# Dicionários Avançados: AVL e Árvore Rubro-Negra

**Antonio Rege L. dos Santos, Victor A. Vieira**

Instituto de Computação
Universidade Federal Fluminense (UFF) – Niterói, RJ – Brazil

Grupo de Informática para Pesquisa em Computação
Instituto Federal do Acre (IFAC) – Rio Branco, AC – Brazil

{antonio.rsantos,victor.vieira}@ifac.edu.br

***Abstract.*** *Binary search trees are the most commonly used tree type in computing. They address a classic problem, which is search. To optimize this operation, strategies involving making the tree have fewer levels are the ones that generate better results. This article presents and compares two types of balanced binary search trees: AVL and Red-Black Tree. The results show that AVL tends to be more efficient for the search in worst cases, while the Red-Black Tree is more efficient in the insertion and removal operations.*

***Resumo.*** *As árvores binárias de busca são o tipo de árvore mais utilizado na computação. Elas abordam um problema clássico, que é a busca. Para otimizar essa operação, estratégias envolvendo fazer com que a árvore possua menos níveis são as que geram melhores resultados. Este artigo apresenta e compara dois tipos de árvores binárias de busca balanceadas: AVL e Árvore Rubro-Negra. Os resultados mostram que AVL tende a ser mais eficiente para a busca em piores casos, enquanto a Árvore Rubro-Negra é mais eficiente nas operações de inserção e remoção.*

## 1. Introdução

Uma árvore, em estrutura de dados, é um tipo abstrato de dados que pode assumir uma forma vazia ou ser formada por um nó raiz e zero ou mais subárvores [1]. As árvores binárias são aquelas nas quais cada nó pode conter até duas subárvores: direita e esquerda. A Figura 1 apresenta uma árvore binária.

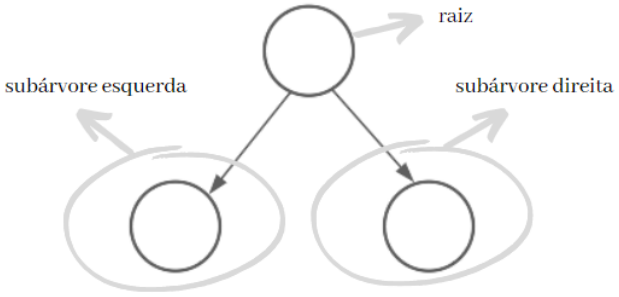


**Figura 1. Estrutura de uma árvore binária**

Dentre os tipos de árvores binárias, estão as Árvores Binárias de Busca (ABB), que são as mais utilizadas na computação [2]. Como as outras árvores binárias, elas possuem nós, raiz, filhos, pais e folhas e obedecem às propriedades. Entretanto, nas ABB, os elementos inseridos são sempre comparados desde o nó raiz, para que se saiba se serão inseridos à esquerda, quando forem menores que um nó, ou à direita, quando forem maiores [1].

A Figura 2 mostra a inserção do nó 40 em uma ABB. Como o valor é maior que 30, que é a raiz da árvore, será comparado com 50. Sendo menor que 50 e não havendo nenhum nó como subárvore esquerda de 50, 40 será inserido.

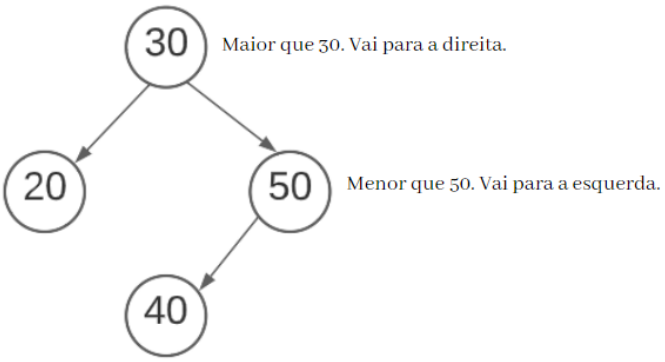


**Figura 2. Inserção de um nó em uma ABB**

O resultado prático da abordagem utilizada em ABB pode ser melhor compreendido com um exemplo. Suponha que será percorrida uma árvore binária de busca com n nós e o percurso é em ordem, ou seja, serão visitados recursivamente primeiro os nós à esquerda, depois o próprio nó, depois os nós à direita. O resultado será sempre a sequência de dados ordenados, já que esse tipo de árvore aborda o clássico problema da busca em estruturas de dados.

