

TRABALHO 2 DE ALGORITMOS E ESTRUTURAS DE DADOS

ARVORES AVL

NOMES: MARCOS RAACH, PEDRO SOUZA E LUCAS MARCHESE

NOVEMBRO 2023

1. Funcionamento da Arvore AVL:

Uma árvore AVL é uma árvore de busca binária em que a altura das subárvores esquerda e direita de cada nodo seja diferente no máximo em uma unidade. Isso ocorre por meio de rotações que são aplicadas após operações de inserção ou remoção, garantindo que a árvore permaneça balanceada.

Quando você insere um novo elemento em uma árvore AVL, a inserção é realizada inicialmente como em uma árvore de busca binária convencional. No entanto, após a inserção, é feita uma verificação para garantir que a árvore ainda esteja balanceada. Comparando as alturas das subárvores esquerda e direita.

Se a diferença de altura, chamado de fator de balanceamento, em qualquer nodo se tornar maior que 1 ou menor que -1, a árvore está desbalanceada. Nesse caso, rotações são aplicadas para restaurar o equilíbrio.

1.1 Existem quatro tipos principais de rotações:

1.1.1 Rotação à Esquerda: Quando a subárvore esquerda da subárvore esquerda de um nodo fica mais alta do que a subárvore direita da subárvore esquerda.

1.1.2 Rotação à Direita: Quando a subárvore direita da subárvore direita de um nodo fica mais alta do que a subárvore esquerda da subárvore direita.

1.1.3 Rotação Dupla à Esquerda-Direita: Quando a subárvore esquerda da subárvore direita de um nodo fica mais alta do que a subárvore direita da subárvore esquerda.

1.1.4 Rotação Dupla à Direita-Esquerda: Quando a subárvore direita da subárvore esquerda de um nodo fica mais alta do que a subárvore esquerda da subárvore direita.

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 árvore 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

```
10 public void add(Integer element){ // O(n)
11     Node prev, current;
12     Node node = new Node();
13     node.element = element;
14     node.right = null;
15     node.left = null;
16     if (root == null) {
17         root = node;
18     } else {
19         current = root;
20         while(true) {
21             prev = current;
22             if (element ≤ current.element) {
23                 current = current.left;
24                 if (current == null) {
25                     prev.left = node;
26                     return;
27                 }
28             }
29             else {
30                 current = current.right;
31                 if (current == null) {
32                     prev.right = node;
33                     return;
34                 }
35             }
36         }
37     }
38 }
```

3. Algoritmo de balanceamento

- a. Nosso algoritmo de balanceamento foi dividido 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 árvore e chama o método que balanceia o nó e seus elementos a esquerda e direita.

Após, chama o outro método de balanceamento, onde existem as rotações.

```
113 public void balanceTree() { You, há 1 segund
114     root = balanceTree(root);
115 }
116
117 private Node balanceTree(Node node) {
118     if (node == null) {
119         return null;
120     }
121     node.left = balanceTree(node.left);
122     node.right = balanceTree(node.right);
123     return balance(node);
124 }
125
```

- 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.

