CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75565 ARIEL GUSTAVO RODRIGUES DA S
24JOG307 - 1 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

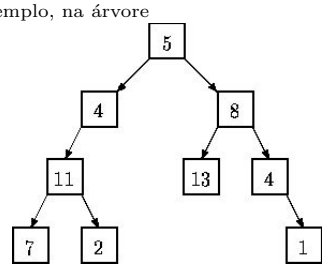
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 42 (12(3(1()())(8()(10(9()())())))(13()(17(16()())(20()()))))
2.
 38 (5(2()())(10(8(7()())(9()()))(18(12()(17(13()())()))())))
3.
 56 (3(2()())(5()(15(11(10(9()())())(12()()))(18()(20()()))))))
4.
 52 (11(1()(3()(6(4()())))))(12()(16(13()())(17()(19()()))))))
5.
 66 (19(13(2()(5()(12(9(6()())(10()()))))))(15()()))(20()()))
```

Responda aqui:

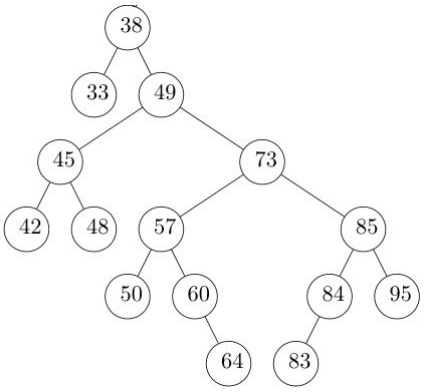
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14)())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(11(10(5(4)(7))())(21(15(13(12)())())(28()(30(29)()))))
(26(2()(25(11(5(3()(4))(6))(21))()))(28()(30)))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<[<(8+4)-6>/{{1-1}/(9*6)}]+({{6-7}+[8-6]}/{(7-7)/[{8-4}]})>
[{{[3{/8]-{8-8}}/<{4+6}/{4/1}>}*9]
<{<8-<9-4>>-({1*4}+2)}+[{{3/4}*{9+4}}/(6+(2-9))]>
[({9+{8*3}}/({2*9}+3))/[<[2*4]/{3-3}>/<<1/7>*9>]]
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 1. Calcule:
$(((p \wedge q) \leftrightarrow (p \wedge p)) \rightarrow ((\sim q) \rightarrow q))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$((((\sim p) \wedge q) \wedge (q \wedge p)) \rightarrow (q \wedge (\sim p)))$

13. Suponha que p vale 0 e q vale 1. Calcule:
$((((p \rightarrow q) \rightarrow (p \wedge q)) \wedge (q \leftrightarrow p)) \wedge (p \leftrightarrow (\sim q)))$

14. Suponha que p vale 0 e q vale 0. Calcule:
$((((p \leftrightarrow q) \vee (q \wedge p)) \wedge (p \wedge (\sim q))) \wedge (p \leftrightarrow (\sim q)))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* (* 4 (* (* 8 2) (+ 5 1))) 1)
(* 6 (- (+ (- 6 4) (- 7 1)) (- (+ 7 1) (+ 5 3))))
(+ (+ (+ 6 (+ 6 2)) 4) 6)
(- (- (* (* 8 4) 5) (+ (* 6 4) 1)) 6)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75572 CASSIANO MAGNO CHAGAS E SA
24JOG307 - 2 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

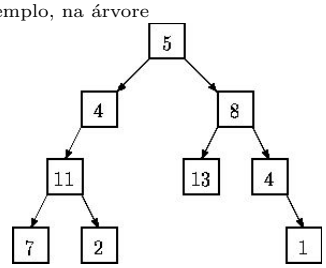
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 47 (3(2()())(6()(10(9(8()())())(19(17(12()(14()()))())()))))
2.
 47 (1()(9(6(4(3()())())())(18(13(10()(12()()))())(19()()))))
3.
 31 (20(13(1()(6(3()())(9()(10()()))))(19(14()(18()()))())))()
4.
 33 (4(3()())(15(5()(10(6()(8()(9()()))))))(20(16()())()))
5.
 42 (18(6(2()(5()()))(9()(15(12()(14()()))(17()()))))(20()()))
```

Responda aqui:

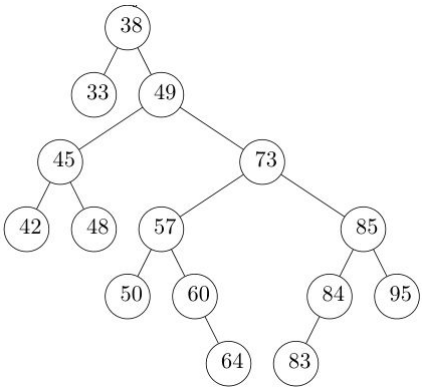
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(22(5(3)(11(7()(9))(14()(18(17)()))))(25()(27()(28()(29)))))
(26(7(3(1)())(25(16(8()(13(11(10)())()))())()))(28()(30)))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<[(3*<3*8>)+{[9/1]-{8-1}}]*(<3*<7+2>>*<<4*5>/1>)>
[([{5-4}+<1+6>]/{<5+4>-5})-<{<7*8>*{2-1}}+[6*<9-{9>]>]
<((<7-3>*4)-<<9/8>-8>)-[<{6/2}-4>-{8+]7+6]}]>
(({(2*7)*{5-8}}-<7/7>)*((6-[3+1])-((1*9)/3)})
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 0. Calcule:
$(((q \wedge p) \vee ((\sim q) \wedge p)) \vee ((\sim q) \wedge p))$

12. Suponha que p vale 1 e q vale 1. Calcule:
$((((\sim q) \vee p) \leftrightarrow (p \leftrightarrow q)) \rightarrow (p \wedge q))$

13. Suponha que p vale 1 e q vale 0. Calcule:
$(((((\sim p) \leftrightarrow q) \leftrightarrow (q \wedge p)) \vee (p \vee q)) \leftrightarrow (p \wedge q))$

14. Suponha que p vale 1 e q vale 1. Calcule:
$(((((\sim q) \wedge p) \vee (q \vee p)) \wedge (q \rightarrow p)) \vee (p \wedge p))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ (* (+ (+ 7 2) (- 5 2)) (* (- 5 2) 4)) 3)
(+ 1 (+ 6 (- (* 7 3) 6)))
(- (* (- (* 7 1) 3) (+ 5 (- 8 1))) 3)
(* (+ 2 (- (+ 7 3) 3)) 6)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75589 CHRISTOPHER BUCH FILIPAK
24JOG307 - 3 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

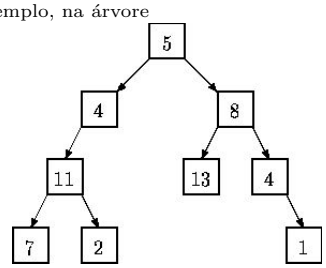
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 34 (6(4()())(18(16(9(7()())(14(13()())()))())(20(19()())())))
2.
 51 (17(3(1()(2()()))(4()(11(6()(10()()))(15()()))))(20()()))
3.
 76 (9(3(2(1()())())(6()()))(19(15(11()())(16()(17()()))())))
4.
 52 (3(2(1()())())(18(4()(11()(13(12()())()))))(19()(20()()))))
5.
 57 (18(3()(5(4()())(12(10(6()())())(14()(17(16()())()))))))())
```

Responda aqui:

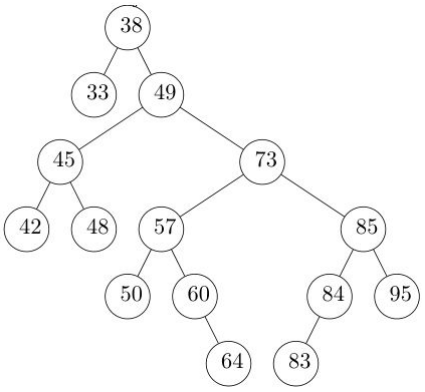
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(1()(13(9(6(3)(8(7)()))(11))(25(17()(19))())))
(12(4()(10(8)()))(19(17(15()(16))(18))(22(20)(24))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<{[(4-9)*1]-((1*7)+(5+4))}-{({4+7}+[9-6])*[(1-7)-(3*9)]}>
(<()4/3}*[9/3])-{5/5}>/<<{1-1}+{2-9}>-([6/9]-<2/9>)>)
<7*(<<9-4>-[8/5]>+((6/4)+(7+8)))>
[(<6-(6+6)>+{{9+5}+{5+3}})/{{5+[3/7]}+(7/{9*5})}]
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \wedge p) \wedge (p \vee (\sim q))) \wedge (q \vee q))$

12. Suponha que p vale 1 e q vale 1. Calcule:
$(((q \rightarrow p) \wedge (p \rightarrow (\sim q))) \wedge ((\sim q) \rightarrow (\sim p)))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$((((p \vee q) \rightarrow (q \wedge (\sim p))) \leftrightarrow (p \rightarrow p)) \leftrightarrow (p \wedge p))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$(((((\sim p) \wedge (\sim q)) \vee (q \leftrightarrow p)) \wedge (((\sim q) \wedge p)) \wedge (q \rightarrow p))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* 5 (* (+ (+ (+ 8 1) (- 6 3)) 2) 4))
(+ (+ (* (+ 8 3) (- 8 1)) (* (* 6 4) 1)) (* (+ (+ 6 3) 3) (* (- 5 3) (- 8 4))))
(+ (- (* (+ 6 4) (- 7 2)) 4) 4)
(+ (+ (+ 6 (+ 7 1)) (+ 2 (+ 8 3))) (+ 6 (* 1 (- (* 6 2) (+ 8 4)))))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75596 ERAN MARTINEZ RAMOS
24JOG307 - 4 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

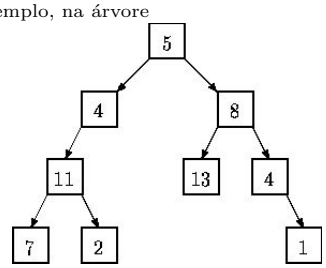
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 37 (2()(7(3()(6(4()())()))(13(10(9()())(11()()))(19()()))))
2.
 46 (6(2()(4()()))(20(16(15(12(11()())(14()()))())(19()()))()))
3.
 84 (6(4(1()())())(11()(19(14(12()())(16(15()())(18()()))())))))
4.
 38 (2()(16(9(4(3()())(6()(8()()))))(13()(15()()))))(19()()))))
5.
 72 (5(2()(4()()))(13(12(6()(9()()))())(16()(20(18()())()))))
```