Dessa forma, entendemos que a busca é otimizada quando utilizamos ABB, se compararmos a outros tipos de abstratos de dados, bem como entendemos que trata-se de um tipo de árvore ideal para quando se quer encontrar conteúdos que estão dispostos de maneira organizada [2].

Dentre os tipos de ABB, encontramos árvores que são consideradas dicionários avançados, como AVL e Árvore Rubro-Negra. Elas serão abordadas neste artigo. O restante do texto está organizado da seguinte forma: a seção 2 apresenta a AVL, discutindo balanceamento e rotações; a seção 3 traz a definição e propriedades da Árvore Rubro-Negra e discute a operação de inserção; na seção 4, é apresentado um comparativo com relação à análise de complexidade das duas árvores; a seção 5 traz as conclusões, incluindo sugestão de trabalho futuro; e, finalmente, há as referências utilizadas na pesquisa.

## 2. AVL

A AVL, cujo nome deriva do acrônimo dos seus criadores, Adelson Velsky e Landis, é a primeira ABB, criada no ano de 1962. Sua característica principal é trabalhar com rotações para manter a árvore como quase completa. Isso garante menos níveis e menor custo para a operação de busca [2]. Serão abordados os custos das operações em ABB e AVL na Seção 4.

### 2.1. Balanceamento

Parte fundamental da compreensão de AVL é o balanceamento. Ele consiste numa tentativa, baseada em rearranjo da árvore, de mantê-la com a menor quantidade de

níveis ou a menor diferença entre alturas de subárvores a partir de um mesmo nó [3]. A altura consiste no tamanho do maior caminho que conecta um nó a uma folha descendente. A Figura 3 apresenta uma árvore onde podemos verificar sua altura e de suas subárvores, para compreensão.

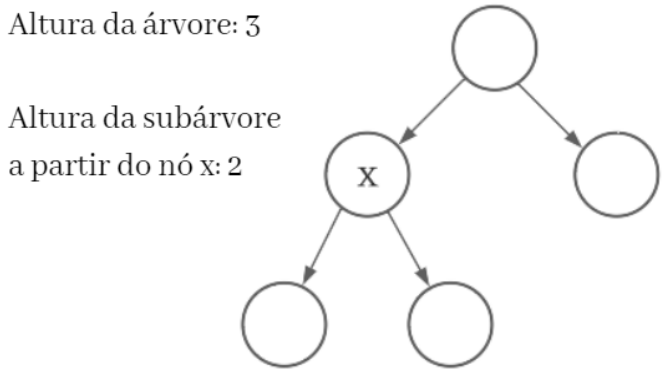


**Figura 3. Altura de uma árvore e de uma de suas subárvores**

O Fator de Balanceamento (FB) é a variável que determina se serão necessárias rotações ou não em uma AVL. Para calcular o FB de um nó, usamos: **altura da subárvore à esquerda - altura da subárvore à direita**. Na AVL, quando o FB de um nó chega em 2, positivo ou negativo, dizemos que a árvore está desbalanceada e partimos para fazer rotações, balanceando-a [2]. A Figura 4 apresenta uma AVL desbalanceada.

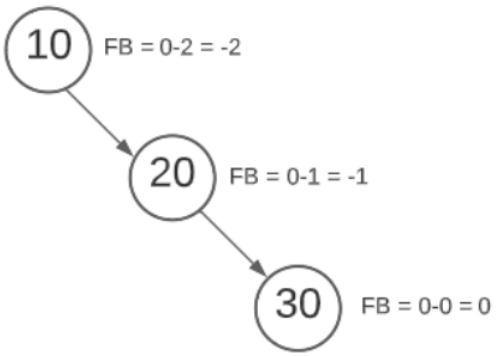


**Figura 4. AVL desbalanceada**

Note que o FB no nó raiz é -2. Isso indica a necessidade de rotação e representa, na prática, que essa árvore possui mais nós à direita, o que pode influenciar negativamente o desempenho da operação de busca - quanto mais nós em um maior caminho até uma folha, pior, quando quisermos encontrar a folha.

