



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e Informática
3ª Avaliação

Curso : Engenharia de Software
Disciplina : Algoritmos e Estruturas de Dados II
Professora : Eveline Alonso Veloso
Valor : 20 pontos
Data : 24/06/2024

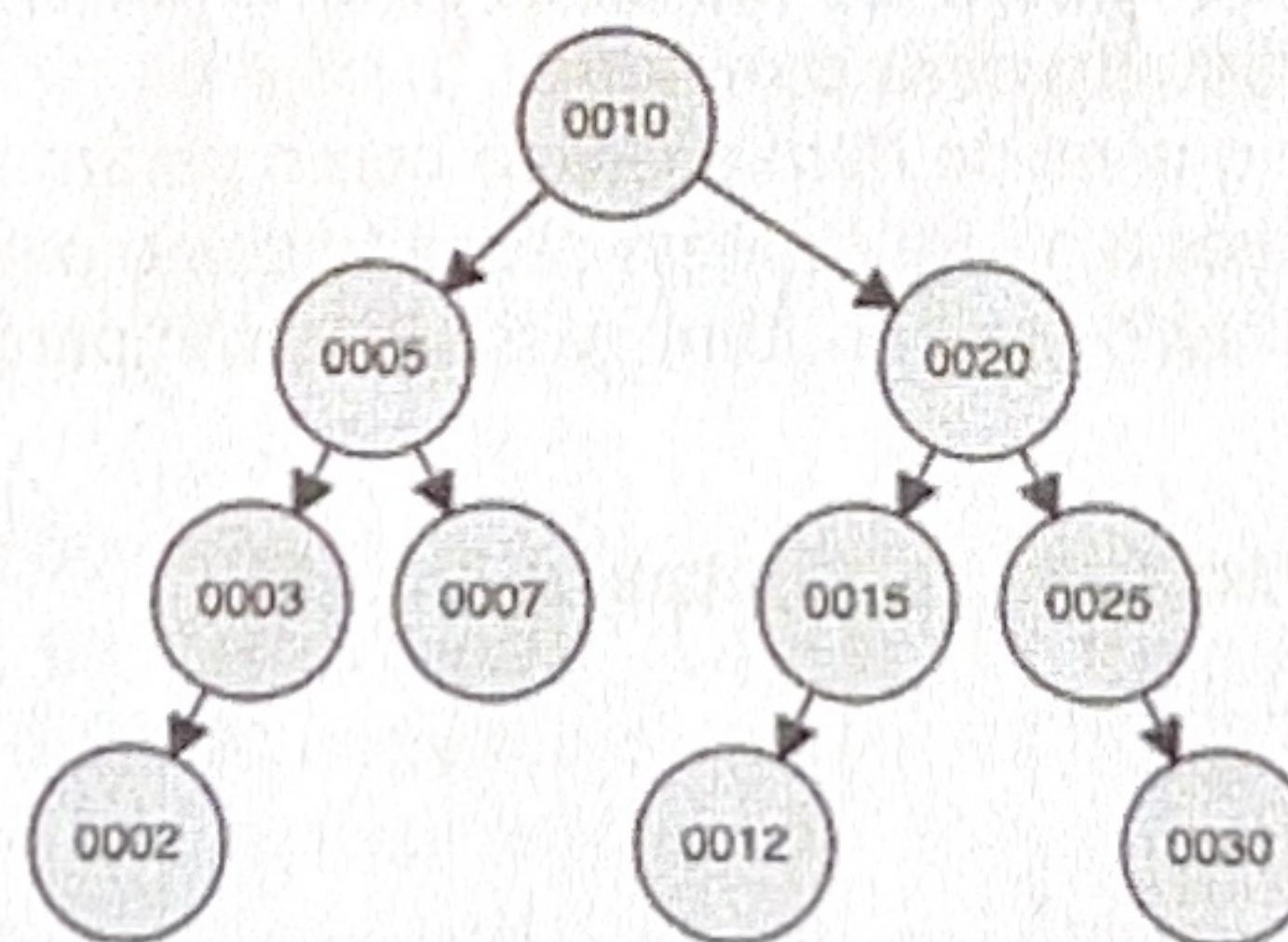
Leia atentamente as instruções abaixo antes de iniciar a resolução da prova.

