Mecanismos construtores de tipos

1) [Questão 37 do Poscomp 2011] Em programas que utilizam grande quantidade de memória, a alocação deste recurso deve ser realizada com muito cuidado. Em algumas circunstâncias, o uso da memória pode ser otimizado com a utilização de registros variantes. Em linguagens como C, o registro variante é construído através de uma união disjuntiva.

Analise a declaração de tipo em C++, a seguir.

```
union PosCompType {char A[2];
                   struct {char B;
                           char C;
                          };
                  };
```

Considere o código a seguir, que utiliza esse tipo.

```
int main()
{   PosCompType Dado;
    Dado.A[0] = 'a';
    Dado.A[1] = 'b';
    Dado.B = 'c';
    Dado.C = 'd';
    printf ("%c %c %c %c\n", Dado.A[0], Dado.A[1], Dado.B, Dado.C);
    return 0;
}
```

A saída do código será:

a) a b a b

b) a b c d

c) c d a b

d) c d c d

e) d c b a

TDA – Tipos Abstratos de Dados

2) **[Questão 25 do Poscomp 2013]** As Estruturas de Dados (ED) são representadas classicamente por Tipos Abstratos de Dados (TAD), que permitem definir e especificar estas estruturas. Cada TAD pode ter diferentes tipos de operações, mas há três operações que são básicas e devem existir em qualquer TAD (além da definição de tipo de dado).

Assinale a alternativa que apresenta, corretamente, essas três operações básicas.

a) TAD de Pilha: Definição do dado (tipo utilizado) e as operações de inclusão inserção (empilhamento), remoção (desempilhamento) e impressão (apresentação dos dados).
b) TAD de Pilha: Definição do dado (tipo utilizado) e as operações de inserção, remoção e impressão (apresentação dos dados).
c) TAD de Fila: Definição do dado (tipo utilizado) e as operações de inserção, remoção e inicialização (criação) da estrutura.
d) TAD de Fila: Definição do dado (tipo utilizado) e as operações de inicialização (criação), inserção e impressão (apresentação dos dados).
e) TAD de Lista: Definição do dado (tipo utilizado) e as operações de inicialização (criação), inserção numa posição da Lista e remoção de todos os elementos da Lista (destruição da lista).

3) **[Questão 28 do Poscomp 2004]** Qual das seguintes expressões posfixas é equivalente à expressão infixa `A+(B/C)*((D-E)/F)`?

a) `ABC/-DE*F+/`

b) `ABC/DE-/F+*`

c) `ABC/DE-F/*+`

d) `ABC/D-EF*/+`

e) `ABD/CE+/F-*`

4) Converter as expressões infixas a seguir para prefixa e posfixa:

a) (A + B) * C
b) A + B * C + D
c) (A + B) * (C + D)
d) A * B + C * D
e) A + B + C + D
f) A + B – C
g) A / B * C – D + E / F / (G + H)
h) ((A + B) * C – (D – E)) * (F + G)

5) Descreva a avaliação passo a passo, realizada com auxílio de uma pilha, das seguintes expressões posfixas:

a) 4 5 6 * +

b) 1 2 – 4 5 + *

6) A seguir é apresentado um algoritmo que converte uma expressão infixa totalmente parentesiada em sua correspondente posfixa. Descreva passo a passo a execução desse algoritmo para a expressão `((1+(2*3))-4)`.

Algoritmo ConverteInfixaParaPosfixa

```
Inicie com uma pilha vazia e uma string para expressão
posfixa também vazia
```

Percorra a expressão infixa da esquerda para a direita e, para cada elemento encontrado, faça:

- Se for um parêntese de abertura, descarte-o
- Se for um operando, acrescente-o à expressão posfixa
- Se for um operador, coloque-o na pilha
- Se for um parêntese de fechamento, descarte-o, retire um operador da pilha e acrescente-o à expressão posfixa

FimAlgoritmo

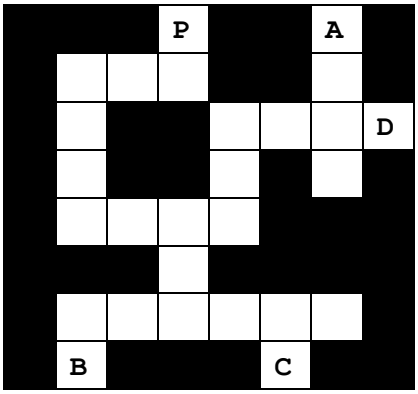
7) Escreva as expressões a seguir nas 3 formas clássicas

Expressão	Infixa (notação convencional)	Prefixa (notação polonesa)	Posfixa (notação polonesa reversa)
$\frac{a + b}{c}$	(a + b) / c	/ + a b c	a b + c /
$\frac{a * b - c * d}{e * f}$			
$\frac{a + b}{2} - \frac{c * d}{3}$			
$\frac{a}{2} * \frac{b}{3} * \frac{c}{4} * \frac{d}{5}$			
$\frac{a}{2} - \frac{b}{3} * \frac{c}{4} + \frac{d}{5}$			
$\frac{a}{2} * \frac{b}{3} + \frac{c}{4} * \frac{d}{5}$			
$a^3 + \frac{3}{b} * \frac{c}{4}$			
$\frac{127 + 3 * c}{2 * a + 4 * b}$			
$\frac{- b + d^{1/2}}{2 * a}$			
$\frac{a + 3 - 4 * c}{1024}$			

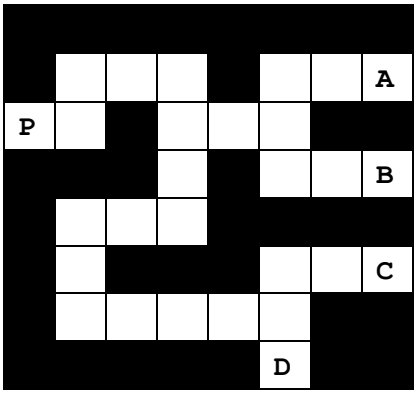
Listas lineares

8) A seguir são dados dois labirintos onde o indivíduo está inicialmente no ponto de partida *P*. As saídas são indicadas pelas células marcadas com as letras *A*, *B*, *C* e *D*. Considerando que o indivíduo utilizará a lógica de exploração de labirinto expressa no pseudocódigo mostrado em seguida, indique para cada labirinto por onde ocorrerá a saída.