Responda aqui:

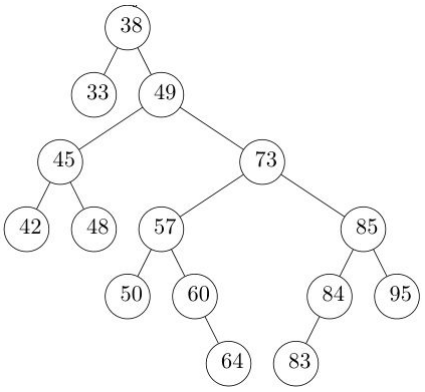
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(19(9(1()(6()(8)))(15(10()(13))(17)))(21()(25(24)(28))))
(21(14(6()(8()(12(9)()))))(18))(25(22)(26()(30(29)()))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
{({1-{3/4}}*{<5*8>/8}<)/{{(2+7)/4}-(<9+9>*2)}}
[([<7+2>+[2*9]]-<7/4/5]>)*[{{1+2}/<2*1>}-{[3+4]-[2*2]}]]
([<2-[6/6]>*({7+2}*7)]+[{<7-3>/{9/2}}/>((7-4)*(4+5))])
(((3*{5/2})+(7/(2-1)))/[{[1*8]/<1-2>}/<{6-1}+5>])
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((p \wedge q) \vee (q \vee (\sim q))) \leftrightarrow (q \leftrightarrow (\sim q)))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$(((q \rightarrow p) \leftrightarrow ((\sim p) \wedge (\sim q))) \vee (p \leftrightarrow q))$

13. Suponha que p vale 1 e q vale 0. Calcule:
$((((p \vee q) \rightarrow (p \wedge p)) \vee (p \wedge q)) \leftrightarrow (p \vee (\sim q)))$

14. Suponha que p vale 0 e q vale 0. Calcule:
$(((((\sim p) \wedge q) \leftrightarrow ((\sim q) \wedge p)) \wedge (p \wedge q)) \vee (p \wedge (\sim q)))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ 6 (* 1 (- (+ (+ 6 4) 4) (- (- 8 2) (- 7 2)))))
(+ 5 (- (* (* (+ 5 2) 5) (* (+ 6 4) 2)) 3))
(+ (+ 2 (* (* 5 2) (+ 8 1))) 1)
(+ (- 4 (- (+ 5 1) 3)) (+ (+ (* 5 4) (+ 5 2)) (- 4 (- 5 4))))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75608 HENRIQUE IVANKIO GUSSE
24JOG307 - 5 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

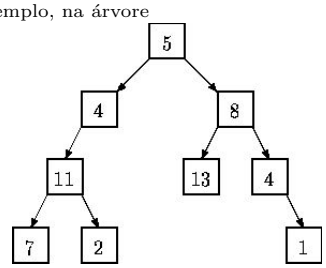
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 31 (3(1()(2()()))(4()(8(6()(7()()))(11()(16()(19()()))))))
2.
 55 (7(3(1()())())(9()(10()(11()(20(13()(15()(19()()))))))))
3.
 51 (18(9(8(3(2()())())())(14(11(10()())())(17()()))))(20()()))
4.
 55 (19(8(7(3()(4()()))())(15(14(10()())())(17()()))))(20()()))
5.
 49 (15(13(7(2()(3()(4()()))))(8()(9()()))))())(16()(18()())))
```

Responda aqui:

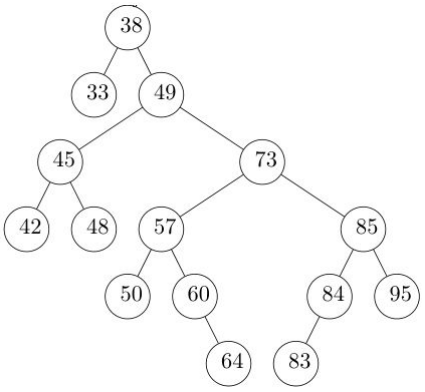
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14)())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)())))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(19(18(12(3()(8))())())(23(21(20)(22))(30(25)())))
(19(2()(13(6()(7()(10)))(15)))(20()(24()(29(27)(30)))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
{{{1+{9+1}}/{9-<9-2>}}-{<(4*5)*5>-[{4/1}-[6/2]]}}
(8*<((1*4)+<9-6>)-{[7*6]/{3*7}}>)
(4+<[{1+1}*[6*7]]+({1/9}*{7+1})>)
<<{[8+5]-1}/<{4/3}-[6-4]>>+([{8-4*{7/8}]+{9-8})>
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos

a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 1. Calcule:
$((((\sim p) \leftrightarrow q) \leftrightarrow (q \leftrightarrow p)) \rightarrow (q \vee p))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$(((p \vee q) \rightarrow (q \leftrightarrow q)) \wedge (q \wedge (\sim p)))$

13. Suponha que p vale 1 e q vale 1. Calcule:
$((((q \vee p) \wedge (q \wedge (\sim p))) \rightarrow (p \rightarrow (\sim q))) \rightarrow (p \leftrightarrow p))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$(((((\sim p) \wedge q) \rightarrow (p \rightarrow p)) \wedge (((\sim q) \wedge p)) \vee (p \wedge (\sim q)))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ (- (- (+ 7 4) 5) 4) 2)
(- (* (- (* 7 4) 3) 4) 1)
(+ (* (* (* 5 2) (- 8 2)) (+ (- 7 1) (- 5 1))) 6)
(- (* 3 (* 6 (- 8 1))) 1)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75615 IAN DOLABELLA OHATA
24JOG307 - 6 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

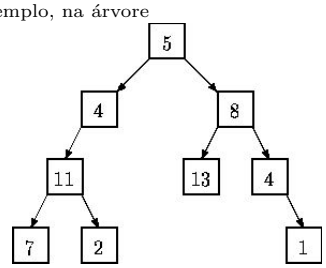
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 51 (2()(14(6(5(3()())())(10(9(8()())())())))(15()(17()())))
2.
 49 (15(2(1()())(5(3()())(9()(12()(14()()))))))(18(16()())()))
3.
 33 (18(6(5(4()())())(12()(13()(14()(15()(17()()))))))(19()()))
4.
 57 (4(2()())(15(12(10(8(6()())())())(13()()))(17()(18()()))))
5.
 32 (6(5(3()())())(15(11(9()())(14(12()())()))(19(16()())())))
```

Responda aqui:

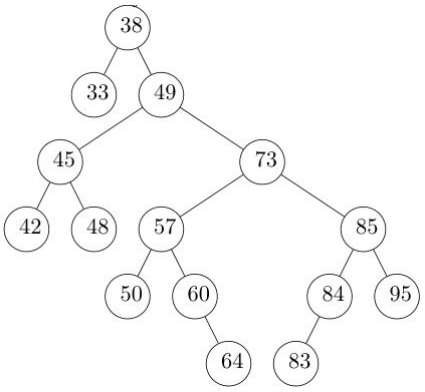
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(9(3(1)(6()(8)))(20(16(14)(19))(25(24(21)())(30))))
(21(18(17(4()(13(6()(8()(9()(10))))())))())(19))(23()(24)))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<{{{2/3}+8}*(1/{5+7})}*((7+<8-1>)*<<4*8>*4>)>
{[{(5/9)*9}-({1-3}-{6*2})]+}[<7*[3/6]>+(8*<6*5>)]}
{{}{8/2}*8}+{[5*4]-8}}+(<6/[3*4]>/<2/[3*5]>)}
((((4+1)-<8*5>)-[{6/5}/[8-4]])-{<1/<5*2>>*[{9+9+(1+9)]})
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \wedge p) \vee (q \leftrightarrow p)) \rightarrow ((\sim q) \vee q))$

12. Suponha que p vale 1 e q vale 1. Calcule:
$(((q \vee p) \leftrightarrow (q \vee (\sim q))) \rightarrow (q \vee (\sim p)))$

13. Suponha que p vale 1 e q vale 0. Calcule:
$(((((\sim q) \wedge p) \leftrightarrow (p \wedge q)) \wedge (p \leftrightarrow (\sim p))) \wedge (p \leftrightarrow q))$

14. Suponha que p vale 1 e q vale 1. Calcule:
$(((((\sim q) \leftrightarrow p) \vee (((\sim p) \rightarrow p)) \wedge (q \vee p)) \vee ((\sim p) \vee q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ (+ (* (* 8 2) (- 7 4)) (+ 1 (* 8 4))) 1)
(- (+ (* 6 (- 7 4)) (+ (* 6 4) 4)) (* (* 1 (+ (- 8 3) 1)) 3))
(- (- (- (* 6 4) 4) 1) 3)
(+ 1 (+ (+ 2 (* 6 2)) 1))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75622 JOAO BONFIN LINO
24JOG307 - 7 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

