1]+(-2)*fa*n1*r1
w12=c1[1][1]+(-2)*fa*n2*r2
w13=c1[2][1]+(-2)*fa*n3*r3
print ("peso w11 = ",w11)
```

```
  print ("peso w12 = ",w12)
  print ("peso w13 = ",w13)
#-------------- execucao da funcao --------------
a=np.array([[-0.1, 0.2, 0.2],[0.3,-0.1,0.3],[0.1, 0.1, 0.9]])
b=np.array([[0.2,0.1,-0.1,-0.1],[-0.1,0.5,0.2,1.1]])
c=[0.1,0.7]
d=[0.2,1]
e=[2,2,2,1,2]
f=0.1

rn845(a,b,c,d,e,f)
```

Dump

```
#dump
def visivel(b):
  if b!='  ':
    a=int(b,16)
  else:
    return' '
  if a>32 and a<127:
    return chr(a)
  else:
    return '.'

def dump(nome):
  import math
  f=open(nome,"rb")
  x=f.read()
  y=x.hex().upper()
  tx=[]
  i=0
  tam1=len(y)
  tam2=32*(math.ceil(tam1/32))
  y=y+'                                '
  y=y+'                                '
  lin=0
  while i<tam2:
    tx.append(y[i:i+32])
    i=i+32
    lin=lin+1
  print("Dump do arquivo "+nome+"          -   dump v1.1, p.kantek, jul/17")
  print("")
  print("          0  1  2  3  4  5  6  7    8  9  A  B  C  D  E  F   ascii")
  print("         -- -- -- -- -- -- -- --   -- -- -- -- -- -- -- --   ----------")

  i=0
  while i<lin:
    l1=visivel(tx[i][0:2])
    l2=visivel(tx[i][2:4])
    l3=visivel(tx[i][4:6])
    l4=visivel(tx[i][6:8])
    l5=visivel(tx[i][8:10])
    l6=visivel(tx[i][10:12])
    l7=visivel(tx[i][12:14])
    l8=visivel(tx[i][14:16])
    lp=l1+l2+l3+l4+l5+l6+l7+l8
    l9=visivel(tx[i][16:18])
    la=visivel(tx[i][18:20])
    lb=visivel(tx[i][20:22])
    lc=visivel(tx[i][22:24])
    ld=visivel(tx[i][24:26])
```

 ©75-21, Pedro Kantek

```
    le=visivel(tx[i][26:28])
    lf=visivel(tx[i][28:30])
    lg=visivel(tx[i][30:32])
    ls=l9+la+lb+lc+ld+le+lf+lg
    pp=tx[i][0:2],tx[i][2:4],tx[i][4:6],tx[i][6:8],tx[i][8:10],tx[i][10:12],tx[i][12:14],tx[i]
    sp=tx[i][16:18],tx[i][18:20],tx[i][20:22],tx[i][22:24],tx[i][24:26],tx[i][26:28],tx[i][28:
    ii=hex(i*16)[2:].rjust(6,'0').upper()
    print(ii,' ',' '.join(pp),' ',' '.join(sp),'*',''.join(lp),''.join(ls))
    i=i+1



#dump("c:/apl2/f458exem.myd")
dump("c:/apl2/f458N001.myd")
```

k-vizinhos

```
# f668
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.myd",499)  #499 é o filme 500
```

QRCODE