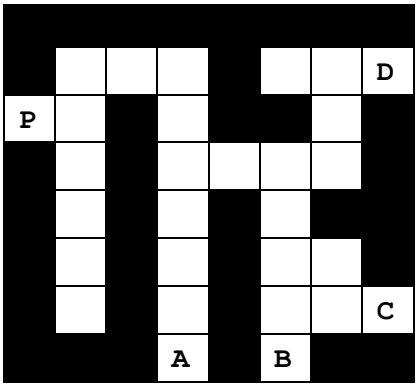
```
InicializaPilha()
Empilha(p)                  // p contém a linha e a coluna de partida no labirinto
Enquanto PilhaNaoVazia() Faça
    pos = Desempilha()
    Se mapa[pos.linha][pos.coluna] é uma saída Então
        Sai do looping
    Senão
        Se mapa[pos.linha][pos.coluna] é uma posição livre ainda não visitada Então
            mapa[pos.linha][pos.coluna] = VISITADO
        FimSe

        aux.linha = pos.linha - 1
        aux.coluna = pos.coluna
        Se aux é uma posição válida livre ainda não visitada Então
            Empilha(aux)
        FimSe

        aux.linha = pos.linha
        aux.coluna = pos.coluna + 1
        Se aux é uma posição válida livre ainda não visitada Então
            Empilha(aux)
        FimSe

        aux.linha = pos.linha + 1
        aux.coluna = pos.coluna
        Se aux é uma posição válida livre ainda não visitada Então
            Empilha(aux)
        FimSe

        aux.linha = pos.linha
        aux.coluna = pos.coluna - 1
        Se aux é uma posição válida livre ainda não visitada Então
            Empilha(aux)
        FimSe
    FimSe
FimFaça
```



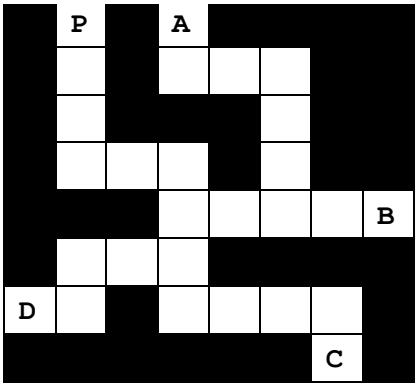
Labirinto 1



Labirinto 2



Labirinto 3



Labirinto 4

9) Considere um programa onde é definido o tipo TL e as variáveis `p`, `aux` e `x` mostradas a seguir, onde `p` e `x` são descritores da lista. São apresentadas 4 pequenos pedaços de código, cujos nomes são *`codigo1`*, *`codigo2`*, *`codigo3`* e *`codigo4`*. Assinale as afirmativas como V (verdadeira) ou F (falsa).

```
struct registro
{
      int valor;
      struct registro *prox;
};
typedef struct registro TL;

(...)
```

```
/* Codigo1 */
(...)
aux = (TL *) malloc(sizeof(TL));
aux->valor = numero;
aux->prox = p;

p = aux;
(...)
```

```
/* Codigo2 */
(...)
aux = (TL *) malloc(sizeof(TL));
aux->valor = numero;
aux->prox = NULL;

if (p == NULL)
   p = aux;
else
   x->prox = aux;

x = aux;
(...)
```

```
/* Codigo3 */
(...)
if (p != NULL)
{  aux = p;
   p = p->prox;
   if (p == NULL)
      x = NULL;
   free(aux);
}
(...)
```

```
/* Codigo4 */
(...)
If (p != NULL)
{  aux = p;
   p = p->prox;

   free(aux);
}
(...)
```

( ) O trecho *codigo1* é uma rotina de inclusão na cabeça, típica de uma estrutura LIFO, comumente chamada de pilha. A variável `p` seria o ponteiro para o topo da pilha.

( ) O trecho *codigo2* é uma rotina de inclusão na cauda, típica de uma estrutura FIFO, comumente chamada de fila. Nesse caso, `x` é um ponteiro para o último elemento da lista.

( ) O trecho *codigo3* é uma rotina de exclusão de elementos de uma fila. Nesse caso, `x` é um ponteiro para o último elemento da lista.

( ) O trecho *código4* é uma rotina de exclusão de elementos de uma pilha. Como é característico de estruturas LIFO, apenas um descritor é necessário e o programa usa `p` para indicar o topo da pilha.

( ) Se o programa utilizar o trecho *codigo1*, terá que utilizar na exclusão o trecho *codigo4*.

( ) Se o programa utilizar *codigo1*, terá que utilizar na exclusão *codigo3*.

( ) Se o programa utilizar *código2* para a inclusão, com o programa recebendo os valores 1, 2, 3 e 4, nesta ordem, o conteúdo da lista será `p->[1]->[2]->[3]->[4]/`

( ) Se o programa utilizar *código2* para a inclusão, com o programa recebendo os valores 4, 3, 2 e 1, nesta ordem, o conteúdo da lista será `p->[1]->[2]->[3]->[4]/`

10) [Questão 27 do Poscomp 2009] Considere as estruturas de dados a seguir.

- Uma lista é um conjunto de dados onde cada elemento contido na lista ocupa sozinho uma posição de 1 até n, onde n é a quantidade de elementos na lista. Uma inserção ou remoção pode ser realizada em qualquer posição da lista.
- Uma fila é um caso especial de lista onde a inserção só pode ser realizada em uma extremidade e uma remoção na outra.
- Uma pilha é um caso especial de lista onde uma inserção ou uma remoção só podem ser realizadas em uma extremidade.