Instruções:

- O celular (e qualquer outro dispositivo eletrônico pessoal) deve ficar desligado e guardado, na mochila ou bolsa, durante toda a prova.
- A prova é individual e sem consulta.
- Não é permitido utilizar nenhuma estrutura de dados/método da linguagem. Utilize apenas o que foi fornecido pela professora.
- Leia atentamente cada enunciado. Respostas que não corresponderem ao que foi solicitado ou que não seguirem o especificado no enunciado da questão não serão consideradas.

Nome: Luca Ferrari Agalim Nota: 16,0

- 1) (2 pontos) Considere a árvore binária de busca abaixo:



Marque a alternativa que apresenta as chaves impressas em um percurso em pós-ordem, após a remoção dos nós de chaves 10, 15 e 20, nessa ordem.

- a) 2 – 3 – 5 – 30 – 25 – 12 – 7
b) 2 – 3 – 7 – 12 – 30 – 25 – 5
c) 3 – 5 – 7 – 30 – 12 – 2 – 25
d) 7 – 3 – 30 – 25 – 12 – 5 – 2
e) 7 – 5 – 3 – 2 – 12 – 25 – 30

2) (2 pontos) Considere o tipo abstrato de dados árvore binária de busca e que cada um de seus nós possui três atributos.

- "item": elemento armazenado no nó;
- "esquerda": referência ao nó à sua esquerda;
- "direita": referência ao nó à sua direita.

Considere ainda uma árvore binária de busca pré-existente cuja raiz é referenciada pela variável "root".

```
private int doSomething(No<E> raizArvore, E item) {  
  
    if (raizArvore == null)  
        throw new NoSuchElementException("O item não foi localizado na  
árvore!");  
    else if (raizArvore.getItem().equals(item))  
        return 0;  
    else if (item.compareTo(raizArvore.getItem()) < 0)  
        return 1 + doSomething(raizArvore.getEsquerda(), item);  
    else  
        return 1 + doSomething(raizArvore.getDireita(), item);  
}
```

Analizando o código apresentado, indique qual é a tarefa realizada por ele:

- a) contar e retornar a quantidade de nós da árvore, a partir do item passado como parâmetro para o método.
- b) contar e retornar o número de folhas da árvore binária de busca, a partir do item passado como parâmetro para o método.
- c) calcular e retornar o nível do item passado como parâmetro para o método.
- d) determinar e retornar a altura do item passado como parâmetro para o método.
- e) determinar e retornar o grau do item passado como parâmetro para o método.

Indique na tabela abaixo suas respostas, à caneta:

1	2
A ✓	C ✓

3) (2 pontos) Considere a árvore AVL cujas chaves, impressas em um caminhamento em pré-ordem, correspondem a: 27 – 24 – 22 – 26 – 30 – 35.

