Balanceamento de Árvores

Amanda Carolina Ambrizzi Ramin[1]

Resumo: Este artigo analisa as principais técnicas de balanceamento de árvores, com foco nos métodos B-tree, AVL e Rubro-Negra, muito utilizados em estruturas de dados para otimizar operações de busca, inserção e remoção. Além de discutir as características e aplicações de cada método, o estudo apresenta as árvores Treap e Splay como alternativas para o balanceamento dinâmico. Compara-se os diferentes métodos e as situações as quais eles se adequam. A análise proporciona uma visão abrangente das opções disponíveis, auxiliando na escolha da técnica mais apropriada para diversas aplicações.

Palavras-chave: Balanceamento de Árvores; AVL; B-Tree; Rubro-Negra

Abstract: This article analyzes the main tree balancing techniques, focusing on the B-tree, AVL, and Red-Black methods, which are widely used in data structures to optimize search, insertion, and deletion operations. In addition to discussing the characteristics and applications of each method, the study introduces Treap and Splay trees as alternatives for dynamic balancing. The different methods are compared, along with the situations to which they are best suited. The analysis provides a comprehensive overview of the available options, helping in the selection of the most appropriate technique for various applications.

Keywords: Tree Balancing; AVL; B-Tree; Red-Black

1. Introdução

O balanceamento de árvores é um tópico fundamental na ciência da computação, principalmente na área de estruturas de dados. Este conceito é desempenha um papel muito importante para otimizar operações como inserção, busca e remoção, garantindo que elas sejam realizadas de forma eficiente, mesmo em grandes volumes de dados. Entre as diversas técnicas de balanceamento, destacam-se três métodos: B-tree, AVL e Red-Black. Cada um deles oferece diferentes abordagens para manter a árvore balanceada, influenciando diretamente a complexidade temporal das operações realizadas. Neste artigo, será explorada as características e aplicações desses métodos.

[1]Aluna do Instituto Mauá de Tecnologia – e-mail: 22.00721-0@maua.br

2. Desenvolvimento

2.1 Árvores

As árvores são um tipo de estrutura de dados hierárquica não linear que é formada por nós ligados por arestas. Cada nó tem um valor ou dados, além disso, ele pode ou não ter filhos – que são outros nós que nascem a partir do nó em questão. O primeiro nó da árvore chama-se raiz, se esse nó é conectado a outro, ele passa a ser um nó pai e o nó conectado passa a ser o nó filho. Cada vez que um nó tem filho, é gerado uma sub-árvore. Ademais, as arestas são importantes pois gerenciam a relação entre os nós. Por último, tem-se as folhas – que são nós que não possuem filhos. Outros conceitos relevantes são a altura e a profundidade. A altura da árvore é o tamanho do caminho mais longo até a folha, enquanto a profundidade de um nó é o tamanho do caminho que chega do nó até a raiz. Esses conceitos podem ser vistos nas figuras 1 e 2.

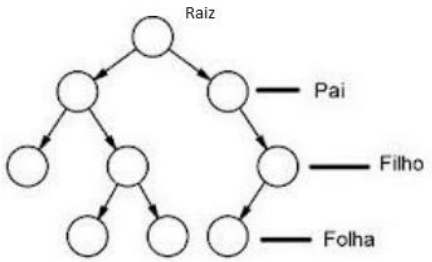


Figura 1 – Relacionamento entre nós

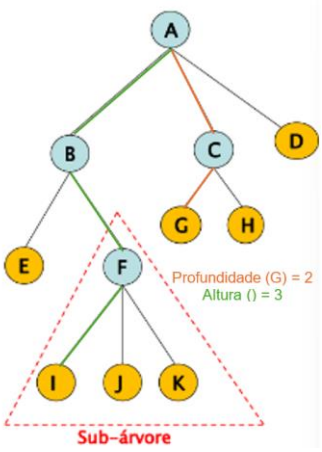


Figura 2 – Exemplo de sub-árvore, altura e profundidade

Existem alguns tipos de árvores como as árvores binárias, ordenadas, isomorfas, cheias, AVL etc. Dentre elas, destacam-se as árvores binárias, que são árvores cuja cada nó pode ter no máximo dois filhos representada pela figura 3. Além disso, esse tipo de árvore está associado a aplicações como expressões aritméticas, tomada de decisão e buscas.