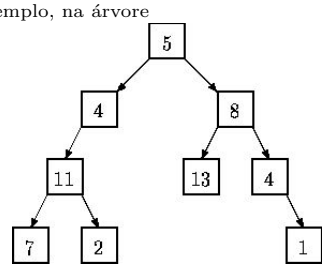
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
  3 (2(1()())(7(6(4()())())(13(8()(12(10()())()))(16()()))))
2.
 42 (5(3(2(1()())())())(13(9(8(7()())())())(20(17()())())))
3.
 53 (1()(11(4(3()())(6(5()())(8()(10()()))))(16()(19()()))))
4.
 54 (10(7()(8()()))(12(11()())(19(15(14()())(17(16()())()))())))
5.
 60 (6(2()(3()()))(17(12(11()())(14(13()())()))(18()(19()()))))
```

Responda aqui:

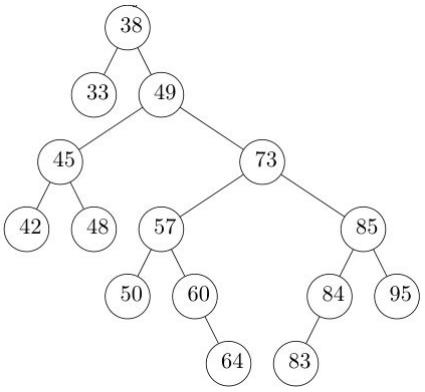
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83)())(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14)())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(13(12(8(6(2()(5))())())())(29(23(18(14)(19()(22)))(24))()))
(1()(11(2()(10(8()(9))()))(26(24(17(14)(18))())())))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
([(<4-4>/<1/4>)/{[4-5]*4}]*<[{6/9}-[2*8]]/{9+2}>)
{[(<9*5>*{5-1})/(<6*6>*3)]/{[4+{3-8}]*<<6/2>*(7-9)>}}
(<<(1/8)/(4/5)>+[<8*6>/]1+1]]>/4)
<{<(6/4)/7>-{[4/9]*[2*6]}}-([{5-9}/(6/4)]*{{9-4}-4])>
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos

a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge ((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge ((\sim p) \vee (\sim p))) \vee ((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 1. Calcule:
$(((q \leftrightarrow p) \wedge (q \vee (\sim p))) \wedge (p \rightarrow (\sim q)))$

12. Suponha que p vale 1 e q vale 1. Calcule:
$(((q \rightarrow p) \rightarrow (q \vee (\sim p))) \wedge (q \wedge p))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$(((((\sim q) \wedge p) \wedge (p \rightarrow q)) \rightarrow (q \rightarrow q)) \vee ((\sim p) \wedge q))$

14. Suponha que p vale 0 e q vale 0. Calcule:
$((((q \vee (\sim p)) \wedge (q \vee q)) \leftrightarrow (p \vee p)) \wedge (q \wedge q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* (- (+ (+ 7 1) 3) 4) 3)
(+ (- (* (- 7 2) 5) 5) (* 2 (* 1 (+ 8 2))))
(+ 2 (* 2 (+ 4 (+ 7 4))))
(+ 4 (+ 2 (- 2 (* 3 (- (+ 5 1) 6)))))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75639 JOAO LUCA FERNANDES DE OLIVE
24JOG307 - 8 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

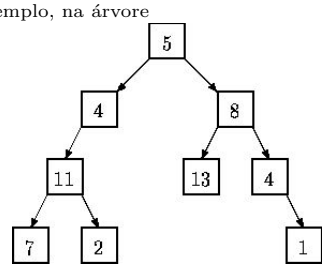
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 67 (19(1()(12(5()(10(7()(8()()))()))(17(15()())(18()()))))())
2.
 40 (19(12(7()(11(10(8()())())))(18(15(13()())())()))(20()()))
3.
 43 (18(16(15(2()(7(4()())(10()(12()()))))())())(19()(20()())))
4.
 56 (17(10(8(2()())())(13(11()())(16(15()())())))(19(18()())()))
5.
 74 (1()(15(9(3(2()())(7()()))(14(10()(12()(13()()))))())))
```

Responda aqui:

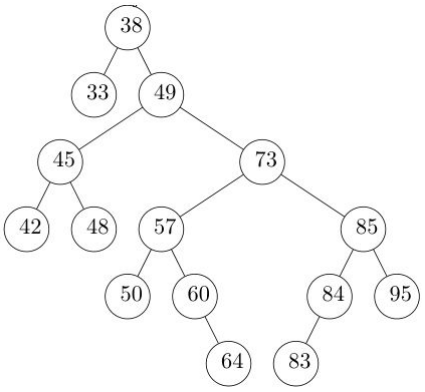
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(3(2)(13(6()(12(8(7)())()))(16()(30(20(18)(24()(27)))()))))
(29(4(3)(16(13(11(10)())())(28(20(19)(22))()))(30))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
[({{4*4<}*{3*9}}/{3+(5+5)})+<[{5*8}-(6/2)]*{8-<3*6>}>]
<(<9*{5+7>-[2+(4/1)])+(<8-(8-3)>-{{4/6}+7})>
[[((3/7)*(8+1))/{{2/3}/3}]+[({6/6}+(2+2))*[(3-1)*6]]]
<[<[8-9]+4>*{(7-1)-<4+7>}]-[<4+3>/{4*4}]-((8*9)-6)]>
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos

a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \vee p) \vee (q \leftrightarrow p)) \wedge ((\sim q) \vee (\sim p)))$

12. Suponha que p vale 1 e q vale 0. Calcule:
$((((\sim p) \wedge q) \vee (q \rightarrow q)) \wedge (p \rightarrow (\sim p)))$

13. Suponha que p vale 1 e q vale 0. Calcule:
$((((q \wedge p) \vee (p \leftrightarrow p)) \leftrightarrow ((\sim q) \rightarrow p)) \wedge (p \wedge p))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$((((p \vee q) \wedge (q \leftrightarrow p)) \vee (p \rightarrow (\sim p))) \wedge (q \wedge q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* 1 (- (* 4 (+ 7 2)) 5))
(* 4 (* (* 3 (- 5 1)) 6))
(- (- (+ (+ 6 3) 5) 1) 6)
(+ (- 4 (- (- 8 1) (- 8 3))) 2)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75646 JOAO PEDRO DE BRITO DUARTE
24JOG307 - 9 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

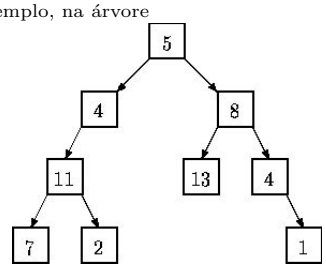
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 45 (18(4(2()())(10(5()(9(8()())()))(13()()))(19()(20()()))))
2.
 94 (20(16(14(4(3()())(12(11(8()(9()()))())()))())(17()()))())
3.
 13 (7(4(2()())(6()()))(9(8()())(16(15(11()(13()()))())())))
4.
 45 (11(1()(9(3(2()())())(10()()))))(18(15(12()())())(20()())))
5.
 24 (14(2()(5(3()())(6()(10()()))))(16(15()())(19(18()())())))
```

Responda aqui:

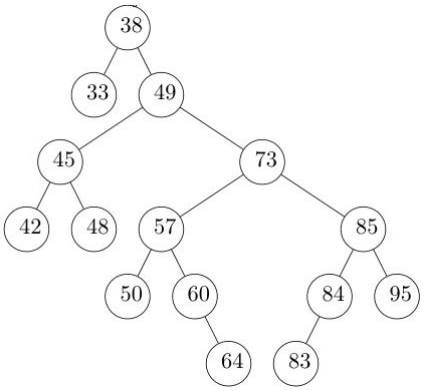
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14)())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)())))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(11(7(4(1)())())(17(13)(20(18()(19))(23()(26)))))
(6(2)(22(11(10)(14(13)(15()(18(16)()))))(24()(27()(30)))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
([<[1*6]-{1-5}>+{{4*4}-{3*5}}]*{[{7/7}+(1*3)]*{<5+9>+[3+3]}})
([[4/(2-5)]*<[5-7]-7>]+{(5*8)*>{3/7}-<7/5>>})
{({<7-1>-{9+7}}*{<6*3>-<3*1>})/[<[3/4]*1>*({9*7}+8)]}
{<{[3+8]+6}/[(2+8)+{1+1}]>/{<(4+1]/2>*((4+3)/{1*7})}}
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 0. Calcule:
$(((p \rightarrow (\sim q)) \wedge (q \vee p)) \rightarrow ((\sim q) \leftrightarrow p))$

12. Suponha que p vale 1 e q vale 0. Calcule:
$((((\sim p) \wedge q) \wedge (q \wedge p)) \wedge (p \wedge (\sim q)))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$((((q \vee p) \wedge (q \vee q)) \wedge (p \wedge q)) \leftrightarrow ((\sim p) \wedge q))$

14. Suponha que p vale 1 e q vale 1. Calcule:
$(((((\sim q) \wedge p) \leftrightarrow (p \wedge q)) \vee (q \wedge (\sim q))) \rightarrow (p \rightarrow (\sim q)))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* (* 2 (* (+ 5 3) (- 8 4))) 5)
(* 1 (- (* (- 7 3) (+ 7 2)) (+ 6 (* 5 2))))
(* (+ (- (- 8 2) 1) 4) 6)
(* (+ (+ (+ 6 2) 2) 1) (+ (* 2 (+ 5 1)) (- (- 8 4) 2)))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75653 JOAO PEDRO MARIANO SUEKE
24JOG307 - 10 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

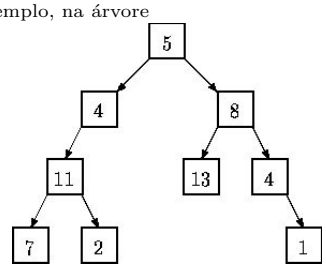
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 74 (12(2()(10(8(4()())())(11()())))(18(17(13()(14()()))())()))
2.
 34 (20(19(5(3()())(9(7()())(18(15()(16()(17()()))))()))())())
3.
 40 (20(17(10(7(1()(4()()))(9()()))(12(11()())()))(19()()))())
4.
 22 (15(4(1()(2()()))(11(10(5()())())()))(16()(19(17()())())))
5.
 20 (13(4(1()(2()()))(8(5()())(9()(12()()))))(19(15()())()))
```

Responda aqui:

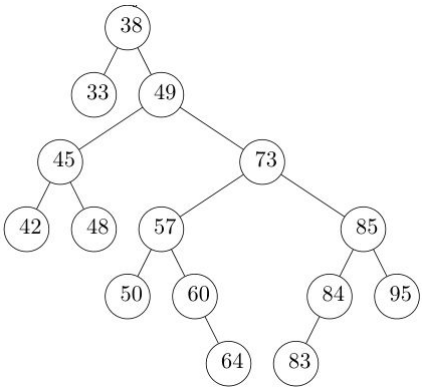
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(7(4)(18(12(8)(17(13)()))(27(19()(23))(29))))
(5(1()(2))(17(9(6()(7))(13))(24(22(21)(23))(25))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<<[[3+3]/{7+2]+{<8+1>-<6-3>}>*6>
([({7+4}+9}-{6*(8-2)}]*<[4+{4-1}]/[{6+7}-{1/5}]>)
<({<3*9>*<7/9>*({5/8}-(8*8)))-4>
([{<3-1>-<1/3>}-[]1*4]+<7/5>]]-9)
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 1. Calcule:
$(((p \rightarrow (\sim q)) \rightarrow (p \wedge q)) \leftrightarrow (p \rightarrow p))$

12. Suponha que p vale 1 e q vale 0. Calcule:
$(((q \wedge (\sim p)) \wedge ((\sim p) \wedge p)) \vee (p \wedge q))$

13. Suponha que p vale 0 e q vale 1. Calcule:
$((((p \wedge q) \wedge ((\sim q) \rightarrow p)) \vee ((\sim q) \leftrightarrow q)) \vee (q \vee q))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$(((((\sim p) \leftrightarrow q) \wedge (q \rightarrow q)) \wedge (p \wedge q)) \wedge (q \leftrightarrow (\sim p)))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ 3 (+ (+ 4 (- (* 7 4) (- 5 3))) 4))
(* 5 (* 6 (+ (- 8 1) 1)))
(+ (+ 2 (- (* 6 2) (- 8 2))) 5)
(+ 2 (+ 6 (* (- 6 4) 6)))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75660 JOAO VICTOR GOMES VILELA GER
24JOG307 - 11 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

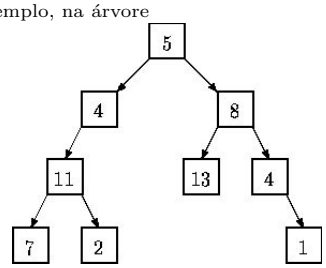
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 53 (7(4(1()())(5()()))(14(8()(13(11()())()))(18(15()())())))
2.
 39 (13(8(5(3(2()())())())(11()()))(20(19(15(14()())())())())))
3.
 59 (19(9(8(1()(5(3()(4()()))(7()()))))())(11()(16()()))))())
4.
 26 (4()(9(8(5()())())(10()(14(11()())(19(16()(17()()))()))))))
5.
 49 (13(12(10(8(4(3()())(5()()))())(11()()))())(16()(20()())))
```

Responda aqui:

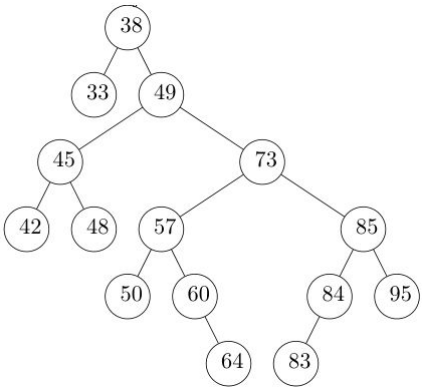
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(16(8(1()(2))(9()(13)))(23(21(19(18(17))())())(25(24)())))
(14(4(3(1()(2))())(8(6)(11)))(18(17)(28()(30(29)()))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
[({[2*5]-7}-{<9*1>+1})/<{<9*3>/6}+[{9/5}*2]>]
([{{5-9}-4}-<1+[7+9]>]*<<(4*1)-<6-4>>*{{6+1}*1)>)
<((<2-8>*{8-7})-[5*[1*7]])/(<<2-5>/2>+((1-2)+<1*)8>))>
{{(<2+4>/(5*6))+(<8*2>-[2+7])}/7}
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos

a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge ((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge ((\sim p) \vee (\sim p))) \vee ((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \leftrightarrow p) \vee (p \wedge q)) \leftrightarrow (q \wedge (\sim p)))$

12. Suponha que p vale 1 e q vale 1. Calcule:
$(((p \wedge (\sim q)) \vee (q \rightarrow (\sim q))) \rightarrow (p \leftrightarrow (\sim q)))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$((((q \wedge p) \wedge ((\sim p) \wedge p)) \wedge (p \wedge q)) \rightarrow (q \wedge (\sim q)))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$((((p \leftrightarrow q) \vee (q \wedge (\sim p))) \wedge (p \leftrightarrow p)) \wedge (q \vee q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ 1 (+ (+ (* 7 4) (* 7 1)) (* 4 (+ 8 2))))
(+ 2 (+ (+ (+ 7 2) 6) 6))
(* 5 (+ 3 (- (* 6 4) (* 7 1))))
(- (* (- (+ 5 3) 5) 2) 4)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75765 LUCAS ANTOSZCZYSZEN
24JOG307 - 12 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

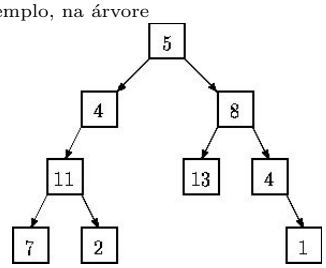
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 41 (13(8(3(2()())(4()()))(10()()))(14()(15()(19(17()())()))))
2.
 34 (1()(3()(9(5()(7()()))(10()(12()(20(17(15()())())())))))))
3.
 78 (4(1()())(20(10(5()(6()()))(16(15(13()())())(17()()))))()))
4.
 39 (19(15(13(9(2()(8(6(4()())())))))()())(18(17()())()))())
5.
 59 (20(9(8(5()(6()()))())(14(11(10()())(13()()))(16()()))))())
```

Responda aqui:

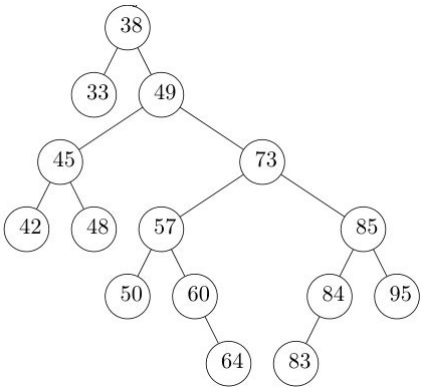
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(23(14(2(1)(8(3()(7))(11)))(18(17)()))(28))
(28(12(6(3(1)())())(27(18(16(15)())(25))()))(29))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<{((7+9)/8)-<[4*1*2>}*[[1*{4-5}]+<7*{3/1}>]>
{[{[9-9]/[7-5]}/<{6*1}+3>]/[[(4*4)*{4*6}]/[<3+9>-3]]}
(<[<9+4>-[3/2]]*(1/[2*5])>+{<[1-6]*(5+4)>*<<7+3>-{1+1}>})
[<{[2/5]+<1+1)}+<<2-9>+3>>*{(9-1)-[<6+2>*(4+2)]}]
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos

a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 1. Calcule:
$(((q \wedge (\sim p)) \wedge (q \wedge (\sim q))) \vee (q \leftrightarrow p))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$((((\sim p) \wedge q) \rightarrow (p \wedge q)) \vee (((\sim q) \wedge p))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$((((p \leftrightarrow q) \wedge (p \vee (\sim q))) \wedge (p \wedge p)) \wedge (q \rightarrow p))$

14. Suponha que p vale 0 e q vale 0. Calcule:
$(((((\sim q) \vee p) \rightarrow (p \rightarrow (\sim q))) \vee (p \rightarrow q)) \wedge (p \wedge q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ 3 (+ (- (- 7 1) 6) (* 3 (* 6 1))))
(* 4 (+ 1 (* 3 (+ 8 4))))
(+ 1 (- (- (+ (- 6 3) (* 6 1)) (* 1 (- 7 3))) (- 3 (- (+ 8 1) 6))))
(+ (+ (+ 6 (+ 6 2)) (+ (+ 5 2) (- 8 4))) (+ 5 (- (- (+ 8 4) 1) 4)))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75677 LUIZ FELIPE GUEDES BUCCHERI
24JOG307 - 13 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