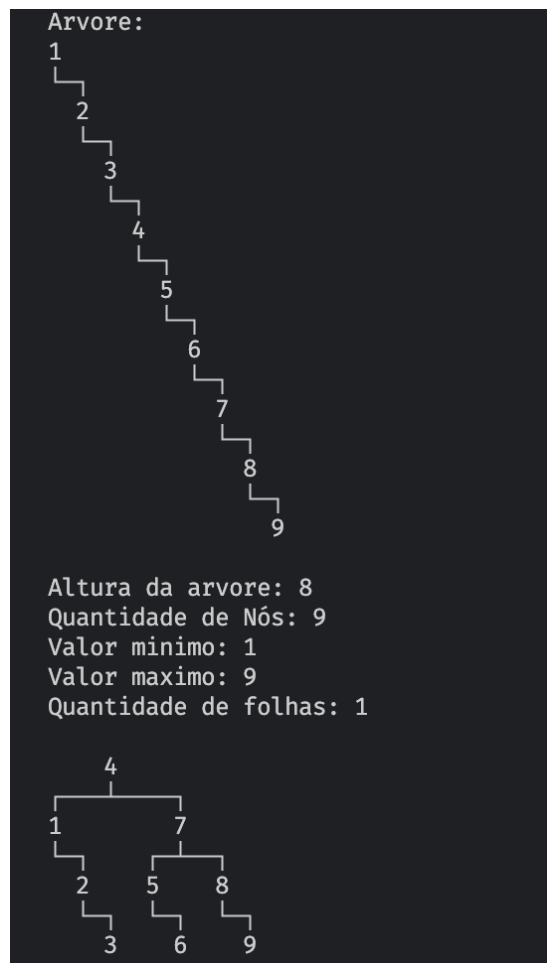
```

126 private Node balance(Node node) {
127     if (node == null) {
128         return null;
129     }
130     updateHeight(node);
131     int balance = getBalance(node);
132
133
134     if (balance > 1 && getBalance(node.left) ≥ 0) {
135         return rotateRight(node);
136     }
137     if (balance < -1 && getBalance(node.right) ≤ 0) {
138         return rotateLeft(node);
139     }
140     if (balance > 1 && getBalance(node.left) < 0) {
141         node.left = rotateLeft(node.left);
142         return rotateRight(node);
143     }
144     if (balance < -1 && getBalance(node.right) > 0) {
145         node.right = rotateRight(node.right);
146         return rotateLeft(node);
147     }
148     // You, há 3 dias • feat: avl tree implementation
149     return node;
150 }
151
152 private void updateHeight(Node node) {
153     if (node != null) {
154         node.height = 1 + Math.max(height(node.left), height(node.right));
155     }
156 }
157
158 private int getBalance(Node node) {
159     if (node == null) {
160         return 0;
161     }
162     return height(node.left) - height(node.right);
163 }
164

```

4. Funcionamento do algoritmo funcionando:

- a. Inicializada a árvore com o método add(element)
 - i. Com números de 1 a 9
- b. Mostrado no terminal:
 - i. Altura da árvore
 - ii. Quantidade de nós
 - iii. Valor mínimo dos números
 - iv. Valor máximo dos números
 - v. Quantidade de nós que são folhas (não tem filhos)
- c. Ocorre o balanceamento da árvore e após isso, a árvore 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

```

Árvore vazia!

Enchendo a árvore novamente...

  9
  |
 8
  |
 7
  |
 6
  |
 5
  |
 4
  |
 3
  |
 2
  |
 1

Altura da árvore: 8
Quantidade de Nós: 9

In Order: 1 2 3 4 5 6 7 8 9
Pre Order: 9 8 7 6 5 4 3 2 1
Pos Order: 1 2 3 4 5 6 7 8 9

      6
     / \
    3   9
   / \ / \
  2  5 8  7
 / \ / \
1  4 7  8

Altura da árvore: 3
Quantidade de Nós: 9

