Implementação Árvore de Pesquisa AVL

Marina Schwerz Bon (Matrícula - 23102573)

1. Introdução á Árvore de Pesquisa AVL

A árvore de pesquisa AVL foi nomeada em homenagem aos seus desenvolvedores, Georgy Adelson-Velsky and Evgenii Landis, que introduziram a sua criação em uma publicação em 1962.

A árvore AVL têm muito em comum com a árvore balanceada de pesquisa, cada sub-árvore é balanceada, e ambas são compregadas em implementações voltadas a busca de dados. Portanto, ambas implementações contam com o balanceamento de sua estrutura. A árvore AVL conta com um mecanismo de balanceamento para atingir um equilíbro. Primeiramente, a diferença entre as alturas da subárvore da esquerda e da direita é sempre menor ou igual a 1, essa diferença é chamada de Fator de Equilíbrio. Se o fator de equilíbrio de qualquer nodo for 1, a subárvore da esquerda está um nível mais alta do que a da direita. Paralelamente, se o fator de equilíbrio de qualquer nodo for -1, a subárvore da esquerda está um nível mais baixa do que a da direita. Em contrapartida, se o fator de equilíbrio de qualquer nodo for 0, a subárvore da esquerda e a subárvore direita têm alturas iguais. Ou seja, sempre que o fator de equilíbrio não for igual a zero, deve ocorrer uma operação de balanceamento.

Para garantir uma árvore balanceada, sempre quando um nodo é introduzido, se necessário, há quatro rotações possíveis, rotação para direita, rotação para esquerda, rotação direita-esquerda e por fim, rotação esquerda-direita. A figura 1 ilustra uma rotacão esquerda-direita da estrutura.

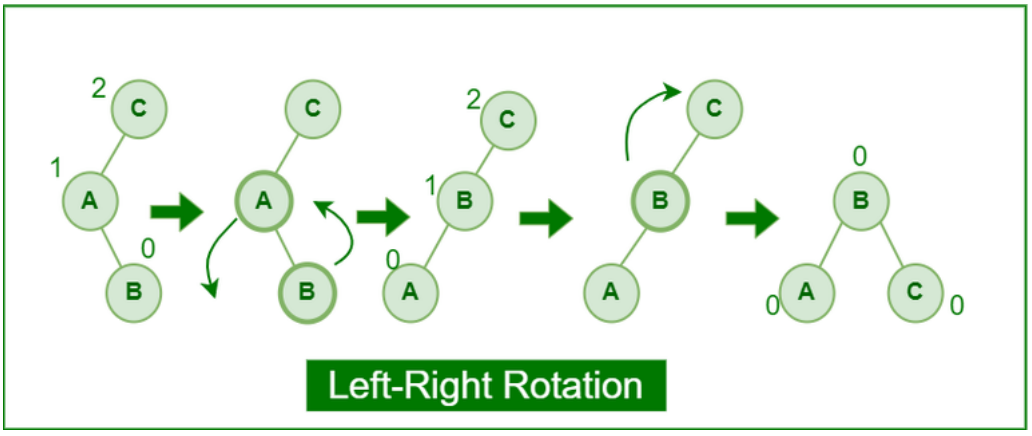

Figura 1. Representação visual de uma rotação esquerda-direita.

2. Método *int Balance (Node n)*

O método "private int balance (Node n)" realiza o cálculo do fator de equílibrio e retorna o resultado. O cálculo é realizado através da diferença entre a altura da subárvore da direita e a

altura da subárvore da esquerda do nodo fornecido. Quando o nodo for nulo, a função retorna 0, indicando que a subárvore está equilibrada. Por outro lado, se o nodo não for nulo, a função retorna a diferença das alturas, -1 ou 1. O grau de complexidade do método é O(log n).

```
    private int balance(Node n) {
        return (n == null) ? 0 : height(n.left) - height(n.right); // calcula o fator
de equilíbrio
    }
```

3. **Método** *int height (Node n)*

O método height(Node n) é responsável por calcular a altura de um nodo na árvore AVL. A altura de um nodo é definida como a distância do nodo até sua folha mais distante, medida pelo número de arestas. A função recebe um nodo n como parâmetro. Se esse for nulo, a altura é considerada 0. Se o nodo não for nulo, o método retorna a altura do nodo, que é armazenada no atributo height. A expressão (n == null) ? 0 : n.height significa: se n for nulo, retorne 0; caso contrário, retorne n.height. O grau de complexidade do método é O(1).

```
    private int height(Node n) {
        return (n == null) ? 0 : n.height;
    }
```

4. **Método** *void add(Integer element) & addRecursive(Node n, Node father, Integer element)*

Primeiramente o método "public void add(Integer element)" chama a função privada addRecursive, que realiza a inserção recursiva de um elemento na árvore. Cada chamada recursiva ajusta as subárvores à medida que o nodo é inserido. A adição é feita da seguinte maneira:

- O método verifica se o nodo n é nulo. Se for, cria um novo nodo com o elemento fornecido.
- Em seguida, compara o elemento a ser inserido com o elemento do nodo n e decide se a inserção deve ser feita na subárvore da esquerda ou da direita.
- Após a inserção, a altura do nodo n é atualizada usando a função height, que retorna a altura da árvore enraizada no nodo.
- É calculado o fator de equilíbrio do nodo a partir da chamada do método balance.
- O código então verifica quatro casos para determinar se é necessário realizar rotações para manter o balanceamento da árvore. As rotações são realizadas chamando as funções rightRotate e leftRotate. O primeiro if após o cálculo do balanceamento representa uma rotação para a direita. Já o segundo if, representa uma rotação para a esquerda. O terceiro

if, representa uma rotação esquerda direita, enquando o quarto if, representa uma rotação direita esquerda.