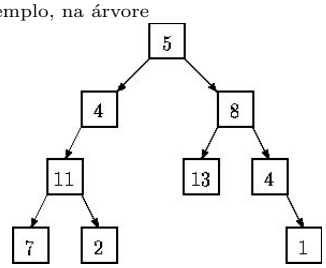
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 49 (19(7(5(4(3()())())())(11(10()())(18(12()())()))))(20()()))
2.
 45 (11(3(2()())(7()()))(16(13(12()())(15()()))(19(18()())())))
3.
 50 (2(1()())(12(6(5(4()())()())())(16(15(13()())())(20()()))))
4.
 35 (18(8(3(1()())(5()(6()())))(17(9()(16()())()))())(19()()))
5.
 38 (16(9(6(4(1()(2()()))())())(13(11()())()))(17()(19()())))
```

Responda aqui:

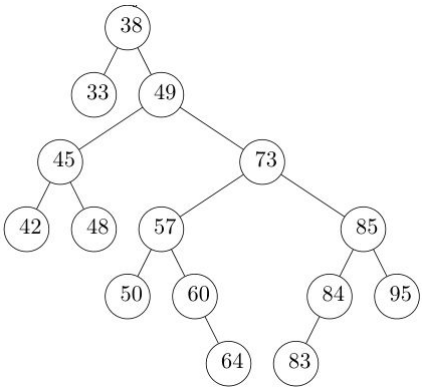
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14)())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)())))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(15(7(4)(8()(14(11)()))))(20(18(17)(19))(21()(29(26)()))))
(8()(28(19(13(10()(12))(14()(18)))(20()(25(22(21)())()))))(29)))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
(({<6/9>*(4+1)}+{1*<8*9>})*({3-(2*9)}-{6/<4/2>}))
<<([1/3]+(4*5))+{[9+1]-5}>*<{<3-5>+{9/2}}*(5+8)>>
{[{{2-5}*[2/1]}/<6*{4/9}>]+([2/(6-6)]/[{5+6}+1])}
{(({2+4}/[3*1])-<7+3>)+({5/<9+2>}/[<6/3>*9])}
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \vee p) \wedge (p \leftrightarrow (\sim q))) \vee (q \rightarrow (\sim q)))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$((((\sim p) \wedge q) \vee (p \vee q)) \wedge (q \wedge p))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$((((p \leftrightarrow (\sim q)) \vee (p \rightarrow q)) \vee (((\sim p) \rightarrow p)) \vee (p \wedge (\sim q)))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$((((p \wedge q) \wedge (((\sim q) \leftrightarrow p)) \vee (q \wedge (\sim p))) \leftrightarrow (((\sim q) \leftrightarrow p))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(- (+ (+ (+ 5 2) (+ 6 2)) 1) 2)
(+ (- 3 (- 5 (- 5 3))) 6)
(+ (+ 1 (- (+ 8 1) (- 5 1))) (+ 3 (+ (* 5 4) 3)))
(- (* 5 (+ (* 6 3) (- 6 4))) 1)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75684 MANUELLA LEAL DE MEIRELLES F
24JOG307 - 14 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

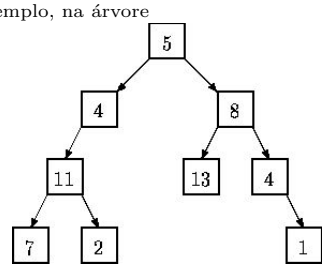
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 34 (4(2()())(12(10(9(5()(7()()))())())(19(15()(18()()))())))
2.
 90 (5(3()(4()()))(20(6()(8()(15(12()())(19(17()())()))))())))
3.
 48 (16(11(7(3()())(8()(10()()))))(15(13()())()))(17()(20()())))
4.
 23 (20(2(1()())(4()(13(8(5()())(11()()))(16(15()())()))))())
5.
 37 (13(1()(7(2()())(12()())))(14()(15()(16()(19()(20()()))))))
```

Responda aqui:

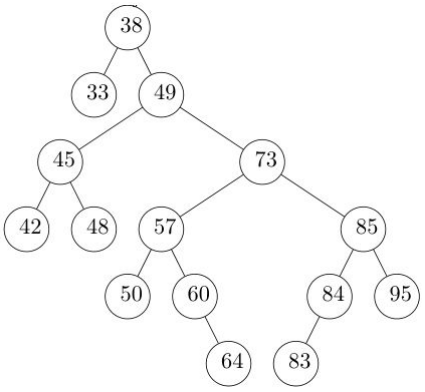
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(11(2()(7))(22(14(12()(13))(19(16)()))(26(25)(27()(29)))))
(12(10(6(2()(5))())())(20(16(15)(18))(23()(27()(28)))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<{((3*4)+[3+6])-{{1+3}->1}}+([2+<4*7>]/{{4/8}+(1*7)})>
{[[(6+5)+5)*<3+<4*6>>]*<((7{/3)*<6+6>)+[4/{1+6}]>}
{<<{3+2}*{5/9}>*{<2/2>-[9+4]}>+3}
([[7*(1+1)]+(1+<5-4>)]*[[6-{3-5}]+([8/7]-(6/9))])
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \leftrightarrow p) \leftrightarrow (q \wedge (\sim q))) \vee (((\sim p) \wedge (\sim p)))$

12. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \leftrightarrow (\sim p)) \rightarrow (p \rightarrow (\sim p))) \wedge (q \wedge (\sim p)))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$((((p \leftrightarrow q) \leftrightarrow (p \rightarrow (\sim q))) \vee (p \vee q)) \vee (((\sim q) \rightarrow (\sim p)))$

14. Suponha que p vale 1 e q vale 0. Calcule:
$((((q \leftrightarrow p) \wedge (q \wedge (\sim p))) \leftrightarrow (p \leftrightarrow q)) \vee (p \rightarrow q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* 4 (+ (+ (+ 8 4) 1) 3))
(+ (- (+ 6 (+ 6 1)) 4) 4)
(* 3 (+ (- (+ (+ 6 4) 6) 3) 4))
(+ (+ (* (+ 5 1) (- 5 1)) (* 2 (* 5 4))) 6)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75691 MARIA EDUARDA DA SILVA
24JOG307 - 15 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

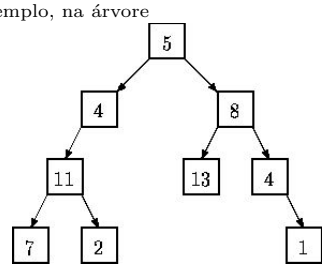
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 35 (9(5()(6()(8(7()())())))(14(11(10()())())(19()(20()()))))
2.
 33 (17(3(1()())(8(6()())(15(9()(13()()))()))))(19()(20()())))
3.
 80 (9(6(3(1()())())())(15(14()())(17(16()())(20(19()())()))))
4.
 16 (9(4(3()())(7(6()())()))(14(11(10()())())(18(15()())())))
5.
 44 (17(9(3()(7(6()())(8()())))(14(11()())()))(18()(20()())))
```

Responda aqui:

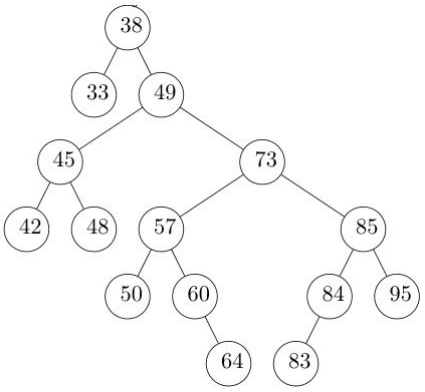
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(18(12(5()(11(9)()))(15()(17)))(27(23(21(20)())())(29(28)())))
(4(1)(18(8(6(5))())(17(9()(11))()))(21()(30(22)())))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
{(<{8+9}-{4-3}>/({9-7}+[9*3]))-[<{6-9}*(3+8)>*{[7/1]-(2*1)}]}
<6/{{[9-4+[1+8]}/{<8*1>*<3/1>}}>
<(<9+9>*[(7*2)+(8*1)])-<(<6/7>+[5+2])/([9+4]-{2+8})>>
[<(4/2)-<{2+6}+[5/4]>>-[(<9*3>+(8/8))*[9-(6-8)])]
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 0. Calcule:
$(((p \wedge q) \vee (q \vee (\sim p))) \rightarrow (((\sim p) \rightarrow (\sim q)))$

12. Suponha que p vale 1 e q vale 0. Calcule:
$(((q \rightarrow p) \rightarrow ((\sim p) \leftrightarrow p)) \vee (q \wedge (\sim p)))$

13. Suponha que p vale 0 e q vale 0. Calcule:
$(((((\sim q) \leftrightarrow p) \wedge (p \vee (\sim q))) \wedge (q \wedge p)) \wedge (p \vee p))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$((((p \rightarrow q) \rightarrow ((\sim q) \rightarrow p)) \leftrightarrow (q \rightarrow q)) \vee (q \rightarrow q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(- (+ 4 (+ 6 (* 6 3))) 4)
(+ (- (* 6 (- 6 1)) (+ (+ 8 1) (* 8 1))) (* (+ (- 8 2) (* 5 2)) 1))
(- (* 4 (+ (* 6 3) 6)) (+ (+ (+ 8 3) (+ 8 4)) (+ 3 (+ 5 3))))
(+ 4 (+ (+ 6 (* 5 1)) (- (* 7 4) 6)))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75703 MATHEUS AITA FABRICIO DE CAR
24JOG307 - 16 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

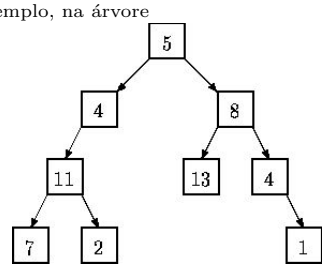
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 45 (10(9(3(1()(2()()))(8(4()())()))())(18(13(11()())())()))
2.
 55 (5(1()(3()()))(14(11(8()(9()()))(12()(13()()))(18()()))
3.
 42 (20(14(12(9(8(5(3(2()())())(7()()))())())(13()()))())())
4.
 32 (16(2()(8(6()())(9()(11(10()())(13()(14()()))))))(20()()))
5.
 49 (12(6(4(1()(2()()))(5()()))(9(7()())()))(20(17()())()))
```

Responda aqui:

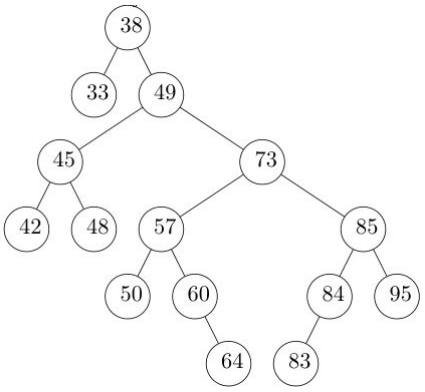
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(4(3)(16(8(7(5)())(13(12)(15)))(24(23)())))
(23(15(2(1)(10(3()(6))(13)))(18(17)()))(28(26)()))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
(<{[1/6]*}9*3}}*[(6/2)-<5-3>]>-({(8-6)*3}*[<4-8>*4]))
{{{[4/5]+5}/<{6*8}*(4/7)>}+[}<9-4>/{1-8}}/((6*6)+[7+4])]}
[{<(1/5)*{2+6}>-([6/7]+1)}/(<<9*6>-9>-<]2/4]+[7*1]>)]
<<[{8-1}-6]+{2-2}*(1+9)]>-[[(4/7)*3]+{6-<5-3>}]>
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos

a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 1. Calcule:
$(((p \rightarrow (\sim q)) \vee ((\sim p) \rightarrow q)) \wedge (q \vee (\sim p)))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$(((q \rightarrow p) \vee (p \wedge q)) \wedge (p \wedge q))$