Analise as afirmativas seguintes sobre essas estruturas de dados:

I. Uma fila pode ser implementada usando duas pilhas;

II. Uma pilha pode ser implementada usando duas filas;

III. Uma lista pode ser implementada usando uma fila e uma pilha.

Assinale a alternativa CORRETA:

a) Apenas a afirmativa I está correta.

b) Apenas a afirmativa II está correta.

c) Apenas a afirmativa III está correta.

d) Apenas as afirmativas I e II estão corretas.

e) Apenas as afirmativas I e III estão corretas.

11) [Questão 25 do Poscomp 2006] Dada uma lista linear de n+1 elementos ordenados e alocados sequencialmente, qual é o número médio (número esperado) de elementos que devem ser movidos para que se faça uma inserção na lista, considerando-se igualmente prováveis as n+1 posições de inserção?

a) $n/2$

b) $(n + 2)/2$

c) $(n - 1)/2$

d) $n(n + 3 + 2/n)/2$

e) $(n + 1)/2$

12) [Questão 16 do ENADE 2014 para Bacharel em Ciência da Computação] Uma pilha é uma estrutura de dados que armazena uma coleção de itens de dados relacionados e que garante o seguinte funcionamento: o último elemento a ser inserido é o primeiro a ser removido. É comum na literatura utilizar os nomes `push` e `pop` para as operações de inserção e remoção de um elemento em uma pilha, respectivamente. O seguinte trecho de código em linguagem C define uma estrutura de dados pilha utilizando um vetor de inteiros, bem como algumas funções para sua manipulação.

```
#include <stdlib.h>
#include <stdio.h>
typedef struct {int elementos[100];
                int topo;
               }pilha;
pilha *cria_pilha()
{   pilha *p = malloc(sizeof(pilha));
    p->topo = -1;
    return pilha;
}
void push(pilha *p, int elemento)
{   if (p->topo >= 99)
       return;
    p->elementos[++p->topo] = elemento;
}
int pop(pilha *p)
{   int a = p->elementos[p->topo];
    p->topo--;
    return a;
}
```

O programa a seguir utiliza uma pilha.

```
int main()
{   pilha * p = cria_pilha();
    push(p, 2);
```

```
    push(p, 3);
    push(p, 4);
    pop(p);
    push(p, 2);
    int a = pop(p) + pop(p);
    push(p, a);
    a += pop(p);
    printf("%d", a);
    return 0;
}
```

A esse respeito, avalie as afirmações a seguir.

I. A complexidade computacional de ambas funções `push` e `pop` é O(1).

II. O valor exibido pelo programa seria o mesmo caso a instrução `a += pop(p);` fosse trocada por `a += a;`

III. Em relação ao vazamento de memória (*memory leak*), é opcional chamar a função `free(p)`, pois o vetor usado pela pilha é alocado estaticamente.

É correto o que se afirma em

a) I, apenas.
b) III, apenas.
c) I e II, apenas.
d) II e III, apenas.
e) I, II e III.

Árvores

13) Suponha uma árvore binária ordenada com a estrutura onde cada elemento é composto por um dado do tipo char e os ponteiros para os filhos da direita e da esquerda. Desenhe como ficaria essa árvore caso ela fosse alimentada com a sequência dos 15 primeiros caracteres que formam o seu nome, sem abreviações e em maiúsculas, desconsiderando os espaços em branco.

14) Desenhe árvores binárias ordenadas, construídas conforme o algoritmo discutido em aula, para as sequências de valores indicadas abaixo:

a) 31, 21, 10, 32, 4

b) 21, 31, 32, 10, 4

c) 4, 10, 21, 31, 32

d) 32, 31, 21, 10, 4

e) 48, 9, 37, -6, -1, 5, -4

f) 29, 35, 35, 33, 48, 20, 19, 6, 2, 48, 22, 36, -4, 10, -1

g) 39, 39, 32, 29, 47, 35, 25, 5, 48, 43, 6, 8, 23, 4, 1, 45, 42

15) Preencha o quadro a seguir indicando quantos nós podem existir em cada nível das árvores, conforme o seu grau (grau é a quantidade de descendentes diretos que um nó pode ter).

Grau 2	
Nível	**Qtde**
0	
1	
2	
3	
4	
5	
6	
7	
8	

Grau 3	
Nível	**Qtde**
0	
1	
2	
3	
4	
5	
6	
7	
8	

Grau 4	
Nível	**Qtde**
0	
1	
2	
3	
4	
5	
6	
7	
8	

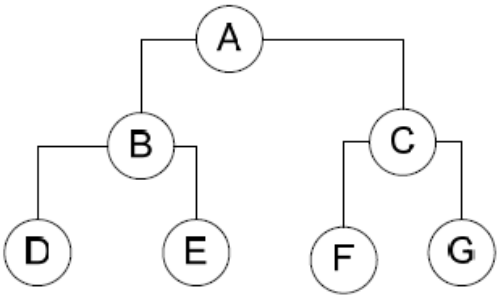
Grau *n*	
Nível	**Qtde**
0	
1	
2	
3	
4	
5	
6	
7	
8	

16) Com base nas descrições das árvores binárias dadas a seguir em pré-ordem e em-ordem, determine qual a descrição em pós-ordem correspondente.

a) Pré-ordem: ABDGCEHIF Em-ordem: DGBAHEICF

b) Pré-ordem: ABCEIFJDGHKL Em-ordem: EICFJBGDKHLA

17) [Questão 33 do Poscomp 2009] Percorrendo a árvore binária a seguir em pré-ordem, obtemos qual sequência de caracteres?