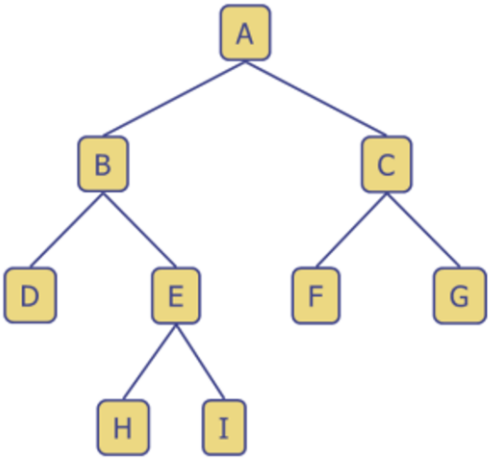


Figura 3 – Exemplo de árvore binária

Dentro das árvores binárias existe outra categoria que são as árvores binárias de pesquisa. A diferença desse tipo para a binária comum é que nela existe uma relação entre as chaves do nó, além de não poder haver chaves repetidas. Uma das organizações possíveis foi desenvolvida por Niklaus Wirth. A regra criada estabelece que os filhos a esquerda precisam sempre ter um menor valor que o pai, enquanto os da direita tem valor maior que o pai (figura 4).

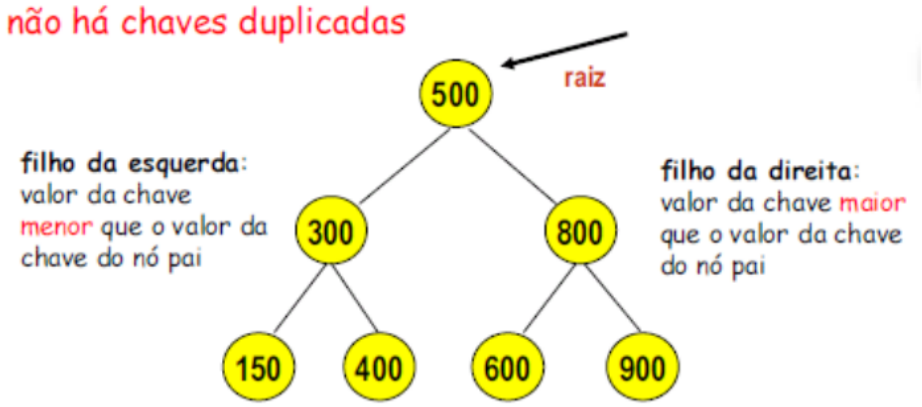


Figura 4 – Representação da regra de Niklaus Wirth numa árvore binária de pesquisa

2.2 Balanceamento de árvores

Primeiramente, é válido definir o que seria balanceamento. Na área da estrutura de dados, o balanceamento refere-se ao processo de manter a eficiência em termos de tempo em estruturas hierárquicas.

Especificamente para as árvores, o balanceamento se refere a manter a árvore equilibrada de forma que a altura da árvore não se torne desproporcionalmente grande em quantidade de nós. Esse

balanceamento é importante para que o tempo execução não degrade de logarítmico para linear já que geralmente as operações são proporcionais à altura da árvore. As figuras 5 e 6 mostram a diferença entre uma árvore balanceada e uma desbalanceada.