13. Suponha que p vale 1 e q vale 0. Calcule:
$(((((\sim p) \wedge q) \rightarrow (q \wedge p)) \rightarrow (p \vee (\sim p))) \vee (q \wedge p))$

14. Suponha que p vale 1 e q vale 0. Calcule:
$((((p \rightarrow q) \vee ((\sim q) \wedge p)) \wedge (p \wedge q)) \vee (p \rightarrow q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ (* (* (+ 7 4) 1) (* (* 6 1) 2)) 2)
(* 2 (+ (* (+ 7 1) (* 5 4)) (* (- 8 2) (+ 5 3))))
(+ 3 (- (- (- (* 6 4) (* 5 4)) 1) 2))
(* 6 (+ (- (+ (* 6 2) (+ 6 3)) (+ 5 (+ 7 3))) (+ (* (* 7 3) 2) 3)))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75710 MATHEUS LEANDRO DE BITENCOUR
24JOG307 - 17 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

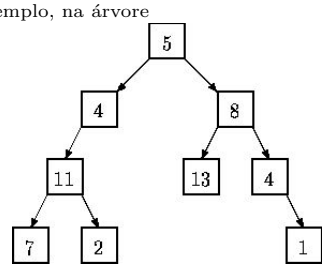
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))  sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))  não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 42 (6(1()(2()()))(13(11()(12()()))(18(16()(17()()))(20()()))))
2.
 35 (13(9(5(2()())(8()()))(11()()))(17(15()(16()()))(20()())))
3.
 51 (1()(15(4(3()())(13(9()())(14()())))(18(17()())(20()()))))
4.
 40 (16(4(1()())(8(5()(7()()))(13(12(10()()))()))))(20()()))
5.
 44 (9(6(1()(2()(3()())))(7()()))(11()(18(17()())(20()()))))
```

Responda aqui:

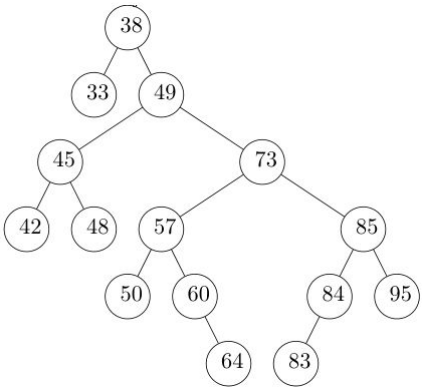
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14)())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(10(2()(7(5)(8)))(25(16()(20(17()(18))(21)))(30)))
(22(18(2()(4()(6()(13(11)()))))())(26(24)(29(27)())))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<{}[{2/6}+1]+{[2+2]/(6-2)}}+([3*[7/4]]*<[8+2]-5>)>
]<(6+{6+4})+<{9/3}-3>>+({{5+1}/6}/((4+2)*2))]
<[([6/6]*)1/8))*<(7+2)/{8*2}>]+1>
(<[9*3]*(<9-2>*<4+9>)>+{((4+3)-<3+8>)+(9*<8/3>)})
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos

a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 1. Calcule:
$((((\sim q) \vee p) \leftrightarrow ((\sim p) \wedge q)) \rightarrow (q \rightarrow p))$

12. Suponha que p vale 1 e q vale 0. Calcule:
$(((q \wedge p) \leftrightarrow (q \vee q)) \vee (q \rightarrow q))$

13. Suponha que p vale 1 e q vale 1. Calcule:
$((((q \wedge p) \wedge (q \wedge (\sim q))) \vee (p \wedge p)) \wedge (p \vee q))$

14. Suponha que p vale 1 e q vale 0. Calcule:
$((((q \wedge (\sim p)) \vee (p \leftrightarrow q)) \vee ((\sim p) \leftrightarrow (\sim q))) \wedge (q \wedge (\sim p)))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* (+ 6 (* (* 7 4) (* 7 2))) 2)
(* (* 1 (- (- 5 1) (- 5 2))) 3)
(* 6 (* 5 (* 4 (- 5 3))))
(* 4 (* (+ (- 8 3) 1) (+ (- 6 4) (* 7 2))))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75727 MAYCON TAVARES WOLPE
24JOG307 - 18 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

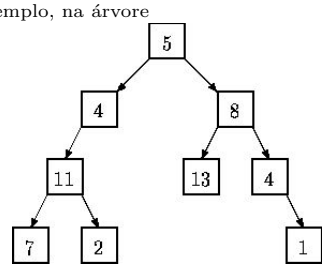
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 58 (3()(16(6()(10(7()())(15(14()())()))))(18(17()())(19()())))
2.
 45 (6(3(2(1()())())(5()()))(20(11(8()())(12()(15()()))())))
3.
 22 (10(6(4(2()())())(9()()))(11()(20(18(15()())(19()())))())))
4.
 78 (20(6(1()(2()(3()())))(7()(8()(18(15()())(19()()))))))())
5.
 54 (1()(7(4(2()())(6(5()())()))(19(10(8()(9()()))())())))
```

Responda aqui:

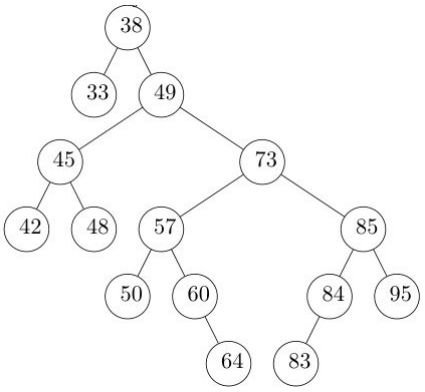
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(27(3(1)(17(12(5()(7))(13))(23(21)(25))))())
(18(13(6(4(3)())(8))(17))(27(21()(24))(28)))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
<{[9+[5*2]]-{{7-1}-(1*5)}}+<[{6-2}*<2+3>]/{[9-1]-(6/9)}>>
[({3-<2*2>}*{3+{1-6}})-([9/[8+6]]+<<4-5>*{1*7}>)]
{1-{({5/3}/[2/6])/[<7-5>+{8/1}]}}
(7-<{{6/8}+[1-8]}-<}2/8}+[6+6]>>)
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \leftrightarrow p) \rightarrow (q \rightarrow (\sim q))) \wedge (p \leftrightarrow p))$

12. Suponha que p vale 1 e q vale 1. Calcule:
$(((q \leftrightarrow p) \wedge (p \leftrightarrow (\sim q))) \wedge (q \leftrightarrow q))$

13. Suponha que p vale 1 e q vale 1. Calcule:
$((((q \vee (\sim p)) \leftrightarrow ((\sim p) \rightarrow p)) \vee (p \rightarrow (\sim q))) \wedge (q \wedge q))$

14. Suponha que p vale 0 e q vale 0. Calcule:
$((((p \rightarrow q) \rightarrow (q \wedge q)) \wedge (p \wedge (\sim q))) \vee (p \leftrightarrow (\sim p)))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(* 4 (- (* 4 (+ (- 8 1) 4)) (* 5 (+ (- 7 1) (- 6 4)))))
(+ 3 (+ (+ 3 (* 7 2)) 6))
(* 4 (+ (+ (+ 8 4) 6) 4))
(* (+ (+ 3 (* 8 4)) (* (- 5 3) (- 5 1))) 5)
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75734 PEDRO TOLEDO LEAL
24JOG307 - 19 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

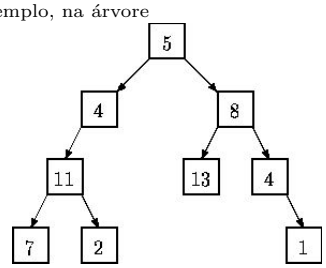
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))  sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))  não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 36 (13(9(2()(3()()))(11()()))(20(19(18(14()(17()()))())())()))
2.
 41 (10(2()(5(3()())(9(8()())())))(11()(17(12()(15()()))())))
3.
 31 (10(4(2(1()())())(7(6(5()())())(8()(9()()))))(20()()))
4.
 79 (17(10(8(2()(5(3()())()))())(11()(15(14(12()())())())))())
5.
 48 (2()(20(16(10(6(4()())(9(7()())()))())(17()(18()()))())))
```

Responda aqui:

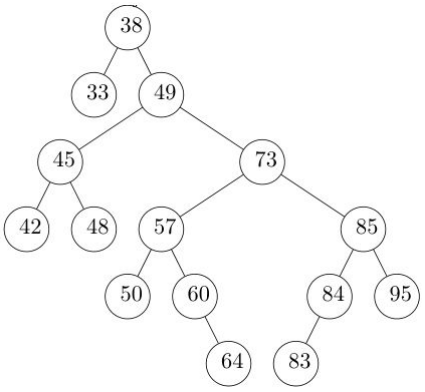
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(21(13(8(3(2)(5))(10))(19(14)()))(25(23)()))
(24(17(16(6(4)(12))())(22(18()(20))(23)))(28))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
([([3+8]-(7-6))/{8/{4+8}}]+<[(3*2)/{1/4}]/{{1+1}*<7-8>}>)
{({{8*3}*<4/6>}*<{7-5}*[5*1]>)/{[<7/9>+{1/8}]-(5/4)*<1+7>}}}
(({6/{2*5}}-[<3/1>*[3/6]])+<<<6-8>*[1+3]>/{[3*8]/{7-3}}>)
({[5/<3/1>]*<(6/9)/3>}-((6+5)/[<3/7>/{3/2}]))
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 0. Calcule:
$(((q \wedge p) \vee (q \wedge q)) \vee (p \wedge q))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$(((q \wedge p) \leftrightarrow (p \leftrightarrow q)) \rightarrow (p \wedge q))$

13. Suponha que p vale 1 e q vale 0. Calcule:
$((((p \leftrightarrow q) \rightarrow (q \wedge p)) \wedge (q \rightarrow p)) \rightarrow (p \leftrightarrow q))$

14. Suponha que p vale 0 e q vale 0. Calcule:
$((((q \vee p) \rightarrow (q \wedge p)) \leftrightarrow (p \rightarrow (\sim p))) \leftrightarrow (p \leftrightarrow p))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ (+ (* (* 8 2) (* 8 4)) 3) (- (+ (+ (- 7 2) (* 6 1)) 1) 3))
(* (* (+ 1 (+ 5 2)) (* 4 (* 7 4))) 1)
(* 3 (+ (- (+ 6 3) 2) 1))
(+ 5 (* 2 (+ (+ (+ 7 1) (+ 8 2)) 6)))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75741 PHELIPE GABRIEL LIMA DA SILV
24JOG307 - 20 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

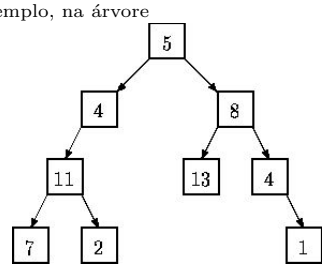
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))  sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))  não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 50 (2()(4()(11(7(6(5()())())())(16(15()())(19(18()())())))))
2.
 55 (6(5(3(1()())())())(10(7()())(14()(15()(18()(19()()))))))
3.
 32 (5(1()())(11(7()(9()()))(18(16(14()())(17()()))(19()()))))
4.
  7 (4(2(1()())()))(20(9(5()(8(7()())()))(16()(19()()))())))
5.
 32 (2()(8(3()(6(5()())()))(16(15(10()())())(18()(20()())))))
```

