UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

ESCOLA DE INFORMÁTICA APLICADA

Lista 1 - EDA

DIOGO ARRUDA

1ª Questão:

a) A função calcula a altura(h) da árvore, retornando -1 caso seja um nó nulo e o maior valor entre a altura da subárvore esquerda e direita + 1

b) Complexidade O(n), pois percorre todos os nós da árvore. Não é possível diminuir a complexidade porque o método tem que passar por todos os nós para fazer o cálculo da altura.

2ª Questão:

a) Retorna, se existir, o nó sucessor daquele nó que chamar a função. Caso não exista, retorna null.

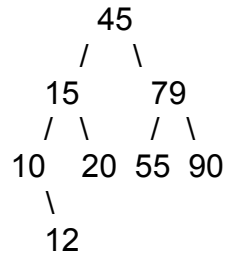
b) O(h).

c) Para achar o nó sucessor, caso geralmente necessário para remoções em árvores que são ordenadas.

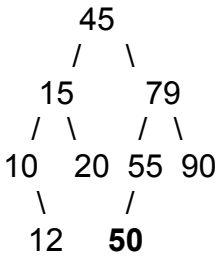
3ª Questão: c. 925, 202, 911, 240, 912, 245, 363. Após passar pelo 911, ele vai pra esquerda, onde contém os nós que são menores que ele. Mas, após passar pelo 240, ele chega no 912, nó maior que 911, então deveria estar a direita do 911.

4ª Questão: Passo a passo das inserções e remoções na árvore (Explicação dos casos no final)

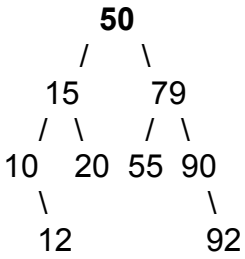
Árvore Original



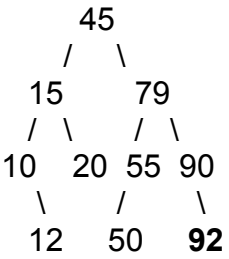
Inserção do 50 (Sem alterações)



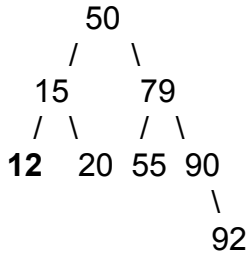
Remoção do 45 (Caso 3)

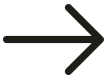


Inserção do 92 (Sem alterações)

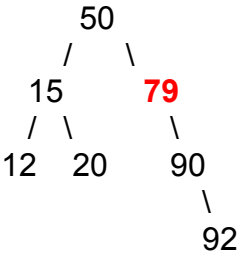


Remoção do 10 (Caso 2)

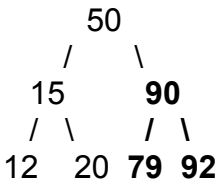




Remoção do 55(Caso 1)
Desbalanceamento



Rotação para a Esquerda



(Árvore final Balanceada)

Casos:

1° Caso: Nó folha (zero filho) → Remoção sem alterações;
2° Caso: Nó com apenas um filho → Nó filho vira filho do antigo avô
3° Caso: Nó com dois filhos → Busca o sucessor.

5ª Questão:

```java
public T ancestral(int chave1, int chave2){
    return ancestral(raiz, chave1, chave2).valor;
}

/**
 * Método privado que busca recursivamente o nó ancestral em comum entre dois nós.
 *
 * @param no nó atual
 * @param chave1 chave do primeiro nó
 * @param chave2 chave do segundo nó
 *
 * @return O {@code no} ancestral entre dos nós
 */

private No<T> ancestral(No<T> no, int chave1, int chave2){

    if (no == null)
        return null;

    if (chave1 < no.chave && chave2 < no.chave)
        return ancestral(no.esq, chave1, chave2);

    if (chave1 > no.chave && chave2 > no.chave)
        return ancestral(no.dir, chave1, chave2);

    return no;

}
```

6ª Questão:

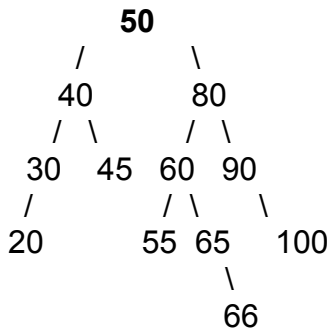
Fórmula Recursiva → $N(h) = N(h-1) + N(h-2) + 1$, onde N = n° de nós.

```java
public static int alturaMinima(int h){

    if(h == 0)
        return 0;
    if(h == 1)
        return 1;
    if(h == 2)
        return 2;

    return (alturaMinima(h - 1) +
            alturaMinima(h - 2) + 1);
}
```

```
Quantidade minima de nós:
N(1) = 1
N(2) = 2
N(3) = 4
N(4) = 7
N(5) = 12
N(6) = 20
N(7) = 33
N(8) = 54
N(9) = 88
N(10) = 143
```

Criei um método pra me ajudar. Então, será necessária uma árvore de 12 nós.