a) A C G F B E D

b) G C F A E B D

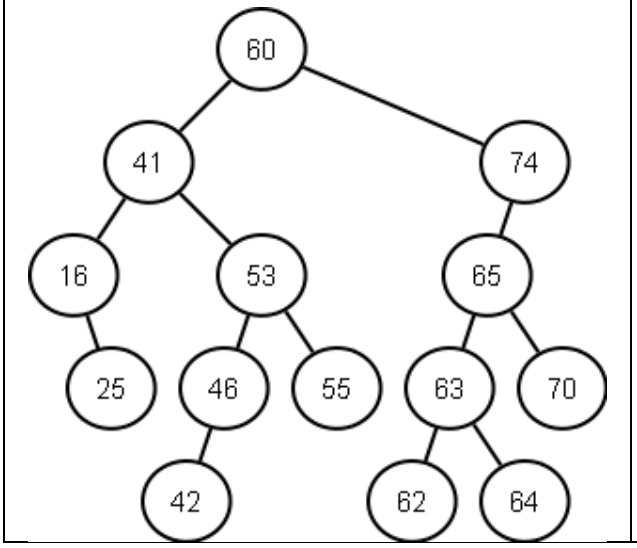
c) A B C D E F G

d) A B D E C F G

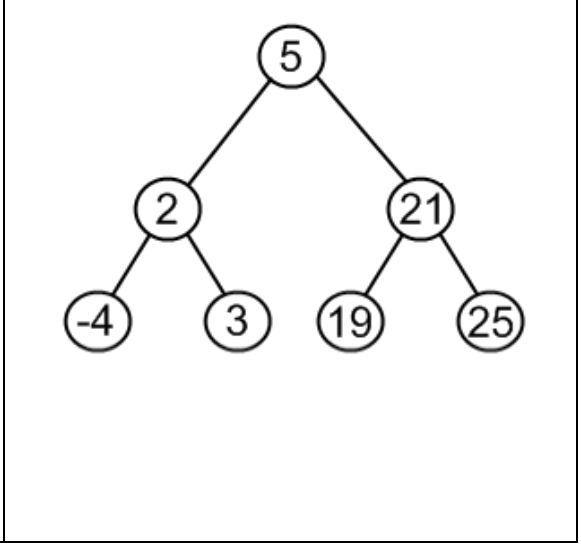
e) D B E A F C G

18) Para cada árvore apresentada a seguir, desenhe seu estado final após os elementos indicados terem sido excluídos utilizando o algoritmo descrito nas páginas 148 a 155 do livro do Veloso.