## 2.2. Rotações

As rotações em AVL podem ser simples ou duplas. Quando os sinais do FB do nó desbalanceado e de seu filho forem iguais ou o FB do filho for igual a 0, será necessário fazer uma rotação simples. Quando os sinais forem diferentes, será necessário fazer uma rotação dupla [2]. A Tabela 1, a seguir, apresenta a relação entre os valores dos fatores de balanceamento e as rotações.

**Tabela 1. Tabela de relação entre fatores de balanceamento e rotações**

| FB do nó | FB do nó filho | Tipo de Rotação |
|---|---|---|
| 2 | 1 | Simples |
| | 0 | |
| | -1 | Dupla |
| -2 | 1 | |
| | 0 | Simples |
| | -1 | |

O esquema a seguir, apresentado na Figura 5, mostra a inserção de um nó na árvore, gerando desbalanceamento. No caso, será necessária uma rotação simples, considerando que a raiz e seu filho possuem o mesmo sinal. O algoritmo da rotação simples gera uma nova raiz para a árvore, que é o elemento filho da raiz anterior. Ainda, a raiz anterior, no caso, se torna filho à esquerda da nova raiz. Após, a árvore está balanceada.

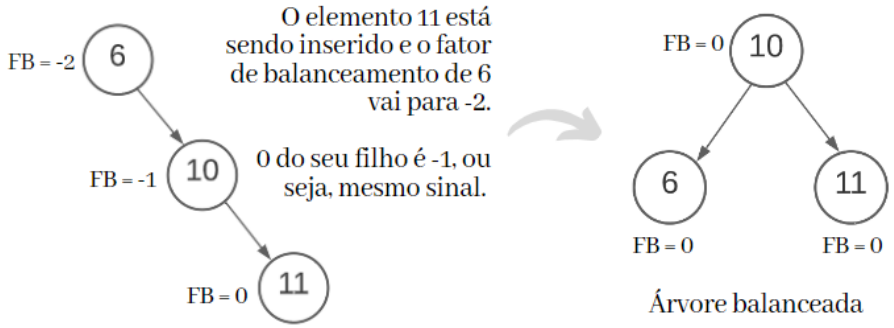


**Figura 5. Rotação simples em AVL**

Já na rotação dupla, é necessário fazer duas rotações: a primeira inserindo a folha entre os dois primeiros elementos da subárvore desbalanceada e a segunda, uma rotação simples [2]. A Figura 6 apresenta um exemplo.

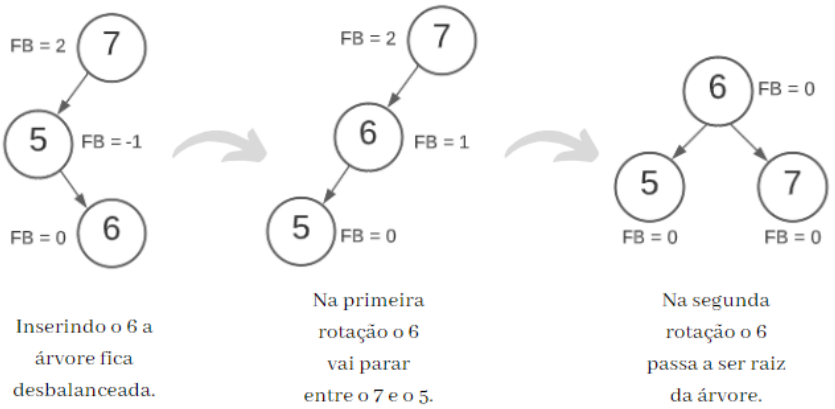


**Figura 6. Rotação dupla em AVL**

Com a inserção do nó, a AVL fica desbalanceada. O FB da raiz vai para 2 e do seu filho é -1. Nesse caso, é necessário rotacionar uma vez para inserir o novo nó entre os dois anteriores e, em seguida, rotacionar outra vez para torná-lo raiz da árvore.

É importante destacar que a operação de remoção também pode gerar necessidade de balanceamento na árvore e esta operação será tratada neste artigo apenas para efeito de entendimento da complexidade assintótica, na Seção 4. Na seção seguinte são apresentadas as características e propriedades de outro tipo de ABB: Árvore Rubro-Negra.

## 3. Árvore Rubro-Negra

A Árvore Rubro-Negra (ARN) também é definida como uma árvore binária de busca e se difere das demais pelo fato de que cada nó armazena, além da chave, uma cor. Seu nome, inclusive, deriva do fato de que a coloração de seus nós pode ser vermelha ou preta, ou seja os nós são rubros ou negros. É importante destacar que o balanceamento nessa árvore é feito através das cores [1][4].