Responda aqui:

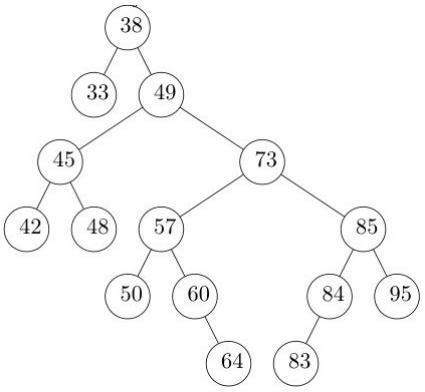
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83)())(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(6(3()(4))(27(10(8(7)(9))(20(19(12())(25)))()))
(2()(7(3()(5))(20(11()(19(17)()))(29(22)(30)))))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
{[{6-(7/6)}*({4+7}*7)]-[{[6*8]/9}+<(1+4)-[5/8]>]}
({{{8*2}*{1-3}}*[{2-1}-<5/6>]}-{([6*6]+(4*7))-(<9+1>*3)})
([{<5+8>*1}/{{8*3}*(7-8)}]/([{2/6}*3]/{<3-8>*(5-2)}))
<([{6*4}+[6*8]]-[9/{4+5}])/(<[4/6]*<5*5>>*<(5-8)/1>)>
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 1 e q vale 1. Calcule:
$(((p \leftrightarrow q) \wedge (q \wedge q)) \wedge ((\sim p) \wedge (\sim p)))$
12. Suponha que p vale 0 e q vale 1. Calcule:
$(((p \vee q) \wedge (p \vee q)) \leftrightarrow (q \wedge (\sim p)))$
13. Suponha que p vale 0 e q vale 0. Calcule:
$((((p \wedge q) \wedge (p \wedge (\sim q))) \leftrightarrow (((\sim q) \rightarrow (\sim p))) \rightarrow (((\sim p) \wedge (\sim q)))$
14. Suponha que p vale 1 e q vale 0. Calcule:
$((((p \wedge q) \wedge (p \rightarrow (\sim p))) \wedge (q \wedge q)) \leftrightarrow (p \wedge q))$
Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ (* (+ (+ 6 3) 5) (+ (* 6 1) 4)) (* (* (- 6 1) 4) 4))
(- (* (+ (- 5 4) (* 8 3)) 5) 4)
(- (+ (+ (* 7 2) 3) 6) 5)
(- (* (* (* 5 4) 1) (+ 3 (- 7 1))) (* 6 (- (+ 8 4) 4)))
```

Responda aqui:

15	16	17	18

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 05/08/2024 - 09:22:28.3
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
Representação de estruturas bidimensionais VIVX555a V: 1.03
75758 REBECA CABRAL DOS SANTOS
24JOG307 - 21 entregar ate 22/ago _____ / _____ / _____

Representação de estruturas bidimensionais

Na Tecnologia da Informação, muitas vezes é necessário representar estruturas com mais do que uma dimensão. Embora disponhamos de muitas ferramentas para isso, seria ideal contar com uma maneira linear de escrever tal representação.

Ainda, se for uma estrutura bem comportada (tal como uma matriz) embora tendo 2 dimensões ela ainda pode ser facilmente escrita como um vetor de vetores e portanto ser transcrita como uma tabela.

A coisa complica se precisarmos descrever uma árvore ou uma lista de listas, ou ainda uma expressão aritmética – com suas regras implícitas de prioridade.

É hora de lançar mão de um formalismo simples para nos ajudar nesta tarefa: trata-se do uso de parênteses para, por meio de relações de prioridade, de encadeamento, ou de qualquer nova relação que se defina, como reorganizar os elementos que agora podem ser escritos de maneira linear.

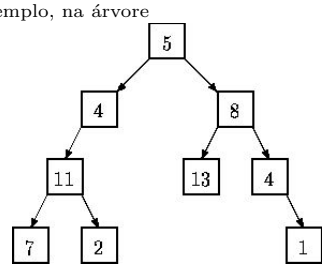
Parênteses

Os parênteses tem diversas funções no mundo da T.I.

- Definição da ordem de operações: Em expressões matemáticas, os parênteses determinam a ordem em que as operações devem ser realizadas, garantindo a clareza e o resultado correto da expressão. Eles definem subgrupos de operações, priorizando-as em relação às operações adjacentes sem parênteses. Essa padronização é fundamental para evitar ambiguidades e garantir a consistência dos resultados.
- Agrupamento de elementos em linguagens de programação: Em linguagens de programação, os parênteses são utilizados para agrupar blocos de código, delimitando funções, métodos, estruturas de controle e outras unidades de código. Essa organização facilita a leitura e a compreensão do código, além de permitir a execução de comandos em uma ordem específica. A sintaxe correta dos parênteses é essencial para o bom funcionamento dos programas.
- Argumentos em funções e métodos: Os parênteses também são utilizados para indicar os argumentos de funções e métodos em linguagens de programação. Eles contêm os valores que serão passados para a função ou método durante a sua execução, permitindo a parametrização e personalização do comportamento do código. A correta utilização dos parênteses garante que os argumentos sejam passados corretamente e na ordem esperada.
- Expressões regulares: Na manipulação de strings e textos, os parênteses desempenham um papel fundamental na criação de expressões regulares. Eles definem padrões de busca e agrupam sub-expressões, permitindo a localização, extração e substituição de partes específicas do texto de forma eficiente e precisa. Essa ferramenta é utilizada em diversas aplicações, como validação de dados, processamento de linguagem natural e extração de informações.
- Manipulação de dados em estruturas: Em linguagens de programação que utilizam estruturas de dados como listas, pilhas, filas e árvores, os parênteses podem ser utilizados para acessar, inserir e remover elementos da estrutura. Eles indicam a posição do elemento desejado ou o valor a ser inserido, permitindo a manipulação eficiente dos dados armazenados na estrutura. Essa funcionalidade é essencial para diversas aplicações que envolvem o armazenamento e o processamento de informações.

Árvores-UVA:112

Dada uma árvore binária (grau=2) de inteiros você deve determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore



há 4 caminhos, cujas somas são 27, 22, 26 e 18.
A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ())(8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha, qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

Você verá uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços.

Para cada caso, você deve informar "sim" caso haja o caminho somando o valor solicitado ou "não" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) sim

20 (5(4(11(7()())(2()()))())(8(13()())(4()(1()()))))) não

10 (3                                                  sim
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )

5 ()                                                   não
```

☞ Para você fazer

Responda sim ou não para as 5 árvores seguintes:

```
1.
 51 (8(6(3(2(1()())())()))())(11()(15(13()())(16()(20()()))))))
2.
 51 (1()(12(11(9(8(2()())())(10()()))())(18(13()())(20()()))))
3.
 60 (19(10(8(3(2()())()))())(15(13(12()())())(16()()))(20()()))
4.
 13 (2(1()())(6(5()())(19(14(8()(10()()))(16()(17()()))))()))
5.
 56 (4(2()())(16(14(8(6()())(10()(11()()))))())(17()(19()()))))
```

Responda aqui:

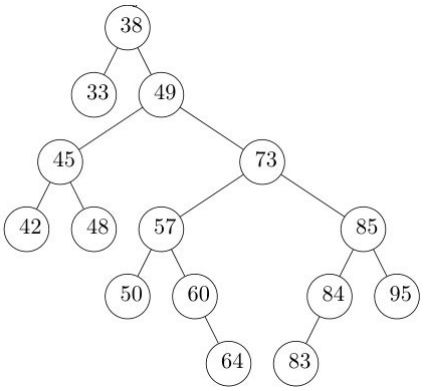
1	2	3	4	5

Árvore binária de pesquisa

Estrutura primordial na ciência da computação, é usada em pelo menos

- arquivos de índices de praticamente todos os sistemas gerenciadores de bancos de dados
- na maioria dos algoritmos de compressão (huffman, por exemplo)
- num dos melhores algoritmos de ordenação que existem
- em corretores ortográficos de texto

É uma estrutura hierárquica (logo não linear) que estabelece uma hierarquia que muitas vezes é descrita com termos usuais em uma família. Assim, cada elemento da árvore, aqui chamado **nodo** tem um pai e vários filhos. Eis o aspecto visual de uma árvore deste tipo



Note que, ao contrário da botânica, na ciência da computação as árvores crescem para baixo. A raiz está no alto e as folhas no chão.