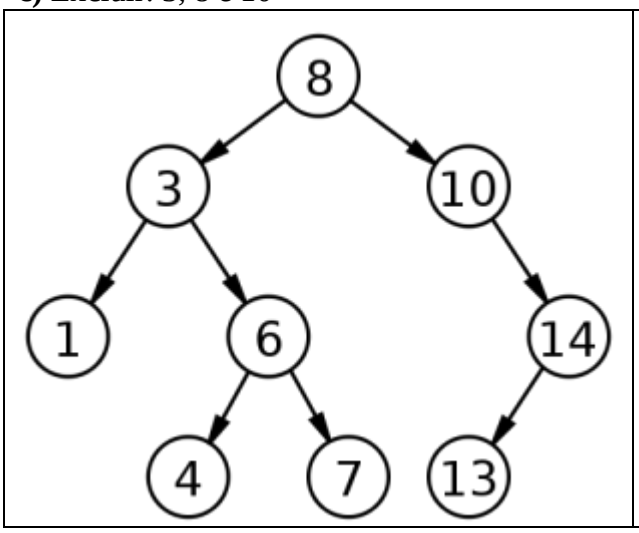
a) Excluir: 25, 53 e 74



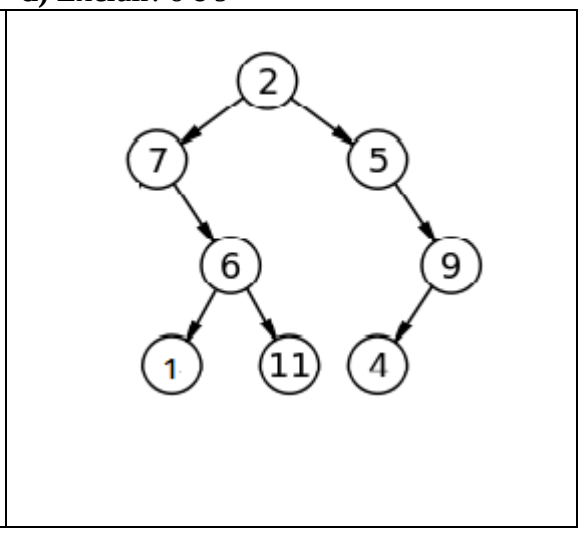
b) Excluir: 19, 21 e 2



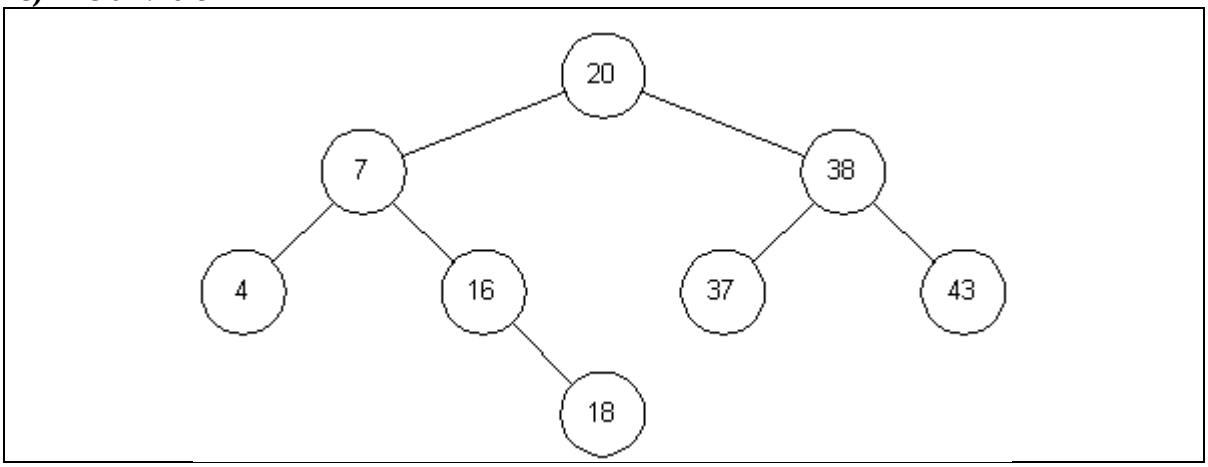
c) Excluir: 3, 8 e 10



d) Excluir: 6 e 9

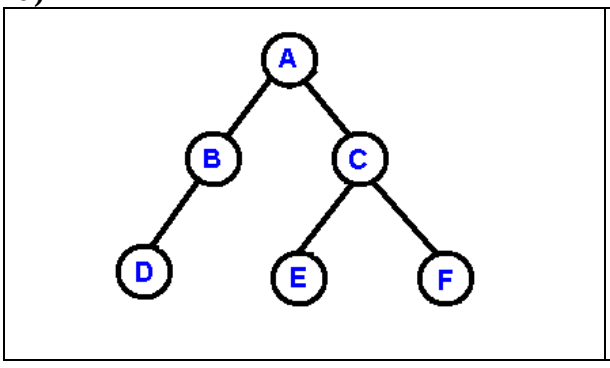


e) Excluir:20 e 7

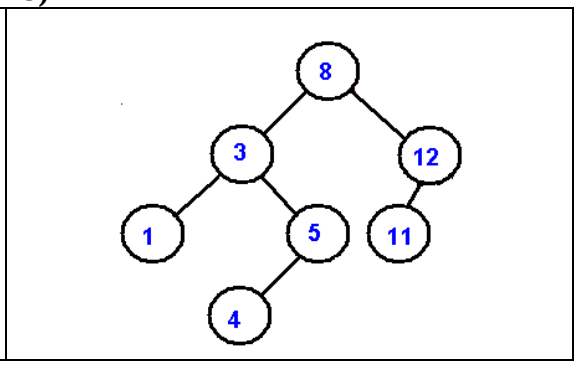


19) Escreva a descrição textual para as árvores apresentadas a seguir, usando os critérios pré-ordem, pós-ordem e em-ordem.

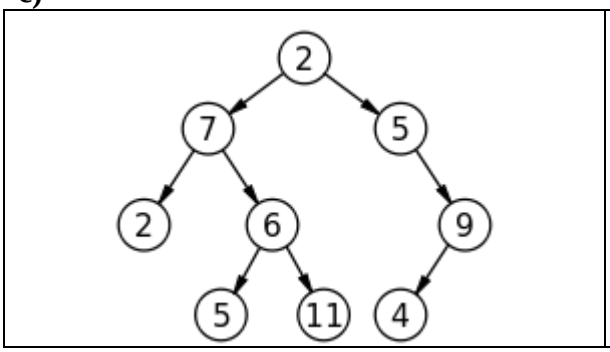
a)



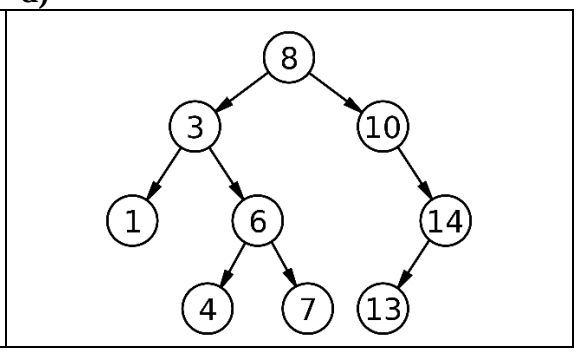
b)



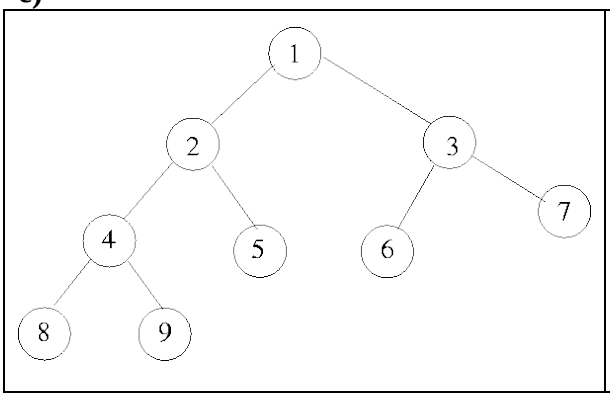
c)


2
7
5
2
6
9
5
11
4

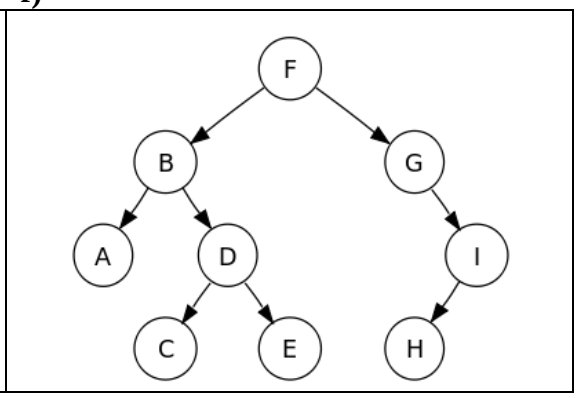
d)


8
3
10
1
6
14
4
7
13

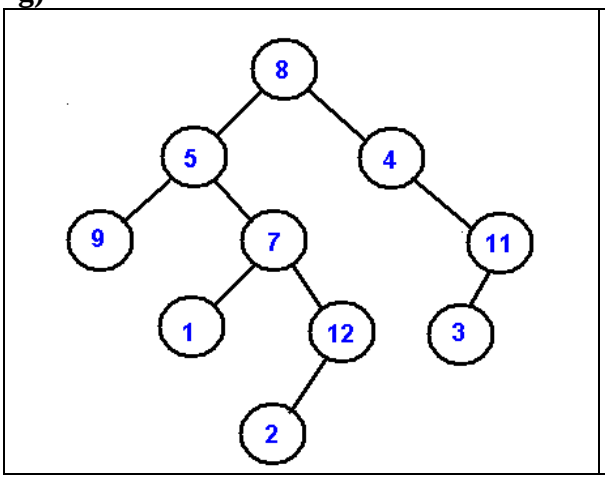
e)


