TRABALHO 2 DE ALGORITMOS E ESTRUTURAS DE DADOS

ARVORES AVL

NOMES: MARCOS RAACH, PEDRO E LUCAS (COLOQUEM O SOBRENOME DE VOCES)

NOVEMBRO 2023

1. Funcionamento da Arvore AVL:

2. Algoritmo de Inserção:
    a. Na linha 10 está a declaração do método, onde é necessário passar um inteiro como parâmetro para inicialização do próprio.
    b. Como apresentado no código abaixo, entre as linhas 11 a 15, ocorre a inicialização de três nodos, o nodo node, o current e o prev e a aplicação do inteiro solicitado antes como o elemento do nodo node.
    c. Entre as linhas 16 e 37 existem algumas condições que mudam o trajeto que o código vai percorrer:
        i. Se a arvore não tiver raiz, o nodo node se torna a raiz.
        ii. Se tiver, o nodo current, recebe as informações da raiz:
            1. O elemento passado como parâmetro for menor ou igual ao elemento que está em current.
                a. O current recebe o elemento que estiver a sua esquerda.
                b. E se o current for igual a nulo, o elemento a esquerda de prev, recebe o valor de node.
            2. A mesma ideia da anterior apenas quando o parâmetro for maior ou igual ao elemento em current

```java
public void add(Integer element){ // O(n)
    Node prev, current;
    Node node = new Node();
    node.element = element;
    node.right = null;
    node.left = null;
    if (root == null) {
        root = node;
    } else {
        current = root;
        while(true) {
            prev = current;
            if (element ⩽ current.element) {
                current = current.left;
                if (current == null) {
                    prev.left = node;
                    return;
                }
            }
            else {
                current = current.right;
                if (current == null) {
                    prev.right = node;
                    return;
                }
            }
        }
    }
}
```

3. Algoritmo de balanceamento
    a. Nosso algoritmo de balanceamento foi divido em 7 métodos para melhor entendimento do que está acontecendo em cada parte.
    b. Colocamos apenas um método como público, que pega a raiz da arvore e chama o método que balanceia o nodo e seus elementos a esquerda e direita. Após, chama o outro método de balanceamento, onde existem as rotações.

```java
public void balanceTree() {      You, há 1 segund
    root = balanceTree(root);
}

private Node balanceTree(Node node) {
    if (node == null) {
        return null;
    }
    node.left = balanceTree(node.left);
    node.right = balanceTree(node.right);
    return balance(node);
}
```

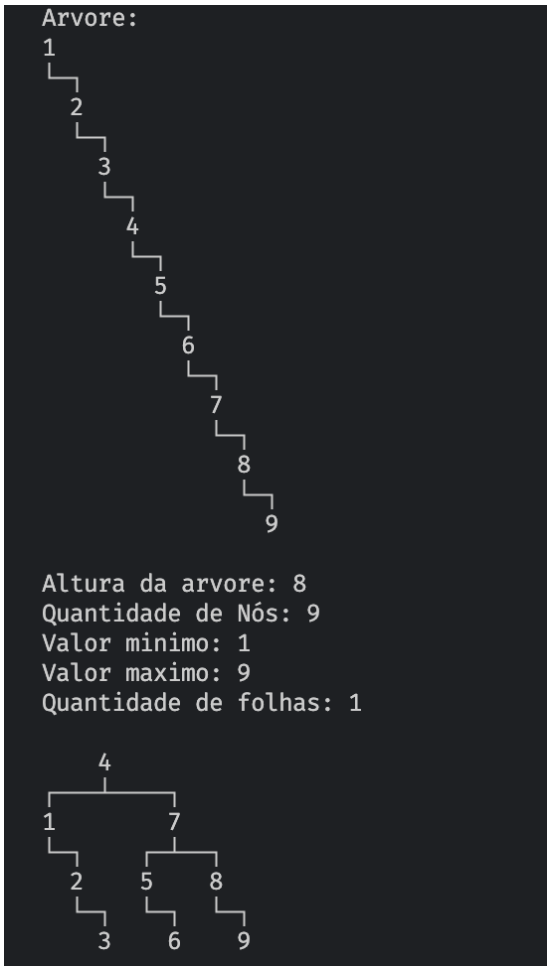
    c. Ocorre primeiro uma validação, caso a raiz seja nula, retorna nulo também, mas caso não aconteça, é utilizado o método updateHeight com o nodo que foi passado como parâmetro e ocorre a checagem do balanceamento do nodo.

i. Se o balanceamento for maior que 1 e 0 balanceamento do nodo a esquerda do nodo passado for maior ou igual a 0, então ocorre a rotação a direita do nodo passado
ii. Se o balanceamento for maior que -1 e 0 balanceamento do nodo a esquerda do nodo passado for menor ou igual a 0, então ocorre a rotação a esquerda do nodo passado.
iii. Se o balanceamento for maior que 1 e 0 balanceamento do nodo a esquerda do nodo passado for menor que 0, então ocorre a rotação a direita do nodo passado
iv. Se o balanceamento for maior que -1 e 0 balanceamento do nodo a esquerda do nodo passado for maior que 0, então ocorre a rotação a esquerda do nodo passado.

d. Método updateHeight:
    i. Se o nodo não for nulo, ele atualiza a altura do nodo com a soma de 1 + o maior valor entre o seu nodo da esquerda ou direita
e. Método getBalance:
    i. Se o nodo não for nulo, ele faz a altura de seu nodo da esquerda menos o da direita.