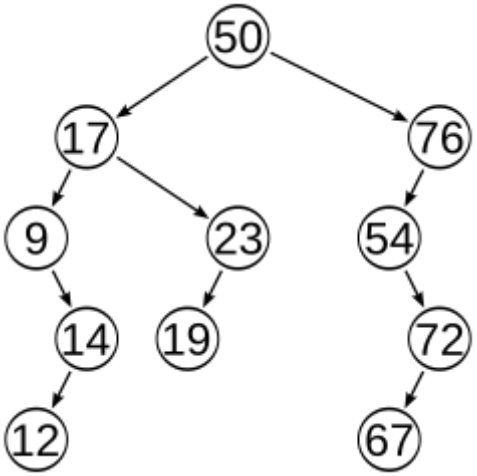


Figura 5 – Árvore desbalanceada

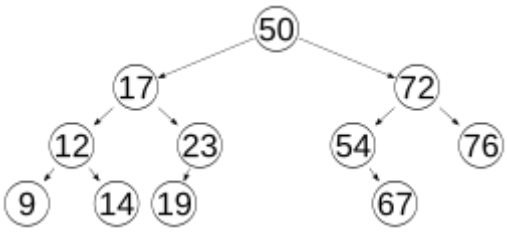


Figura 6 – Árvore da figura 5 após o balanceamento

Ademais, a fim de realizar esse balanceamento foram criados alguns métodos que serão apresentados na secção a seguir.

2.3 Métodos de balanceamento

Os métodos de balanceamento incluem, mas não se limitam a:

- ✓ Árvores Binárias de Pesquisa (BST) Balanceadas: que têm regras específicas para manter a árvore balanceada após inserções e exclusões, por exemplos as árvores AVL e as árvores Rubro-Negro,
- ✓ Árvores B e B+: que garantem que todas as folhas estejam no mesmo nível, ajudando a manter a árvore balanceada.

Serão destacados os métodos mais comuns como AVL, Rubro-Negro e B-Tree.

2.3.1 Árvore AVL

Essa árvore é dita como uma árvore binária de pesquisa auto balanceada em que a diferença de altura entre as sub-árvores da direita e da esquerda de qualquer nó é no máximo um, essa diferença ganhou o nome de Fator de Balanceamento (FB). Nas figuras 7 e 8 têm-se exemplos de árvores AVL e não AVL.

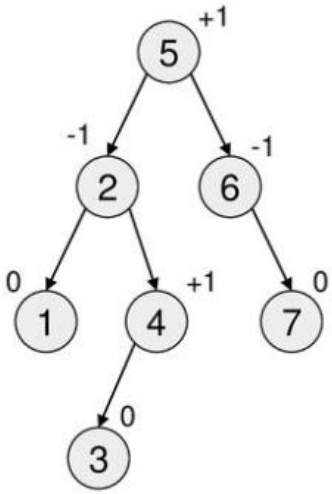


Figura 7 – Árvore AVL, dado que todos o FB variam apenas entre -1, 0 e 1

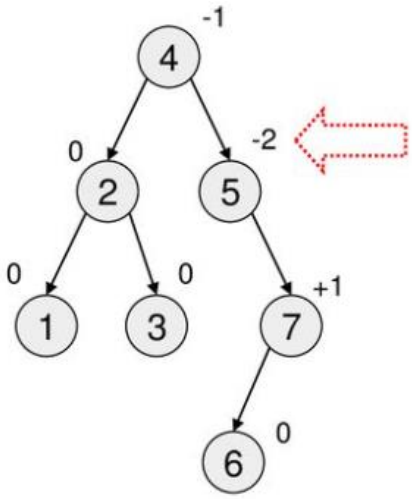


Figura 8 – Árvore não AVL, dado que como indicado pela seta o FB de um dos nós não se encontra na faixa esperada de -1 a 1

O balanceamento AVL é feito por meio de rotações seguindo algumas condições. Essas rotações podem ser simples ou duplas tanto para a esquerda quanto para a direita, assim existindo quatro tipos de rotação. As condições de cada tipo de rotação estão listadas e exemplificadas abaixo.

Rotação Simples à Direita: Ocorre sempre que um nó tem FB positivo (+2) e seu filho à esquerda também tem FB positivo (+1). Nesse caso, o filho à esquerda (P) assume a posição do nó desbalanceado (Q), de forma que o filho à direita de P passa a ser filho à esquerda de Q e Q se torna filho à direita de P. Exemplo na figura 9.