1
2
3
4
5
6
7
8
9

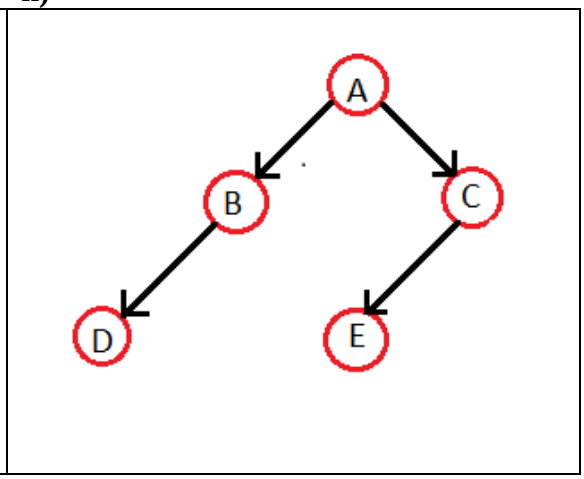
f)


F
B
G
A
D
I
C
E
H

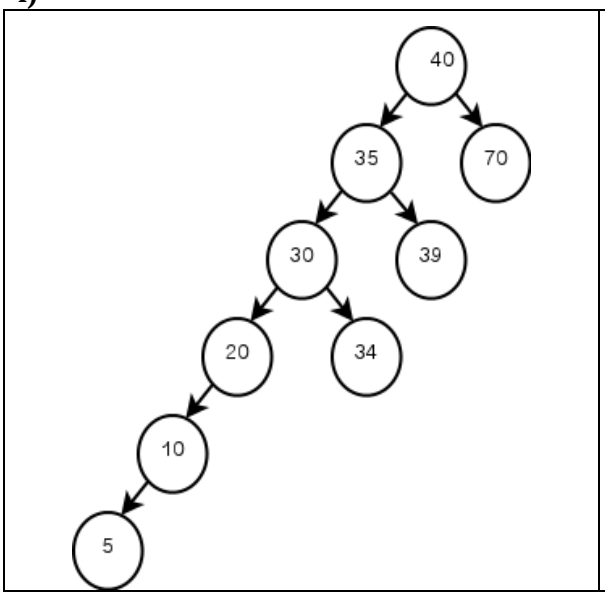
g)


8
5
4
9
7
11
1
12
3
2

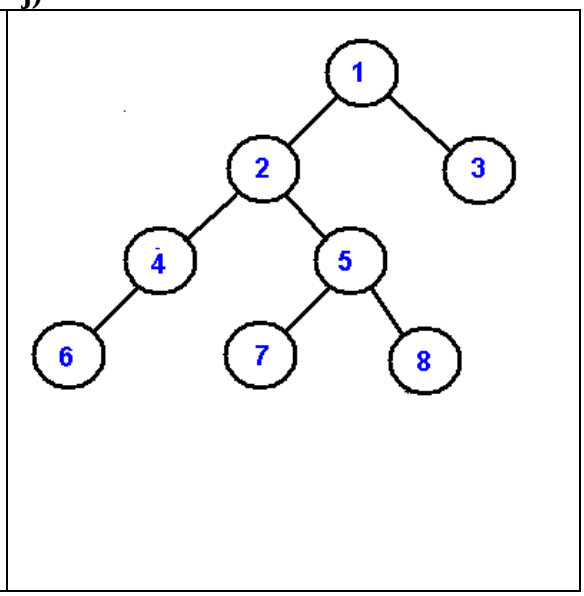
h)


A
B
C
D
E

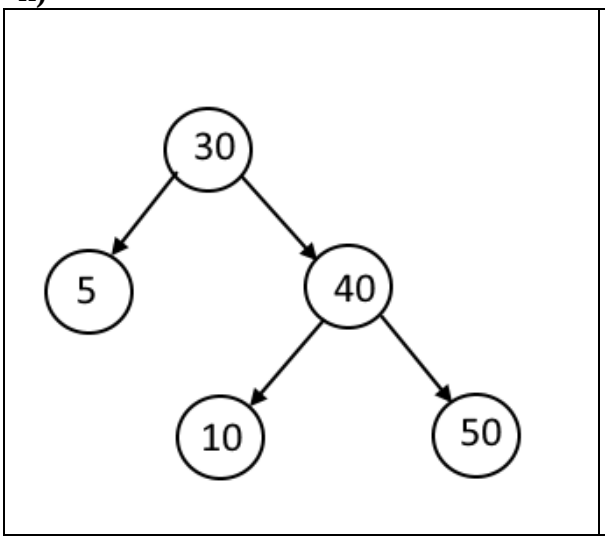
i)



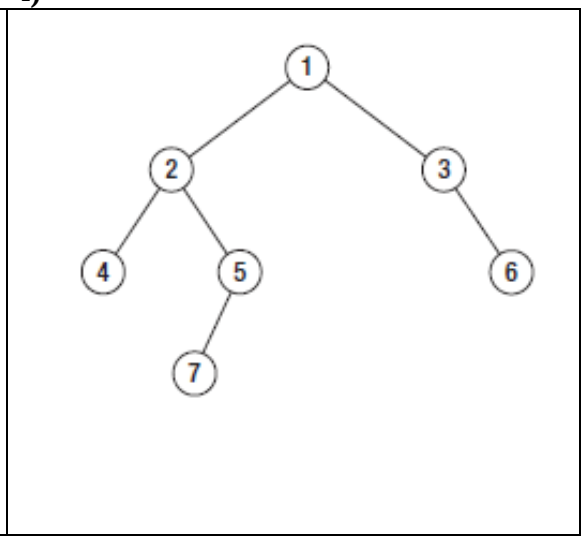
j)



k)



l)



20) [Questão 26 do Poscomp 2014] Sobre árvores binárias, considere as afirmativas a seguir.

I. Qualquer nó de uma árvore binária é raiz de, no máximo, outras duas subárvores comumente denominadas subárvore direita e subárvore esquerda.

II. Uma dada árvore binária A armazena números inteiros e nela foram inseridos 936 valores não repetidos. Para determinar se um número x está entre os elementos dessa árvore, tal número será comparado, no máximo, com 10 números contidos na árvore A.