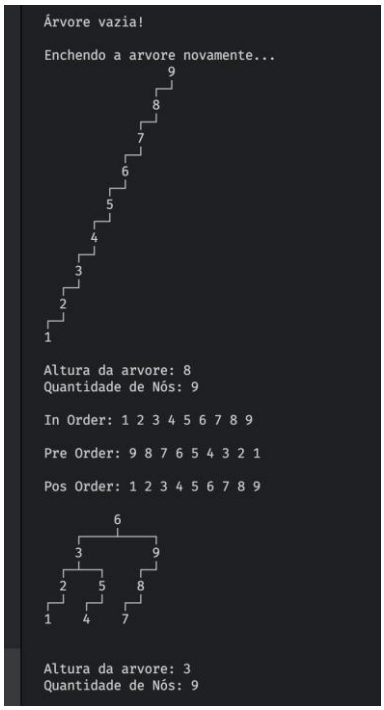
```java
private Node balance(Node node) {
    if (node == null) {
        return null;
    }
    updateHeight(node);
    int balance = getBalance(node);


    if (balance > 1 && getBalance(node.left) >= 0) {
        return rotateRight(node);
    }
    if (balance < -1 && getBalance(node.right) <= 0) {
        return rotateLeft(node);
    }
    if (balance > 1 && getBalance(node.left) < 0) {
        node.left = rotateLeft(node.left);
        return rotateRight(node);
    }
    if (balance < -1 && getBalance(node.right) > 0) {
        node.right = rotateRight(node.right);
        return rotateLeft(node);
    }      You, há 3 dias • feat: avl tree implementation

    return node;
}

private void updateHeight(Node node) {
    if (node != null) {
        node.height = 1 + Math.max(height(node.left), height(node.right));
    }
}

private int getBalance(Node node) {
    if (node == null) {
        return 0;
    }
    return height(node.left) - height(node.right);
}
```

4. Funcionamento do algoritmo funcionando:
    a. Inicializada a arvore com o método add(element)
        i. Com números de 1 a 9
    b. Mostrado no terminal:
        i. Altura da arvore
        ii. Quantidade de nodos
        iii. Valor mínimo dos números
        iv. Valor máximo dos números
        v. Quantidade de nodos que são folhas (não tem filhos)
    c. Ocorre o balanceamento da arvore e após isso, a arvore é mostrada no terminal, já totalmente balanceada.



    d. Remove todos os nodos da arvore.
    e. Adiciona todos os nodos na arvore, do número 9 ao número 1.
    f. Mostra todas as informações da arvore novamente
    g. E mostra os 3 caminhamentos diferentes
        i. In Order
        ii. Pré Order
        iii. Pós Order

h. Balanceamento da arvore e após isso, mostra ela no terminal, juntamente com o seu tamanho e altura da arvore



5. Aqui está a classe Main do projeto:

```java
Run | Debug
public static void main(String[] args) {
    AvlTree tree = new AvlTree();

    tree.add(element:1);
    tree.add(element:2);
    tree.add(element:3);
    tree.add(element:4);
    tree.add(element:5);
    tree.add(element:6);
    tree.add(element:7);
    tree.add(element:8);
    tree.add(element:9);

    System.out.println(x:"Arvore:");
    tree.printTree();
    System.out.println();
    tree.treeInfo();
    System.out.println();
    tree.balanceTree();
    tree.printTree();

    System.out.println();

    tree.clearTree();
    tree.printTree();
    System.out.println();
    System.out.println(x:"Enchendo a arvore novamente...");

    tree.add(element:9);
    tree.add(element:8);
    tree.add(element:7);
    tree.add(element:6);
    tree.add(element:5);
    tree.add(element:4);
    tree.add(element:3);
    tree.add(element:2);
    tree.add(element:1);
    tree.printTree();
    System.out.println();
    tree.treeInfo();
    System.out.println();
    System.out.print(s:"In Order: ");
    tree.inOrder(tree.getRoot());
    System.out.println();
    System.out.println();
    System.out.print(s:"Pre Order: ");
    tree.preOrder(tree.getRoot());
    System.out.println();
    System.out.println();
    System.out.print(s:"Pos Order: ");
    tree.posOrder(tree.getRoot());
    System.out.println();
    System.out.println();
    tree.balanceTree();      You, anteontem • feat: AVL exte
    tree.printTree();
    System.out.println();
    System.out.print(s:"Tamanho da arvore: ");
    System.out.println(tree.size());
    System.out.print(s:"Altura da arvore: ");
    System.out.println(tree.height());
}
```