```

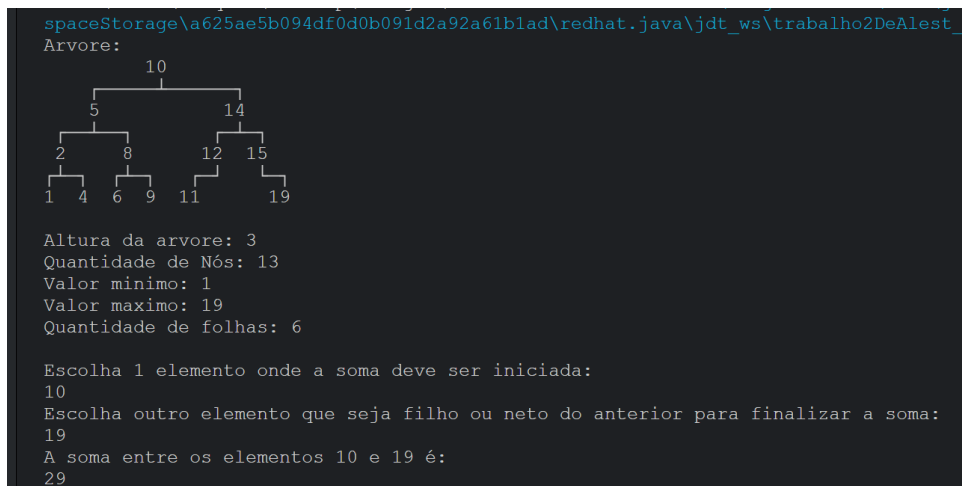
5. Classe Main da árvore AVL:

```

Run | Debug
2  public static void main(String[] args) {
3      AVLTree tree = new AVLTree();
4
5      tree.add(element:1);
6      tree.add(element:2);
7      tree.add(element:3);
8      tree.add(element:4);
9      tree.add(element:5);
10     tree.add(element:6);
11     tree.add(element:7);
12     tree.add(element:8);
13     tree.add(element:9);
14
15     System.out.println(x:"Árvore:");
16     tree.printTree();
17     System.out.println();
18     tree.treeInfo();
19     System.out.println();
20     tree.balanceTree();
21     tree.printTree();
22
23     System.out.println();
24
25     tree.clearTree();
26     tree.printTree();
27     System.out.println();
28     System.out.println(x:"Enchendo a árvore novamente ... ");
29
30     tree.add(element:9);
31     tree.add(element:8);
32     tree.add(element:7);
33     tree.add(element:6);
34     tree.add(element:5);
35     tree.add(element:4);
36     tree.add(element:3);
37     tree.add(element:2);
38     tree.add(element:1);
39     tree.printTree();
40     System.out.println();
41     tree.treeInfo();
42     System.out.println();
43     System.out.print(s:"In Order: ");
44     tree.inOrder(tree.getRoot());
45     System.out.println();
46     System.out.println();
47     System.out.print(s:"Pre Order: ");
48     tree.preOrder(tree.getRoot());
49     System.out.println();
50     System.out.println();
51     System.out.print(s:"Pos Order: ");
52     tree.posOrder(tree.getRoot());
53     System.out.println();
54     System.out.println();
55     tree.balanceTree();
56     tree.printTree();
57     System.out.println();
58     System.out.print(s:"Tamanho da árvore: ");
59     System.out.println(tree.size());
60     System.out.print(s:"Altura da árvore: ");
61     System.out.println(tree.height());
62 }

```

6. Funcionamento dos algoritmos na Arvore BST:



7. Classe main da Arvore BST:

```
2
3 public class MainBST {
4     public static void main(String[] args) {
5         BinarySearchTree bst = new BinarySearchTree();
6         Scanner sc = new Scanner(System.in);
7
8         bst.add(10);
9         bst.add(5);
10        bst.add(14);
11        bst.add(12);
12        bst.add(15);
13        bst.add(19);
14        bst.add(8);
15        bst.add(2);
16        bst.add(4);
17        bst.add(6);
18        bst.add(9);
19        bst.add(1);
20        bst.add(11);
21
22        System.out.println("Arvore:");
23        bst.printTree();
24        System.out.println();
25        bst.treeInfo();
26        System.out.println();
27        System.out.println("Escolha 1 elemento onde a soma deve ser iniciada: ");
28        int start = sc.nextInt();
29        System.out.println("Escolha outro elemento que seja filho ou neto do anterior para finalizar a soma: ");
30        int end = sc.nextInt();
31        System.out.println("A soma entre os elementos " + start + " e " + end + " é: ");
32        System.out.println(bst.sumBetween(start, end));
33        sc.close();
34    }
35 }
36
```

You, 22 hours ago • feat: add relatorio