III. Uma dada árvore binária de busca A armazena números inteiros e nela foram inseridos 936 valores não repetidos. Para determinar se um número x está entre os elementos dessa árvore, serão feitas, no máximo, 10 comparações.

IV. Uma dada árvore binária de busca A armazena números inteiros e nela foram inseridos 936 valores não repetidos. Supondo que r seja o nó raiz da árvore A e que sua subárvore esquerda contenha 460 elementos e sua subárvore direita possua 475 elementos. Para determinar se um número x pertence a essa árvore, serão feitas, no máximo, 476 comparações.

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.

b) Somente as afirmativas I e IV são corretas.

c) Somente as afirmativas III e IV são corretas.

d) Somente as afirmativas I, II e III são corretas.

e) Somente as afirmativas II, III e IV são corretas.

21) [Questão 28 do Poscomp 2009] Considere uma árvore binária de busca *T* com *n* nós e altura *h*. A altura de uma árvore é o número máximo de nós de um caminho entre a raiz e as folhas. Analise as afirmativas a seguir:

I. $h < 1 + \log_2 n$;

II. Todo nó que pertence à subárvore esquerda de um nó *x* tem valor maior que o pai de *x*.

III. Uma busca em ordem simétrica (*in-order*) em *T* produz uma ordenação crescente dos elementos de *T*.

Assinale a alternativa CORRETA:

a) Apenas a afirmativa I está correta;

b) Apenas a afirmativa II está correta;

c) Apenas a afirmativa III está correta;

d) Apenas as afirmativas I e II estão corretas;

e) Apenas as afirmativas I e III estão corretas.

22) [Questão 24 do Poscomp 2015] Considere *T* uma árvore binária cheia, em que *n*, *ne*, *ni* e *h* representam o número de nós, o número de nós externos, o número de nós internos e a altura de *T*, respectivamente. Portanto, a essa árvore *T* aplica-se a seguinte propriedade:

a) $ni = ne + 1$

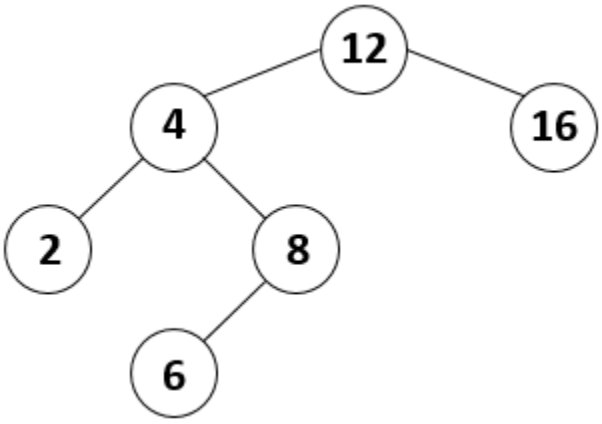
b) $h - 1 \leq ne \leq 2h$

c) $h + 1 \leq ni \leq 2h$

d) $\log(n+1) \leq h \leq n - 1$

e) $2h + 1 \leq n \leq 2h+1 - 1$

23) [Questão 23 do Poscomp 2016] Considere a árvore binária da figura a seguir:



Os resultados das consultas dos nós dessa árvore binária em pré-ordem e pós-ordem são, respectivamente:

a) (2 4 6 8 12 16) e (2 6 8 4 16 12).

b) (12 4 2 8 6 16) e (2 4 6 8 12 16).

c) (2 6 8 4 16 12) e (12 4 2 8 6 16).

d) (2 4 6 8 12 16) e (12 4 2 8 6 16).

e) (12 4 2 8 6 16) e (2 6 8 4 16 12).

24) [Questão 24 do Poscomp 2016] A operação de destruição de uma árvore requer um tipo de percurso em que a liberação de um nó é realizada apenas após todos os seus descendentes terem sido também liberados. Segundo essa descrição, a operação de destruição de uma árvore deve ser implementada utilizando o percurso

a) em ordem.

b) pré-ordem.

c) central.

d) simétrico.

e) pós-ordem.

25) [Questão 14 do ENADE 2008, para Ciência da Computação, Engenharia da Computação e Bacharelado em Sistemas de Informação] Um programador propôs um algoritmo não-recursivo para o percurso em preordem de uma árvore binária com as seguintes características.

- Cada nó da árvore binária é representado por um registro com três campos: `chave`, que armazena seu identificador; `esq` e `dir`, ponteiros para os filhos esquerdo e direito, respectivamente.
- O algoritmo deve ser invocado inicialmente tomando o ponteiro para o nó raiz da árvore binária como argumento.
- O algoritmo utiliza `push()` e `pop()` como funções auxiliares de empilhamento e desempilhamento de ponteiros para nós de árvore binária, respectivamente.

A seguir, está apresentado o algoritmo proposto, em que λ representa o ponteiro nulo.

```
Procedimento preordem (ptraiz : PtrNoArvBin)
   Var ptr : PtrNoArvBin;
   ptr := ptraiz;
   Enquanto (ptr ≠ λ) Faça
      escreva (ptr↑.chave);
      Se (ptr↑.dir ≠ λ) Então
         push(ptr↑.dir);
      Se (ptr↑.esq ≠ λ) Então
         push(ptr↑.esq);
      ptr := pop();
   Fim_Enquanto
Fim_Procedimento
```

Com base nessas informações e supondo que a raiz de uma árvore binária com *n* nós seja passada ao procedimento `preordem()`, julgue os itens seguintes.

I O algoritmo visita cada nó da árvore binária exatamente uma vez ao longo do percurso.

II O algoritmo só funcionará corretamente se o procedimento `pop()` for projetado de forma a retornar λ caso a pilha esteja vazia.

III Empilhar e desempilhar ponteiros para nós da árvore são operações que podem ser implementadas com custo constante.

IV A complexidade do pior caso para o procedimento `preordem()` é $O(n)$.

Assinale a opção correta.

a) Apenas um item está certo.

b) Apenas os itens I e IV estão certos.

c) Apenas os itens I, II e III estão certos.

d) Apenas os itens II, III e IV estão certos.

e) Todos os itens estão certos.