O grau de complexidade do método é O(log n).

```
   public void add(Integer element) {
       root = addRecursive(root, null, element);// chamada do método private Node
addRecursive
       count++;
   }

   private Node addRecursive(Node n, Node father, Integer element) {
       if (n == null) {
           return new Node(element);
       }
       if (element < n.element) { // elemento menor, subárvore da esquerda
           n.left = addRecursive(n.left, n, element);
       } else if (element > n.element) { // elemento maior, subárvore da direita
           n.right = addRecursive(n.right, n, element);
       } else {
           return n;
       }

       n.height = 1 + Math.max(height(n.left), height(n.right)); // atualiza altura

       int balance = balance(n); // chama o método para calcular o balance

       if (balance > 1 && element < n.left.element) { // rotação direita
           return rightRotate(n);
       }
       if (balance < -1 && element > n.right.element) { // rotação esquerda
           return leftRotate(n);
       }
       if (balance > 1 && element > n.left.element) { // rotação esquerda direita
           n.left = leftRotate(n.left);
           return rightRotate(n);
       }
       if (balance < -1 && element < n.right.element) { // rotação direita esquerda
           n.right = rightRotate(n.right);
           return leftRotate(n);
       }
       return n;
   }
```

5. **Método** *Node rightRotate(Node n)*

O método "private Node rightRotate(Node n)" realiza a rotação para a direita ao redor do nodo n. O algoritmo envolve três etapas principais:

- Reatribuição de Ponteiros:
  - O nodo à esquerda de n é armazenado em uma variável auxiliar chamada x.
  - A subárvore da direita de x é atribuída à subárvore da esquerda de do nodo n
  - O nodo n é atribuído como a subárvore da direita de x.
- Atualização das alturas:
  - As alturas dos nodos n e x são recalculadas após a rotação.
  - A altura dos nodos é definida como 1 mais o máximo entre as alturas de suas subárvores da esquerda e da direita.
- Retorno da nova raiz:
  - O nodo x é a nova raiz da subárvore que foi rotacionada para a direita.

A rotação para a direita é aplicada quando a subárvore da esquerda de um nó se torna mais pesada que a subárvore da direita, visando restaurar o balanceamento da árvore. A complexidade do método é O(log n).

```
private Node rightRotate(Node n) {
    Node x = n.left;
    Node aux = x.right;

    x.right = n; // rotação
    n.left = aux; // rotação

    n.height = 1 + Math.max(height(n.left), height(n.right)); // atualiza altura
    x.height = 1 + Math.max(height(x.left), height(x.right)); // atualiza altura

    return x;
}
```

6. **Método** *Node leftRotate(Node n)*

O método "private Node rightLeft(Node n)" realiza a rotação para a esquerda ao redor do nodo n. O algoritmo envolve três etapas principais:

- Reatribuição de Ponteiros:
  - O nodo à direita de n é armazenado em uma variável auxiliar chamada x.
  - A subárvore da esquerda de x é atribuída à subárvore da direita de do nodo n
  - O nodo n é atribuído como a subárvore da esquerda de x.
- Atualização das alturas:
  - As alturas dos nodos n e x são recalculadas após a rotação.

    - A altura dos nodos é definida como 1 mais o máximo entre as alturas de suas subárvores da esquerda e da direita.
- Retorno da nova raiz:
    - O nodo x é a nova raiz da subárvore que foi rotacionada para a esquerda.

A rotação para a esquerda é aplicada quando a subárvore da direita de um nó se torna mais pesada que a subárvore da esquerda, visando restaurar o balanceamento da árvore. A complexidade do método é O(log n).

```
private Node leftRotate(Node n) {
    Node x = n.right;
    Node aux = x.left;

    x.left = n; // rotação
    n.right = aux; // rotação

    n.height = 1 + Math.max(height(n.left), height(n.right)); // atualiza altura
    x.height = 1 + Math.max(height(x.left), height(x.right)); // atualiza altura

    return x;
}
```

7. Referências

"AVL Tree (Data Structures) - Javatpoint." *Www.Javatpoint.Com*, JavaTPoint, www.javatpoint.com/avl-tree.

"AVL Tree Data Structure." *GeeksforGeeks*, 2 Nov. 2023, www.geeksforgeeks.org/introduction-to-avl-tree/.

"Deletion in an AVL Tree." *GeeksforGeeks*, 18 Jan. 2023, www.geeksforgeeks.org/deletion-in-an-avl-tree/?ref=lbp.

freeCodeCamp.org. "AVL Tree Insertion, Rotation, and Balance Factor Explained." *freeCodeCamp.Org*, 16 July 2021, www.freecodecamp.org/news/avl-tree-insertion-rotation-and-balance-factor/.

"Insertion in an AVL Tree." *GeeksforGeeks*, 3 Oct. 2023, www.geeksforgeeks.org/insertion-in-an-avl-tree/?ref=lbp.

Maverick. "What Is AVL Tree: AVL Tree Meaning." *GeeksforGeeks*, 15 Mar. 2023, www.geeksforgeeks.org/what-is-avl-tree-avl-tree-meaning/.