UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ – ARA
Estruturas de Dados e Algoritmos
SEMESTRE 2019.1
Avaliação 3 – Fila A – 27.06.2019
Prof. Cristian Cechinel

Nome: _____

Toda a avaliação deve ser desenvolvida na linguagem JAVA.

1. [2,5 pontos] Desenvolva um método que retorne a quantidade de elementos armazenados em uma árvore binaria de busca.

2. [2,5 pontos] Considere as seguintes operações executadas sequencialmente em uma árvore AVL que está inicialmente nula. Após cada operação realize o cálculo do fator de balanceamento de cada nó (fator h) e informe se a árvore está balanceada ou desbalanceada. Em caso de desbalanceamento, realize as rotações necessárias para que a árvore fique balanceada identificando quais são essas rotações, e recalcule o fator de balanceamento (fator h). Considere que a árvore está inicialmente vazia

Obs1. Essa questão somente será considerada correta se todas as operações realizadas estiverem corretas, incluindo aqui os valores dos fatores de balanceamento (fator h) e as rotações realizadas.

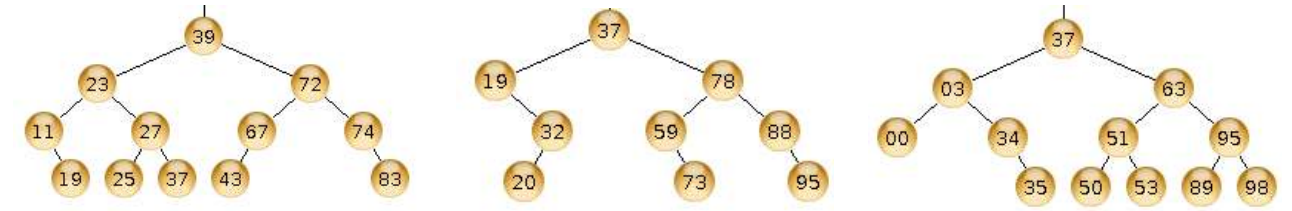
Obs2. Na resolução, identifique de maneira clara qual a operação que está sendo realizada e o fator de balanceamento dos nodos. As ações devem ser executadas sequencialmente (letra a, b, c, d …)

a) Inserir 20	b) inserir 19	c) inserir 25	d) inserir 26
e) inserir 29	f) retirar 19	g) inserir 23	h) excluir 29

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ – ARA
Estruturas de Dados e Algoritmos
SEMESTRE 2019.1
Avaliação 3 – Fila A – 27.06.2019
Prof. Cristian Cechinel

3. [2,5 pontos] Considere as árvores binárias de busca abaixo e imprima os elementos das mesmas de acordo com cada um dos seguintes tipos de percurso: pré-ordem, simétrico e pós-ordem. Identifique a árvore em questão e o percurso que está sendo impresso.

3.a)	3.b)	3.c)



4. [2,5 pontos] Considere o código a seguir referente a uma busca em largura em um determinado grafo. Marque V (Verdadeiro) ou F (Falso) em cada uma das alternativas.

```
24 public void busca(int o){
25     this.origem = o;
26     FilaEncadeada fila = new FilaEncadeada();
27     for (int i = 0; i < grafo.getV(); i++){
28         this.cor[i] = BRANCO;
29         this.distancia[i] = 0;
30         this.predecessor[i] = -1;
31     }
32     fila.insereFinal(origem);
33     while (!fila.vazia()){
34         int atual = fila.retiraInicio();
35         for (int i = 0; i < grafo.getV(); i++){
36             if (grafo.isAdjacente(atual, i)){
37                 if (cor[i] == BRANCO){
38                     cor[i] = CINZA;
39                     predecessor[i] = atual;
40                     distancia [i] = distancia[i] + 1;
41                     fila.insereFinal(atual);
42                 }
43             }
44         }
45         cor[atual] = PRETO;
46     }
47 }
```

( ) A busca em largura deveria utilizar uma pilha para realizar o controle da busca nos vértices, e não uma fila encadeada como está sendo feito no código.

( ) O código possui um erro na linha 40, sendo que o correto seria:

distancia[i] = distancia [atual] + 1

( ) O código possui um erro na linha 39, sendo que o correto seria:

predecessor[atual] = i

( ) O código possui um erro na linha 41, sendo que o correto seria:

fila.insereFinal(atual + i)

( ) Ao final da execução do algoritmo, o vetor **cor** terá todas as suas posições preenchidas com o valor PRETO.

( ) A busca em largura está sendo realizada a partir de um vértice de origem que é enviado por parâmetro e que é o primeiro vértice a ser inserido na fila encadeada.

Obs. Cada alternativa preenchida com resposta ERRADA eliminará uma preenchida com resposta CORRETA.