Heaps

26) Desenhe o Max-Heap referente às entradas de dados de a) até f). Utilize uma lógica própria para montar cada heap e considere que os valores informados foram armazenados no vetor na ordem indicada no enunciado.

a) 53, 90, 30, 44, 14, 7, 68

b) 2, 7, 26, 25, 19, 17, 1, 90, 3, 36

c) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

d) 91, 81, 65, 59, 27, 25, 10, 56, 16, 76, 62, 90, 42, 69, 47, 41

e) 88, 87, 84, 82, 65, 11, 93, 97, 55, 20, 86, 14, 45, 51, 3, 95, 12, 4, 24

f) 8, 3, 10, 14, 6, 4, 13, 7, 1

27) Construa novos Max-Heaps para as entradas de dado do exercício anterior, agora utilizando o pseudocódigo apresentado a seguir.

Executar então os passos a seguir para cada elemento do vetor que contém a árvore binária completa, começando do primeiro:

Faça M ser o índice do elemento a ser processado (considere sempre os índices começando em 1)

Seja V o vetor que contém a árvore completa (com elementos de 1 a N)

Faça F = M + 1

Enquanto (F maior que 1 E V[F / 2] menor que V[F]) FAÇA

Troque os elementos V[F / 2] e V[F] de lugar

Corte o valor de F pela metade

FimFAÇA

28) Desenhe um Min-Heap para cada entrada de dados do exercício 1.

29) **[Questão 26 do Poscomp 2005]** Considere um *heap H* com 24 elementos tendo seu maior elemento na raiz. Em quantos nós de *H* pode estar o seu segundo **menor** elemento?

a) 18

b) 15

c) 14

d) 13

e) 12

Pesquisa e ordenação

30) [Questão 22 do Poscomp 2015] Quais destes algoritmos de ordenação têm a classe de complexidade assintótica, no pior caso, em $O(n.\log n)$?

a) QuickSort, MergeSort, e HeapSort

b) QuickSort e SelectionSort

c) MergeSort e HeapSort

d) QuickSort e BubbleSort

e) QuickSort, MergeSort e SelectionSort

31) [Questão 36 do Poscomp 2013] Sobre a escolha adequada para um algoritmo de ordenação, considere as afirmativas a seguir.

I. Quando os cenários de pior caso for a preocupação, o algoritmo ideal é o Heap Sort.

II. Quando o vetor apresenta a maioria dos elementos ordenados, o algoritmo ideal é o Insertion Sort.

III. Quando o interesse for um bom resultado para o médio caso, o algoritmo ideal é o Quick Sort.

IV. Quando o interesse é o melhor caso e o pior caso de mesma complexidade, o algoritmo ideal é o Bubble Sort.

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.

b) Somente as afirmativas I e IV são corretas.

c) Somente as afirmativas III e IV são corretas.

d) Somente as afirmativas I, II e III são corretas.

e) Somente as afirmativas II, III e IV são corretas.

Hashing

32) [Questão 31 do Poscomp 2009] Considere uma tabela de espalhamento (tabela *hash*) de comprimento $m = 11$, que usa endereçamento aberto (*open addressing*), a técnica de tentativa linear (*linear probing*) para resolver colisões e com a função de dispersão (função *hash*) h(k) = k mod m, onde k é a chave a ser inserida. Considere as seguintes operações sobre essa tabela:

- Inserção das chaves 3, 14, 15, 92, 65, 35 (nesta ordem);
- Remoção da chave 15; e
- Inserção da chave 43.

Escolha a opção que representa esta tabela após estas operações:

a) 65 – ø – 35 – 14 – ø – 92 – 3 – ø – ø – ø – 43

b) 43 – ø – 35 – 3 – 14 – 92 – ø – ø – ø – ø – 65

c) 65 – ø – 35 – X - 14 – 92 – 3 – ø – ø – ø – 43

d) 65 – ø – 35 – 3 – 14 – 92 – ø – ø – ø – ø – 43

e) 43 – ø – 35 – 3 – 14 – X – 92 – ø – ø – ø – 65

33) [Questão 20 (discursiva) do ENADE 2008, para Ciência da Computação, Engenharia da Computação e Bacharelado em Sistemas de Informação] Tabelas de dispersão (tabelas *hash*) armazenam elementos com base no valor absoluto de suas chaves e em técnicas de tratamento de colisões. As funções de dispersão transformam chaves em endereços-base da tabela, ao passo que o tratamento de colisões resolve conflitos em casos em que mais de uma chave é mapeada para um mesmo endereço-base da tabela.

Suponha que uma aplicação utilize uma tabela de dispersão com 23 endereços-base (índices de 0 a 22) e empregue $h(x) = x$ mod 23 como função de dispersão, em que x representa a chave do elemento cujo endereço-base deseja-se computar. Inicialmente, essa tabela de dispersão encontra-se vazia. Em seguida, a aplicação solicita uma sequência de inserções de elementos cujas chaves aparecem na seguinte ordem: 44, 46, 49, 70, 27, 71, 90, 97, 95.

Com relação à aplicação descrita, faça o que se pede a seguir.

a) Escreva, no espaço reservado, o conjunto das chaves envolvidas em colisões.

b) Assuma que a tabela de dispersão trate colisões por meio de encadeamento exterior. Esboce a tabela de dispersão para mostrar seu conteúdo após a sequência de inserções referida.

34) [Questão 23 do Poscomp 2011] Ao usar o cálculo de endereço ou *hashing*, geralmente é necessário o uso de um método de tratamento de colisões. Sobre esse método, é correto afirmar:

a) O tratamento de colisões é necessário apenas quando a tabela está cheia e se necessita inserir mais uma chave.
b) O tratamento de colisões é necessário para determinar o local da chave no momento da inserção na tabela.
c) O tratamento de colisões é necessário quando a tabela está vazia, pois não é possível calcular o endereço diretamente nesse caso.
d) O tratamento de colisões é necessário quando a chave inserida ainda não existir na tabela de endereçamento.
e) O tratamento de colisões é necessário, pois o *hashing* gera repetição de endereço para diferentes chaves.