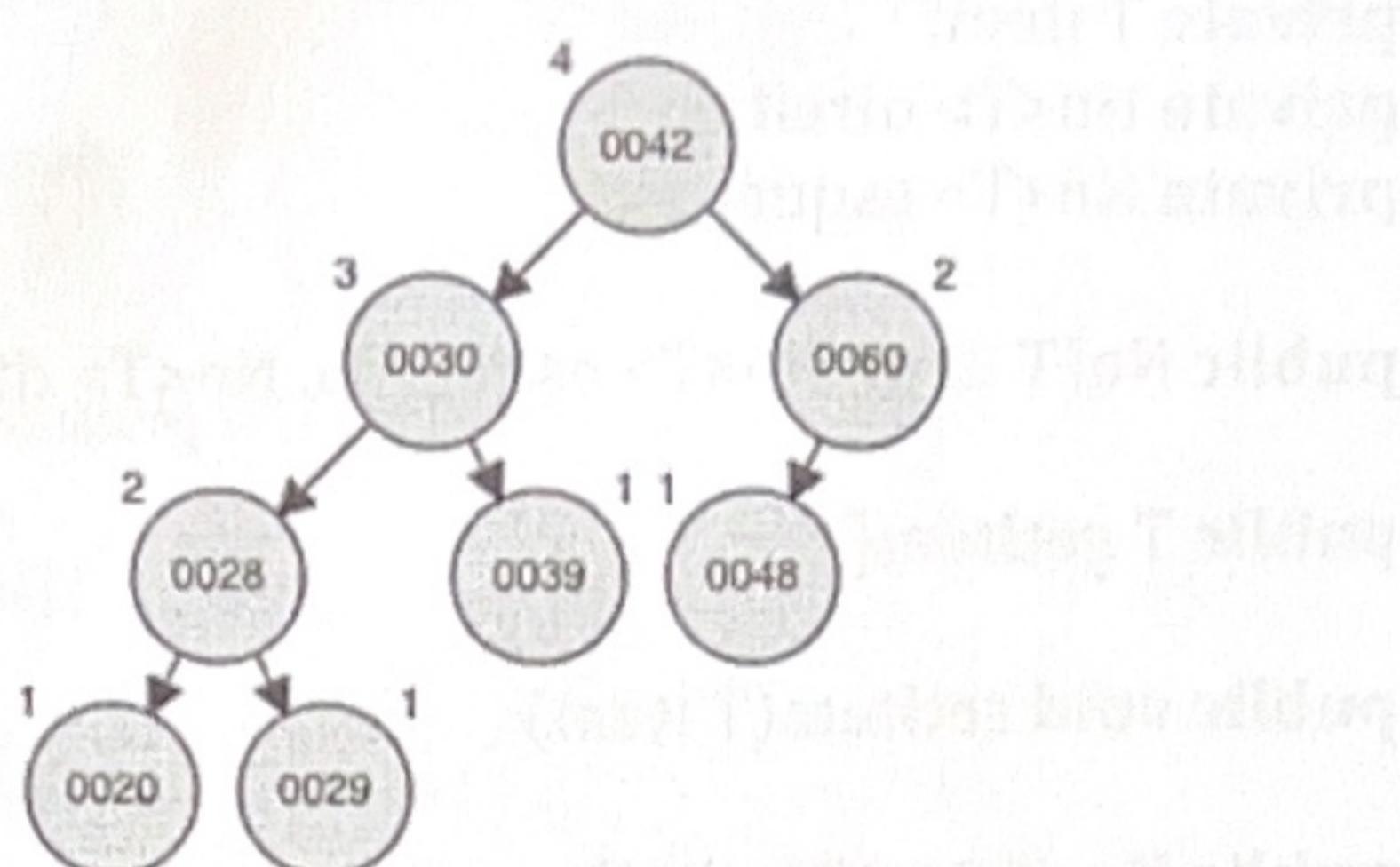
Indique seu percurso em pós-ordem após a inserção de um nó contendo a chave 37.

80

~~24, 22, 26, 35, 30, 37, 27~~

22, 26, 24, 30, 37, 35, 27

- 4) (2 pontos) Considere a árvore AVL abaixo:



Após a remoção dos elementos de chaves 20 e 39, nessa ordem, e a realização das operações de balanceamento, se necessárias; podemos afirmar que, realizando o percurso na árvore em pré-ordem, a sequência de visita dos nós será:

20

42, 29, 28, 30, 60, 48 ✓

- 5) (2 pontos) Considere uma tabela de dispersão de tamanho $M = 13$, inicialmente vazia, que usa endereçamento aberto, com *rehash linear*, para resolver possíveis colisões; e função de dispersão $h(k) = k \bmod M$, onde "k" é a chave a ser inserida nessa tabela de dispersão. Suponha que foram inseridas as seguintes chaves nessa tabela de dispersão, nessa ordem: 50, 39, 18, 81, 72, 19, 40, 11, 3, 67, 32, 90 e 7. O conteúdo dessa tabela de dispersão, a partir da posição 0, após essas inserções será:

20

39, 40, 67, 81, 3, 18, 19, 72, 32, 90, 7, 50, 11 ✓

- 6) (6 pontos) Implemente, em Java, na classe `ABB<E extends Comparable<E>>`, especificada abaixo e desenvolvida durante as aulas teóricas da disciplina, o método `public E obterSucessor(E item)`, capaz de recuperar e retornar o menor elemento armazenado na árvore binária de busca que seja maior do que o item informado como parâmetro para esse método. A determinação dos elementos da árvore binária de busca que são maiores, menores ou iguais ao item passado como parâmetro para o método deve basear-se no(s) critério(s) empregado(s) na implementação do método `public int compareTo(E outroItem)` do item armazenado na ABB. Se não for encontrado, na árvore binária de busca, nenhum elemento correspondente ao passado como parâmetro para esse método, ou o item informado como parâmetro não apresentar sucessor na árvore binária de busca, uma exceção deve ser lançada.

Utilize **obrigatoriamente** as classes especificadas abaixo:

```
public class ABB<E extends Comparable<E>> {  
    private No<E> raiz;  
}
```

```

public class No<T extends Comparable<T>> {

    private T item;
    private No<T> direita;
    private No<T> esquerda;

    public No(T item, No<T> esquerda, No<T> direita);

    public T getItem();

    public void setItem(T item);

    public No<T> getDireita();

    public void setDireita(No<T> direita);

    public No<T> getEsquerda();

    public void setEsquerda(No<T> esquerda);
}

```

- 7) (4 pontos) Implemente, em Java, na classe `TabelaHash<K, V>`, especificada abaixo e desenvolvida durante as aulas da disciplina, e que representa uma tabela *hash* que trata possíveis colisões por meio de listas encadeadas, o método `public void substituir(K chave, V novoValor)`, que substitui o valor associado à chave passada como parâmetro para o valor especificado no parâmetro `novoValor`.

Utilize **obrigatoriamente** as classes especificadas abaixo:

```

public class TabelaHash<K, V> {

    private Lista<Entrada<K, V>>[] tabelaHash;

    private int capacidade;

    private int funcaoHash(K chave);
}

```

```

public class Lista<E> {

    private Celula<E> primeiro;
    private Celula<E> ultimo;
    private int tamanho;

    public E pesquisar(E procurado);
}

```

```
}
```

```
public class Entrada<K, V> {  
  
    private final K chave;  
    private V valor;  
  
    public Entrada(K chave, V item);  
  
    public K getChave();  
  
    public V getValor();  
  
    public void setValor(V valor);  
  
    @Override  
    public boolean equals(Object outroObjeto);  
  
    @Override  
    public int hashCode();  
}
```

Questão 6:

```
public E obterSucessor(E item){  
    No<E> achado = procurar(item);  
    if(achado.getDireita() == null) throw NoSuch...();  
    return obterMenor(achado.getDireita());  
}  
  
private E obterMenor(No<E> raizSub){  
    if(raizSub.getEsquerda() == null){  
        return raizSub.getItem();  
    }  
    return obterMenor(raizSub.getEsquerda());  
}  
  
private No<E> procurar(E item, No<E> raizSub){  
    if(raizSub == null) throw NoSuchElementException();  
    int comp = item.compareTo(raizSub.getItem());  
    if(comp == 0) return raizSub;  
    if(comp > 0) return procurar(item, raizSub.getDireita());  
    else return procurar(item, raizSub.getEsquerda());  
}
```

Questão 7:

```
public void substituir (K chave, V novoValor) {  
    int hash = funcaoHash (chave);  
    Lista<Entrada<K,V>> lista = tabelaHash[hash];  
    if (lista == null)  
        throw new NoSuchElementException();  
    Entrada<K,V> entrada = lista.pesquisar (chave,  
        new Entrada (chave, null));  
    entrada.setValor (novoValor);  
}
```