Árvore Binária de Pesquisa Balanceada - AVL

Gabriela Panta Zorzo, Morgana Luiza Weber

Escola Politécnica – Pontifícia Universidade Católica do RS (PUCRS)
Av. Ipiranga, 6681 – 90.619-900 – Porto Alegre – RS – Brasil

{gabriela.zorzo,morgana.weber}@edu.pucrs.br

Abstract. *This paper presents the concepts of an AVL Self-Balancing Binary Search Tree, as well as how it operates and the main algorithms. It brings as an example the implementation of an AVL Tree and part of the coding developed for it to function. This article is part of the discipline of Algorithms and Data Structures I, of the Polytechnic School of Pontifícia Universidade Católica do Rio Grande do Sul.*

Resumo. *Este artigo apresenta os conceitos de uma Árvore Binária de Pesquisa Balanceada AVL, bem como seu funcionamento e seus principais algoritmos. Ele traz como exemplo a implementação de uma Árvore AVL e parte dos códigos desenvolvidos para o seu funcionamento. Este artigo compõe a disciplina de Algoritmos e Estruturas de Dados I, da Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.*

1. O que é a Árvore de Pesquisa Balanceada AVL

A Árvore de Pesquisa Balanceada AVL recebe seu nome devido aos seus criadores Adelson-Velskii e Landis, e foi implementada em 1962. Ela pode ser definida como uma Árvore Binária de Pesquisa Balanceada onde cada um de seus nodos é associado a um fator de balanceamento que deve ser +1, 0 ou -1. Este fator de balanceamento é calculado através da subtração entre a altura da sub-árvore da direita e da sub-árvore da esquerda [JavaTPoint 2020]. Se o fator de balanceamento for +1, significa que a sub-árvore da esquerda é um nível mais alta que a sub-árvore da direita. Caso o fator de balanceamento seja -1, a sub-árvore da direita possui um nível a mais que a sub-árvore da esquerda. Se este fator for igual a 0, ambas as sub-árvores possuem a mesma altura (Figura 1).

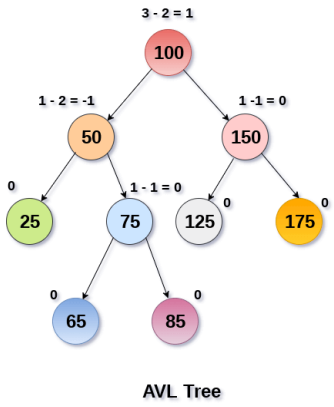

Figura 1. Fatores de Balanceamento [JavaTPoint 2020]

Este balanceamento é dado de forma automática durante os processos de inserção e remoção de novos nodos, a partir de regras de balanceamento estipuladas. Caso o balanceamento não esteja adequado, a árvore deverá ser balanceada através de rotações em seus nodos. Exitem duas formas de balancear uma árvore AVL: através da rotação para a direita ou da rotação para a esquerda. [Baeldung 2020].

2. Algoritmos da Árvore de Pesquisa Balanceada AVL

Para adicionar nodos em uma Árvore AVL, primeiramente é necessário verificar o balanceamento dos nodos da árvore para saber onde inserir o novo nodo. Esta verificação é feita através do algoritmo da Figura 2. Além disto, para fins de inserção, o posicionamento dos nodos segue o mesmo princípio da inserção de uma Árvore Binária de Pesquisa, onde nodos com valores maiores que o valor do nodo pai ficam posicionados à esquerda, enquanto nodos com valores menores ficam posicionados à direita [de Souza 2020]. Desta forma, temos 4 tipos de inserção: rotação LL (nodo inserido na sub-árvore da esquerda da sub-árvore da esquerda de X), rotação RR (nodo inserido na sub-árvore da direita da sub-árvore da direita de X), rotação LR (nodo inserido na sub-árvore da direita da sub-árvore da esquerda de X) e rotação RL (nodo inserido na sub-árvore da esquerda da sub-árvore da direita de X), sendo X o nodo cujo fator de balanceamento é diferente de -1, 0 e 1 [JavaTPoint 2020].