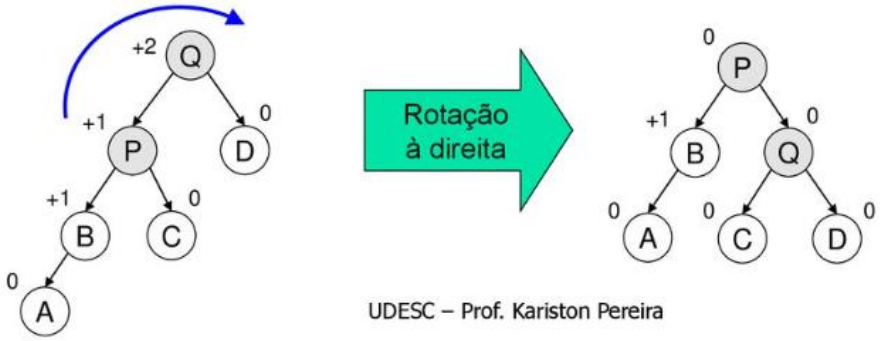


Figura 9 – Exemplo de Rotação Simples à Direita

Rotação Simples à Esquerda: Ocorre sempre que um nó tem FB negativo (-2) e seu filho à direita também tem FB negativo (-1). Nesse caso, o filho à direita (P) assume a posição do nó desbalanceado (Q), de forma que o filho à esquerda de P passa a

ser filho à direita de Q e Q se torna filho à esquerda de P. Exemplo na figura 10.

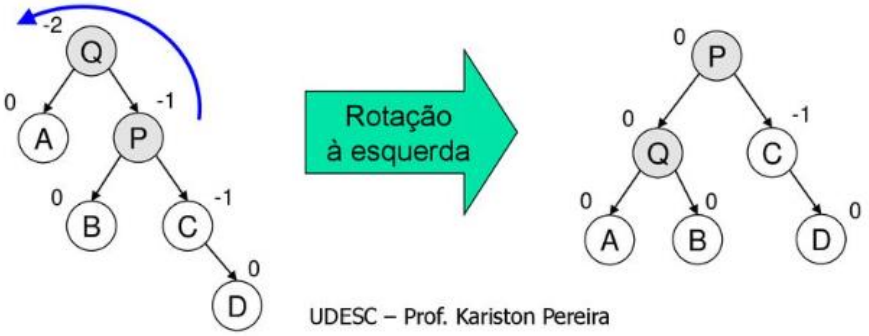


Figura 10 – Exemplo de Rotação Simples à Esquerda

Rotação Dupla à Direita: Ocorre sempre que um nó tem FB positivo (+2) e seu filho à esquerda tem FB negativo (-1). Nesse caso, precisa ser feita uma rotação à esquerda no nó filho da esquerda e uma rotação à direita no próprio nó. Exemplo na figura 11.

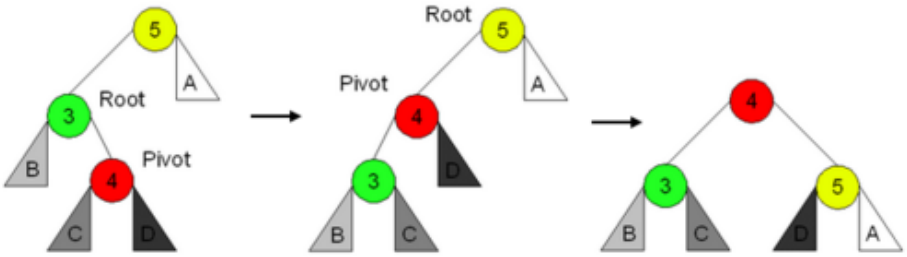


Figura 11 – Exemplo de Rotação Dupla à Direita

Rotação Duplo à Esquerda: Ocorre sempre que um nó tem FB (-2) negativo e seu filho à direita tem FB positivo (+1). Nesse caso, precisa ser feita uma rotação à direita no nó filho da direita e uma rotação à esquerda no próprio nó. Exemplo na figura 12.