```
import qrcode as qq
import matplotlib.pyplot as plt
qr = qq.QRCode(
    version=1,   # tamanho do QRCODE 10=menor, 40=maior
    error_correction=qq.constants.ERROR_CORRECT_L,  #correçao de erro
                                            #   l=7%, M=15% Q=25% e H=30%
    box_size=20,  # tamanho do pixel
    border=8, #tamanho da borda
)
qr.add_data('Oi nois aqui, traveis')  #conteúdo do QRCODE
qr.make(fit=True)

img = qr.make_image()
plt.imshow(img)
plt.show()
```

uva100 - problema $3n+1$

```
import sys
cache = {1: 1}  # {number: cycle_length, ...}
def get_cycle_length(num):
    cycle_length = cache.get(num)
    if cycle_length is None:
        cycle_length = get_cycle_length(num // 2 if num % 2 == 0 else 3 * num + 1) + 1
        cache[num] = cycle_length
    return cycle_length
def main():
    for line in sys.stdin:
        min_num, max_num = map(int, line.split())
        a,b=min_num,max_num
        if max_num<min_num:
          max_num,min_num=min_num,max_num
        max_cycle_length = max(map(get_cycle_length, range(min_num, max_num+1)))
        print(a,b, max_cycle_length)
    sys.exit(0)
main()
```

Algumas dicas sobre este código:

```
import sys # importa a leitura em stdin
cache = {1: 1}  # define dicionário para conter o tamanho do ciclo
def get_cycle_length(num): #define a função que retorna o tamanho do ciclo
    cycle_length = cache.get(num) pega o valor do dicionario correspondente a num
    if cycle_length is None: # não existe ainda
        cycle_length = get_cycle_length(num // 2 if num % 2 == 0 else 3 * num + 1) + 1
        # recursivamente, pega o ciclo de n//2 ou n*3+1 conforme o caso e soma 1
        cache[num] = cycle_length  # inclui este novo valor recém calculado
    return cycle_length # devolve o valor recém calculado
def main():  #define a função main()
    for line in sys.stdin: # para cada linha na entrada
        min_num, max_num = map(int, line.split()) # converte 2 inteiros
        # lembrando, map aplica a função (operando1) à cada elemento da lista operando2
        a,b=min_num,max_num  # guarda a ordem original, para futura impressão
        if max_num<min_num: # se precisar, inverte
          max_num,min_num=min_num,max_num
        max_cycle_length = max(map(get_cycle_length, range(min_num, max_num+1)))
        # maximo é o max de map(tamanho, nos elementos de range min..max+1
        print(a,b, max_cycle_length) #imprime o resultado
    sys.exit(0) # importante no UVA
main()  # chama a função main
```

Uma dica conceitual: Um iterador está para um conainer como um índice está para uma lista, com o detalhe que o iterador também pode ser usado em uma lista. Um iterador é o que é um cursor em um banco de dados.

Algumas dicas

1. Definir um dicionário dic = chave: valor

   ```
   Exemplo: oi={1: 1}
   ```

2. Consultar um dicionário aa = dic[chave]

3. Criar uma nova chave dic[novachave] = novovalor

4. Substituir um valor velho dic[velhachave] = novovalor

5. Saber se uma chave existe:

   ```
   aa = dic.get(chave)  % note o parenteses
   if aa is None  % então chave não existe
   ```

 ©75-21, Pedro Kantek

6. Atribuição condicional:

```
aa = aa//2 if aa%2==0 else 3*aa+1 equivale a
if aa%2 == 0:
  aa = aa//2
else:
  aa = 3*aa + 1
```

Capítulo 3

Bibliografia

3.1 Python

para saber mais sobre Python

BEA13 BEAZLEY, David e JONES, Brian K. **Python Cookbook**. São Paulo, Novatec, 2013. 713p.

RAM15 RAMALHO, Luciano. **Python Fluente**. São Paulo, Novatec, 2015. 797p.

MEN10 MENEZES, Nilo Ney Coutinho. **Introdução à Programação com Python**. São Paulo, Novatec, 2010. 327p.

MCK18 MCKINNEY, Wes. **Python para Análise de Dados**. São Paulo, Novatec, 2018. 615p.

MUE16 MUELLER, John Paul, **Começando a programar em Python para leigos** Rio de Janeiro, Alta Books, 2016. 379p.

KIN15 KINSLEY, Harrison e McGUGAN, Will. **Introdução ao desenvolvimento de Jogos em Python com PyGame**. São Paulo, Novatec, 2015.

PYTxx Comunidade Python Mundial. Site `https://www.python.org`.

PYTxx Comunidade Python Brasil. Site `https://python.org.br`.

KAN18 KANTEK, Pedro Luis Garcia Navarro. **Python**. Curitiba, UFPR, 2021. Disponível em `www.algoritmovivo/ativi`

DOW16 DOWNEY, Allen B. **Pense em Python**. São Paulo, Novatec, 2016.

FERsd FERG, Steven. **Pensando em Tkinter**. São Paulo, se, sd. Disponível em `http://thinkingtkinter.sourceforg`

3.2 Algoritmos e programação

Livros e materiais sobre programação.

DEL12 DELGADO, Armando Luiz. **Linguagem C++**. Curitiba, UFPR, 2012.

DEW89 DEWDNEY, A. K. **The (new) Turing Omnibus**. New York, W. H. Freeman, 1989. 460p.

COR14 CORMEN, Thomas H. **Algoritmos**. Rio de Janeiro, Elsevier, 2014. 188p.

BHA17 BHARGAVA, Aditya. **Entendendo Algoritmos**. São Paulo, Novatec, 2017. 259p.

SKI97 SKIENA, Steven S. - **The Algorithm Design Manual**. New York, Springer. 1997. 486p.

SKI03 SKIENA, Steven S e REVILLA, Miguel A. **Programming Challenges**. New York, Springer. 2003. 361p.

SKI12 SKIENA, Steven S e REVILLA, Miguel A. **Desafios de Programación**. New York, Springer. 2012. 343p. Tradução do inglês ao espanhol.

FOR05 FORBELLONE, A. L. V. e EBERSPÄCHER, H. F. - **Lógica de Programação**. São Paulo. 2005. 214p.

FOR72 FORSYTHE, Alexandra I. et all. **Ciência de Computadores** - 1. curso. Volume 1. Rio de Janeiro, Ao Livro Técnico. 1972. 234p.

FOR72b FORSYTHE, Alexandra I. et all. **Ciência de Computadores** - 1. curso. Volume 2. Rio de Janeiro, Ao Livro Técnico. 1972. 560p.

KNU71 KNUTH, Donald E. **The Art of Computer Programming**. Volumes 1 e 2. Reading, Massachussets, Addison Wesley Publishing Company, 1971, 624p.

WIR89 WIRTH, Niklaus. **Algoritmos e estruturas de dados**. Rio de Janeiro, Prentice-Hall do Brasil, 1989, 245p.

KAO08 KAO, Ming-Yang. (ed.) **Encyclopedia of Algorithms**. New York, Springer, 2008. 1200p.

RUS03 RUSSEL, Stuart e NORVIG, Peter. Inteligencia Artificial. Editora Campus, 2003.

3.3 Métodos Numéricos

SAN07 SANCHES, Ionildo José e FURLAN, Diogenes. **Métodos Numéricos**. Curitiba, UFPR, 2007.

FRA07 FRANCO, Neide Bertoldi. **Cálculo Numérico**. São Paulo, Pearson, 2007.

JUS18 JUSTO, Dagoberto Adriano et alli. **Cálculo Numérico, versão Python**. Porto Alegre, UFRGS, 2018.

BAR87 BARROSO, Leônidas Conceição et alli. **Cálculo Numérico (com aplicações)**. São Paulo, Harbra, 1987.

RUG88 RUGIERO, Márcia e LOPES, Vera Lúcia. **Cálculo Numérico - aspectos teóricos e computacionais**. São Paulo, Pearson, 1988.

PAC78 PACITTI, Tércio e ATKINSON, Cyril. **Programação e Métodos Computacionais**. vol 1 e 2. Rio de Janeiro, LTC, 1979.

CAM14 CAMPOS Filho, Frederico Ferreira. **Algoritmos Numéricos**. Rio de Janeiro, LTC, 2014.

Índice Remissivo