Algumas definições:

lei fundamental da árvore binária de pesquisa (ABP) filhos **menores** que um nodo estão à esquerda e filhos **maiores** estão à direita.

raiz é o único nodo que não tem pai (no exemplo 38)

folhas nodos sem filhos (33, 42, 48, 50, 64, 83 e 95)

altura é o comprimento do maior caminho entre raiz e alguma folha (no exemplo 5)

irmãos nodos que tem o mesmo pai (50 é irmão do 60)

netos filhos dos filhos (no exemplo, 64 é neto do 57, 73 é neto do 38.

grau número máximo de filhos (no exemplo: 2, daí árvore binária)

Na representação parentizada, a (árvore x) esta hierarquizada por parênteses, na forma `(raiz(filhoesquerdo)(filhodireito))`. Note que `(x ()())` pode aparecer como `(x)`.

Eis como ficaria a representação do exemplo acima parentizado:

```
(38(33)(49(45(42)(48))(73(57(50)(60()(64)))(85(84(83))(95)))))
```

Exemplos

```
A árvore ABP
(4(3(1)())(13(7()(10))(21(17(14))())(30(24()(27()(28)))()))))
tem altura = 6 e a ABP
(28(20(6(3(1)(5))(7()(19(14)()))(24(23)(26)))())
tem altura = 5
```

☞ Para você fazer

Desenhe as 2 árvores a seguir e descubra a **altura** de cada uma

```
(7(2()(4))(18(13(11)(17(16)()))(19()(27(24)()))))
(26(2(1)(15(9(4)(14))(19(16)(20))))(27))
```

Responda aqui:

6	7

Analisador sintático

Você deve fazer o papel de um analisador sintático (simples) de expressões aritméticas parentisadas. Não deve entrar no mérito dos símbolos ou dos dígitos existentes na expressão. Outrém cuidará disso. Sua responsabilidade é apontar inadequação no uso de parênteses. Veja-se a seguinte tabela

expressão	certo/errado
`(a+b)`	certo
`((3+5)`	errado
`(((1*6)))`	certo
`)33-4(`	errado
`2*7`	certo

Nos exemplos acima, usou-se apenas um tipo de parênteses, mas nenhuma generalidade se perde se usarmos uma família maior de parênteses: será feito isso. Agora tem-se parênteses redondos `()`, parênteses bicudos `<>`, colchetes `[]` e chaves `{}`.

Obviamente, eles terão que vir aos pares, e cada parêntese aberto terá que ser fechado pelo seu homólogo.

O algoritmo para programar esta análise é bem simples: começa-se a expressão da esquerda para a direita. Usa-se uma pilha que começa vazia. Cada vez que um "abre" é encontrado ele é empilhado. Quando um "fecha" é encontrado, verifica-se se o topo da pilha é o seu homólogo de abertura. Se for, ambos são descartados e a análise segue. Se não for, tem-se um erro. Se ao final do processamento a pilha não estiver vazia, tem-se um erro. Se a pilha se esvaziar enquanto houver algum fechamento na expressão, tem-se um erro.

Você deve analisar a expressão e depois responder `sim` se a expressão tiver parênteses corretos e `não` se não tiver.

Exemplos Sejam as seguintes expressões parentizadas:

```
<[[9+(2+4)]+<[3/9]/<1*2>>]-(<(1/1)+(8+8)>+[<2/7>-{2-4}])>
[[{[5*4]-6}*{<8*8>-9}]+<(4/{3+1})/[7*<3-6>]>]
<[[<5-2>/<8*6>]+(1/2]+<4/1>)]/<([4*3]-{1-3})+(4-{3+3})>>
[[({4/8}/{2-3})/[<8+2>*{5+6}]]-(<[8+2]+{1-7}>+<9-}5*4}>)]
```

As duas primeiras estão corretas e as duas últimas estão erradas. Note `+(1/2]`, este parênteses redondo está errado. Na quarta Veja que externamente aos parênteses redondos há dois colchetes à esquerda e apenas um à direita.

Resolva os seguintes casos:

☞ Para você fazer

```
({([4+2]*[5/4])+{(1/6)+(3*9)}}/[[(6*1)*(6-1)]*[5+6]])
<({(8*2)/[7-4]}+<[4*2]-{4*7}>)-{{[8*9]+[2*9]}/(<9/2>-6)}>
{({<3-5>+<5*4>}*<5-[8/8]>)/{[<2-2>/3]+{(8+1)+{7-4}}}}
{<<[4-6]*[1+2]>+[<1-5>+{6*4}]>+5}
```

Responda aqui:

8	9	10	11

Lógica proposicional

Remontando à civilização grega (cerca de 2400 a.C.) é uma das grandes construções intelectuais da humanidade. Era desenvolvida e treinada pelos cidadãos gregos para suas discussões na ágora. Todos já devem ter ouvido falar na proposição *Todo humano é mortal, Sócrates é humano, logo....* Você deve estar se perguntando, estudar os gregos, logo agora, não é algo fora de moda ? Eu respondo: é uma das bases da mais moderna pesquisa em inteligência artificial.

A lógica proposicional estuda as proposições que são afirmações que só podem ter dois valores verdade: 0 (falso) e 1 (verdadeiro) – os gregos não usavam números, nós como estamos mais perto dos computadores do que eles,já fizemos uma pequena adaptação. Então por exemplo: Curitiba é uma cidade (1); A piscina tem água (1); A cor do céu é verde (0) e assim por diante.

As proposições podem ser operadas no chamado cálculo sentencial, através de 5 operações: E ($\wedge$), OU ($\vee$), NÃO ($\sim$), SE... ENTÃO ($\rightarrow$) e SE E SOMENTE SE... ENTÃO ($\leftrightarrow$).

Então se $p = 1$ e $q = 0$, a expressão $p \vee q$ certamente vale 1 e $p \wedge q$ vale 0. Tais resultados podem ser facilmente alcançados estudando-se a tabela a seguir:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\sim p$
1	1	1	1	1	1	0
1	0	0	1	0	0	
0	1	0	1	1	0	1
0	0	0	0	1	1	

Ao resolver uma expressão lógica, muitas vezes há a necessidade de usar parênteses, por isso elas estão aqui.

Exemplos a. com $p = 0$ e $q = 0$ pede-se
$(((q \vee p) \rightarrow (p \wedge q)) \vee (q \wedge q))$, que vale 1.
b. agora $p = 1$ e $q = 0$, então
$(((q \vee p) \rightarrow (q \wedge (\sim p))) \vee (p \wedge (\sim p)))$ vale 1.
c. p e q valem 1. resolvendo
$(((((\sim q) \leftrightarrow p) \wedge (q \vee q)) \vee (p \rightarrow p)) \wedge (((\sim p) \wedge (\sim q)))$, resulta 1.
d. $p = 0$ e $q = 1$, e tem-se
$((((q \vee (\sim p)) \wedge (q \leftrightarrow q)) \wedge (((\sim p) \vee (\sim p))) \vee (((\sim p) \vee q))$, cujo resultado é 0.

☞ Para você fazer

11. Suponha que p vale 0 e q vale 1. Calcule:
$(((p \vee (\sim q)) \vee (p \vee q)) \wedge (q \leftrightarrow p))$

12. Suponha que p vale 0 e q vale 1. Calcule:
$(((q \rightarrow p) \wedge (p \vee q)) \wedge (p \rightarrow q))$

13. Suponha que p vale 0 e q vale 1. Calcule:
$((((q \vee p) \wedge (q \vee p)) \leftrightarrow (p \wedge p)) \leftrightarrow (q \wedge p))$

14. Suponha que p vale 0 e q vale 1. Calcule:
$((((q \vee p) \rightarrow (p \vee p)) \rightarrow (p \wedge (\sim q))) \wedge ((\sim p) \vee q))$

Responda aqui:

11	12	13	14

Notação pré-fixada

Desde criancinhas, aprendemos a escrever uma expressão aritmética como $a + b$, querendo indicar a adição de a e b. É a chamada notação in-fixada, na qual a operação é escrita entre os operandos. Tem a grande vantagem de isolar cada operando, e provavelmente por isso que ela é usada.

Entretanto, outras notações tem sido desenvolvidas e usadas. A Linguagem LISP, usa a notação pré-fixada, na qual a expressão $a + b$ é escrita como $+\ a\ b$.

Já a notação polonesa reversa (RPN) inventada por Charles Hamblin na década de 50 e popularizada (com este nome) por Jan Lukasiewicz, usa notação pós-fixada na qual a mesma expressão é escrita como $a\ b\ +$. A importância da RPN, reside na simplicidade da análise sintática pelo que (até onde sei) é implementada por TODOS os compiladores de todas as linguagens.

Aqui, vamos interpretar e depois executar expressões escritas na forma pré-fixada. Para evitar confusão, a expressão $a + b$ vai ser escrita entre parênteses, assim $(+\ a\ b)$.

Exemplos Sejam 4 expressões:

```
(* (- (+ (* 6 2) 1) 5) (+ 1 (+ (* (- 7 2) 2) 2)))
(* (+ 2 (- (* 6 3) (- 5 4))) 3)
(- (* 2 (- (+ 6 4) 2)) 5)
(* 5 (+ (* (+ 5 4) (+ 8 3)) 2))
```

Cujos resultados são 104, 57, 11 e 505.

☞ Para você fazer

```
(+ (- (* (+ 6 4) (+ 8 3)) 6) 6)
(+ (* 5 (+ (- 8 1) (- 7 4))) (* 4 (+ 4 (* 5 4))))
(* (- (+ (* 7 4) 6) (* (* 8 1) (- 6 4))) (* 3 (* 2 (- 6 1))))
(* 3 (- (* (+ 6 3) 2) (+ (- 8 2) 2)))
```

Responda aqui:

15	16	17	18