Uma árvore binária de busca pode ser considerada uma ARN se obedecer às seguintes propriedades [1][4][5]:

- todo nó for rubro ou negro;
- a raiz da árvore for negra;
- todo nó nulo for negro - conceitualmente, todo nó nulo é considerado negro em árvores rubro-negras;
- um nó rubro tiver ambos os filhos negros;
- qualquer caminho de um nó até um nó nulo tiver sempre o mesmo número de nós negros.

Essas propriedades são mantidas através de mudanças de cores nos nós e rotações. Considerando que as cores são o que importa para o balanceamento, o algoritmo faz as verificações e garante que as propriedades sejam satisfeitas [4].

### 3.1. Inserção

Na inserção em uma ARN, cada nó inserido possui por definição cor rubra. O modelo de inserção segue exatamente o de uma ABB. Na Figura 7 temos um exemplo de árvore ARN balanceada, após a inserção do nó 62.

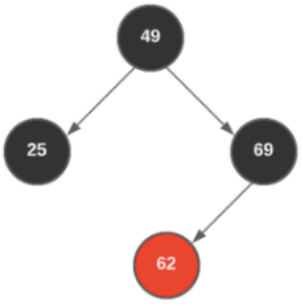


**Figura 7. ARN balanceada**

Após a inserção, deve ser verificado se as propriedades da árvore se mantêm. Caso alguma delas seja violada, será necessário restabelecer a corretude. Isso pode ser feito através de trocas de cores em alguns nós ou rotações [5]. A Figura 8 apresenta o caso de inserção da figura anterior, onde são necessárias alterações de cores.


**Figura 8. Etapas do balanceamento após inserção em ARN**

Ao verificarmos que a inserção do elemento 62 gerou um pai rubro, consideramos que a árvore está desbalanceada e partimos para os ajustes necessários. Vemos que, logo após a inserção, o pai e o tio do novo nó são rubros. Foi necessário alterar a cor dos dois para negros. Essa alteração tornou a raiz da árvore rubra, o que também fere uma das propriedades da árvore, sendo necessário alterá-la para que se considere balanceada.

No exemplo anterior, foram necessárias alterações de cores nos nós para o balanceamento. Porém, existem casos em ARN onde será necessário fazer rotações para balanceá-la, como o que aparece na Figura 9.

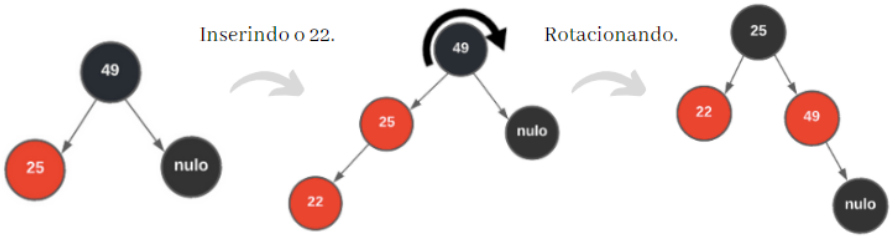


**Figura 9. Rotação simples e mudança de cores para balanceamento em ARN**

Vemos que a árvore contém os elementos 49 como raiz e 25 como folha. Em seguida, é necessário que aconteça uma rotação simples à direita para que a árvore seja balanceada, além do ajuste nas cores dos elementos.

A operação de remoção em ARN é bastante complexa e possui várias etapas. Disponibilizamos o link do site programiz.com, que descreve de maneira precisa o algoritmo e os ajustes necessários para manter o balanceamento da árvore [6]. Vistas as propriedades de AVL e ARN, a seção seguinte traz um comparativo entre as duas.

## 4. Comparações

Visto que AVL e ARN são dois dos tipos de ABB, a Tabela 2 apresenta uma comparação das complexidades no pior caso de cada operação nessas árvores.

**Tabela 2. Comparação entre a complexidade das árvores**

| Operações | ABB | AVL | ARN |
|---|---|---|---|
| inserção, remoção e busca | O(n) | O(log n) | O(log n) |

Percebe-se que, tanto em AVL quanto em ARN, o algoritmo trabalha em notação assintótica de O(log n). Já em ABB, fica em O(n). A partir daí, podemos concluir que AVL e ARN são ABB otimizadas, pois permitem que inserção, remoção e busca sejam feitas com menor custo no pior caso [1][3].