Árvore AVL com H = 5 e com o mínimo de nós:



7ª Questão:

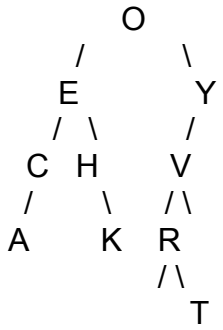
Com base no resultado do método.

a) h = 4.
b) h = 6.
c) h = 8.
d) h = 8.

8ª Questão:

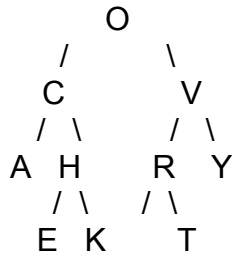
---

a)



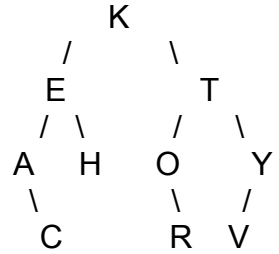
---

b)

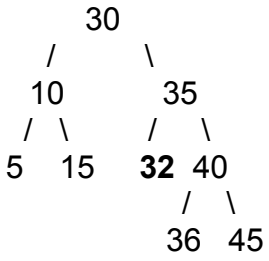


---

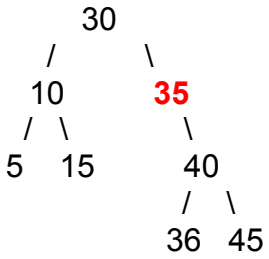
c) Ordem: K, E, T, A, H, O, Y, C, R, V



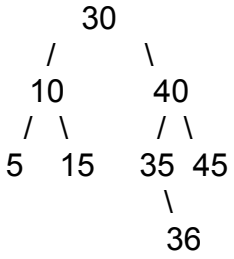
9ª Questão: Sim, ex:



Essa árvore está balanceada. Com a remoção do **32**, a subárvore do nó 35 estará com o fator de balanceamento em -2, o que significa que a sua subárvore direita está “pesada”, mas, seu filho direito está com o fator balanceamento em 0, já que possui apenas um filho esquerdo e um direito.



Para solucionar o problema, basta uma rotação simples para a Esquerda:



10ª Questão: Todos os métodos explicados em código.

1° Passo, criação do método estático público.

```java
/**
 * Método estático que transforma uma árvore BST com atributos de AVL em uma árvore AVL
 *
 * @param raiz raiz de uma árvore
 *
 * @return uma árvore AVL
 */

public static <T> ArvAVL<T> converteEmAVL(ArvAVL<T> raiz){

    return converte(raiz);
}
```

Complexidade: insere n-vezes um nó. Complexidade da inserção é log(n), então a complexidade final fica **O(n log(n))**

2° Passo, criação do método estático privado que transforma a árvore em uma lista ordenada, e depois insere numa nova árvore AVL, que já tem seus métodos de balanceamento.

```java
/**
 * Método estático privado que faz a conversão da BST em AVL
 *
 * @param arvore árvore a ser convertida
 *
 * @return uma árvore AVL
 */

private static <T> ArvAVL<T> converte(ArvAVL<T> arvore){

    List<No<T>> listaNos = criaListaOrdenada(arvore.raiz);
    ArvAVL<T> novaArvore = new ArvAVL<>();

    for (No<T> no : listaNos)
        novaArvore.insere(no.chave, no.valor);

    return novaArvore;
}
```

Complexidade: n log(n)

Passo Extra: Criação do método que percorre a árvore e salva os valores ordenadamente em uma lista. Pra não ficar tudo no mesmo método.

```java
/**
 * Método estático recursivo que transforma a BST em uma lista ordenada
 *
 * @param no raiz da árvore
 *
 * @return lista ordenada de nós.
 */

private static <T> List<No<T>> criaListaOrdenada(No<T> no){

    List<No<T>> lista = new ArrayList<>();

    if (no != null)
    {
        lista.addAll(criaListaOrdenada(no.esq));
        lista.add(no);
        lista.addAll(criaListaOrdenada(no.dir));
    }

    return lista;
}
```

Complexidade: O(n)