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

Figura 2. Algoritmo de Verificação do Fator de Balanceamento

No algoritmo de rotação à esquerda (Figura 3), os arranjos de nodos da direita são transformados em arranjos na esquerda. Já no algoritmo de rotação à direita (Figura 4), o contrário acontece [Programiz 2020].

```
private Node leftRotate(Node x) {
    Node y = x.right;
    Node T2 = y.left;

    y.left = x;
    x.right = T2;

    x.height = Math.max(height(x.left), height(x.right)) + 1;
    y.height = Math.max(height(y.left), height(y.right)) + 1;

    return y;
}
```

Figura 3. Algoritmo de Rotação à Esquerda

Desta forma, a Árvore Binária de Pesquisa AVL vai sendo automaticamente balanceada conforme novos nodos vão sendo inseridos, respeitando sempre a regra do fator de balanceamento.

```java
private Node rightRotate(Node y) {
    Node x = y.left;
    Node T2 = x.right;

    x.right = y;
    y.left = T2;

    y.height = Math.max(height(y.left), height(y.right)) + 1;
    x.height = Math.max(height(x.left), height(x.right)) + 1;

    return x;
}
```

Figura 4. Algoritmo de Rotação à Direita

3. Implementação de uma AVL

Para fins de teste, foi instanciada uma árvore AVL e inseridos 10 elementos a ela de forma manual através de um método add criado. Foram adicionados os elementos 10, 5, 1, 3, 24, 12, 33, 7, 99 e 0, respectivamente. O resultado final da árvore após as inserções, está na Figura 5, que exibe a impressão dos nodos a partir de um código que se encontra no GitHub [Wani 2020]. Desta forma, é possível observar que a árvore está balanceada de acordo com o critério do fator de balanceamento, uma vez que a diferença entre suas alturas respeita os valores de +1, 0 e -1.

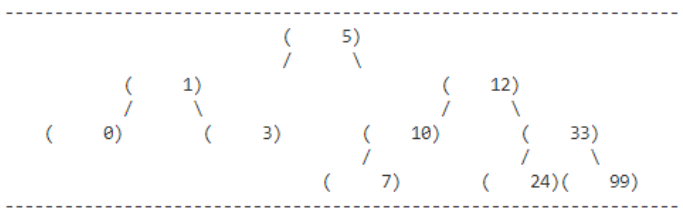

Figura 5. Árvore AVL Resultante das Inserções

Com o objetivo de consultar os elementos da Árvore AVL instanciada, foram chamados os demais métodos implementados, tais como getParent(), contains(x), heigth(x), isBalanced(), size(), isEmpty(), positionsPre(), positionsCentral(), positionsPos() e positionsWidth() (Figura 6).

```
Pai de 7: 10
33 esta na arvore:  true
55 esta na arvore: false
Altura da arvore: 4
Esta balanceada? true
A arvore tem 10 elementos.
A arvore esta vazia? false
Caminhamento pre-fixado: 5, 1, 0, 3, 12, 10, 7, 33, 24, 99,
Caminhamento central: 0, 1, 3, 5, 7, 10, 12, 24, 33, 99,
Caminhamento pos-fixado: 0, 3, 1, 7, 10, 24, 99, 33, 12, 5,
Caminhamento por largura: 5, 1, 12, 0, 3, 10, 33, 7, 24, 99,
```

Figura 6. Chamada de Métodos de Consulta

Referências

Baeldung (2020). *Guide to AVL Trees in Java*. Disponível em: https://www.baeldung.com/java-avl-trees Accessado em: 24.11.2020.

de Souza, J. F. (2020). *Árvores AVL*. Disponível em: https://www.ufjf.br/jairo-souza/files/2009/12/5-Indexação-Arvore-AVL.pdf Accessado em: 24.11.2020.

JavaTPoint (2020). *AVL Tree*. Disponível em: https://www.javatpoint.com/avl-tree Accessado em: 24.11.2020.

Programiz (2020). *AVL Tree*. Disponível em: https://www.programiz.com/dsa/avl-tree Accessado em: 24.11.2020.

Wani, N. J. (2020). *AVL Tree*. Disponível em: https://gist.github.com/nehaljwani/8243688 Accessado em: 24.11.2020.