Teoricamente, AVL e ARN possuem a mesma complexidade. Na prática, considera-se que AVL é mais rápida na operação de busca, por se tratar de um tipo de árvore mais balanceada que a Rubro-Negra - AVL tem menos níveis com grande quantidade de dados e geralmente é uma árvore completa com pequena quantidade de dados [7].

Em compensação, AVL é mais lenta nas operações de inserção e remoção, visto que ARN trabalha o balanceamento a partir das cores e alterá-las possui custo inferior ao de fazer rotações [7]. Acredita-se que por esse fato as árvores rubro-negras sejam mais comumente utilizada em bibliotecas que implementam mapas finitos nas linguagens de programação, como em java.util.TreeMap em Java e std::map e std::set em C++, considerando que são realizada mais operações de inserção e remoção que operações de busca nas aplicações [7]-[10].

Para teste comparativo, foram implementados dois códigos em C++ testados com a execução das mesmas entradas e saídas. O primeiro programa implementa uma em AVL com um algoritmo recursivo. O segundo implementa uma ARN com a biblioteca std::set. Usamos a biblioteca std::chrono pra verificar o tempo de execução.

Os resultados não permitiram concluir qual dos dois algoritmos é mais rápido, já que estão expressos em unidade de tempo e os valores foram muito próximos, considerando as médias obtidas na execução dos programas. Acredita-se que a utilização de dados em larga escala possibilitaria visualizar a diferença. Os códigos estão disponíveis em github.com/vrcvieira/binary-search-trees.

## 5. Conclusões

Este artigo abordou dois tipos de árvores binárias de busca: AVL e Árvore Rubro-Negra. São tipos abstratos considerados dicionários avançados em Estrutura de Dados por trabalharem otimizando a operação de busca. Percebe-se que ambas as estruturas são úteis para quando se precisa manipular dados de maneira organizada, com caminhos definidos.

No estudo comparativo, verificou-se que ambas as ABB estudadas são melhores em custo computacional que árvores não balanceadas. Ainda, não foi possível concluir qual dos dois tipos de árvores estudados é melhor em tempo de execução, mas as pesquisas realizadas apontaram para ARN como a mais eficiente nas operações de inserção e remoção e AVL mais eficiente na operação de busca.

Como trabalho futuro, sugere-se realizar o mesmo teste adotado neste trabalho com dados em larga escala. Isso permitiria visualizar qual é mais eficiente em tempo de execução, mesmo que os algoritmos possuam abordagens diferentes - um é recursivo, programado completamente, e o outro foi desenvolvido utilizando uma biblioteca da linguagem C++.

## References


[1] Leiserson, Charles E.., Stein, Clifford., Cormen, Thomas H.., Rivest, Ronald L.. Introduction to Algorithms. Reino Unido: MIT Press, 2009.

[2] Ascencio, Ana; Araújo, Graziela. Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. Brazil: Pearson, 2015.

[3] STRBAC-SAVIĆ, S.; TOMASEVIC, M. Comparative performance evaluation of the AVL and red-black trees. ACM International Conference Proceeding Series, [s.l.], p. 14–19, 2012. ISBN: 9781450312400, DOI: 10.1145/2371316.2371320.

[4] LIEW, C. W.; NGUYEN, H. Using an Intelligent Tutoring System to Teach Red Black Trees. [s.l.], p. 1280–1280, 2019. DOI: 10.1145/3287324.3293823.

[5] XHAKAJ, F.; LIEW, C. W. A new approach to teaching red black trees. Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, [s.l.], v. 2015-June, p. 278–283, 2015. ISBN: 9781450334402, ISSN: 1942647X, DOI: 10.1145/2729094.2742624.

[6] “Deletion From a Red-Black Tree”. https://www.programiz.com/dsa/deletion-from-a-red-black-tree, julho de 2022.

[7] “Why is std::map implemented as a red-black tree?”. https://stackoverflow.com/questions/5288320/why-is-stdmap-implemented-as-a-red-black-tree, julho de 2022.

[8] “Class TreeMap<K,V>”. https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html, julho de 2022.

[9] “std::map”. https://en.cppreference.com/w/cpp/container/map, julho de 2022.

[10] “std::set”. https://en.cppreference.com/w/cpp/container/set, julho de 2022.