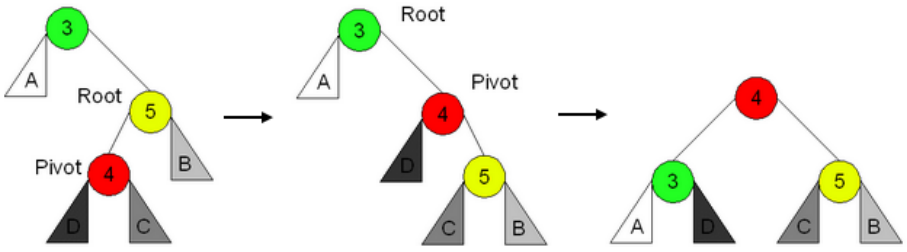


Figura 12 – Exemplo de Rotação Dupla à Esquerda

Por fim, as árvores AVL são bastante aplicadas em sistemas em que a eficiência da busca é fundamental e em sistemas que a árvore se modifica com frequência seja por inserção ou exclusão de nós como banco de dados e sistemas de arquivos.

2.3.2 Árvore Rubro-Negra

Esse tipo de árvore é outro tipo de árvore binária auto balanceada, porém essa árvore possui um campo extra para a cor, que tem como objetivo manter a árvore aproximadamente balanceada, já que ela garante que o maior caminho entre a raiz e as folhas seja no máximo o dobro do menor caminho. Essa árvore apresenta cinco propriedades:

1. Cada nó dessa árvore é colorido de vermelho ou preto;
2. A raiz é sempre preta;
3. Todas as folhas – nós nulos – são pretos;
4. Se um nó é vermelho, seus dois filhos e seu pai são obrigatoriamente pretos (não se pode ter dois nós vermelhos consecutivos).
5. Todo caminho de um nó até suas folhas contém a mesma quantidade de nós pretos – esse caminho é chamado de black Heights.

Essa árvore pode ser exemplificada pela figura 13.

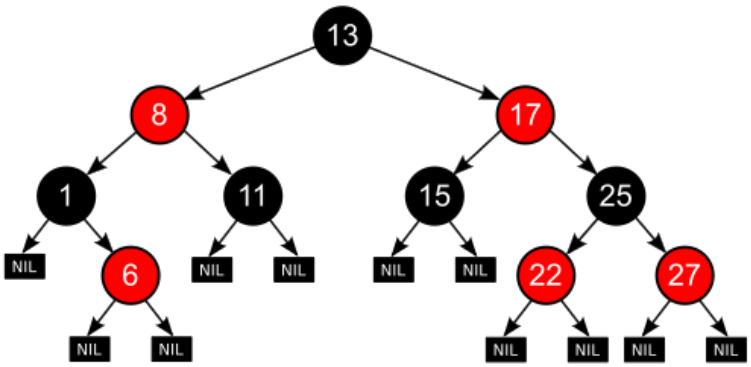


Figura 13 – Exemplo de Árvore Rubro-Negro

O balanceamento dessa árvore é feito a partir da troca de cor dos nós e das mesmas rotações da árvore AVL, porém essas rotações e trocas de cor só ocorrem para que os cinco critérios listados acima sempre sejam correspondidos.

Cada vez que um nó é inserido, ele é colorido de vermelho e a árvore é reavaliada e, caso as propriedades não sejam respeitadas, são feitas rotações ou recolorimento dependendo da situação. Já na exclusão de um nó, a árvore também é reavaliada e corrigida.

São tratados os quatro tipos de desbalanceamento representados na figura 14.

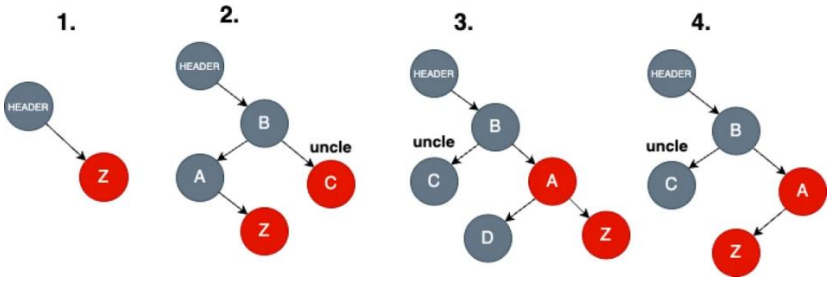
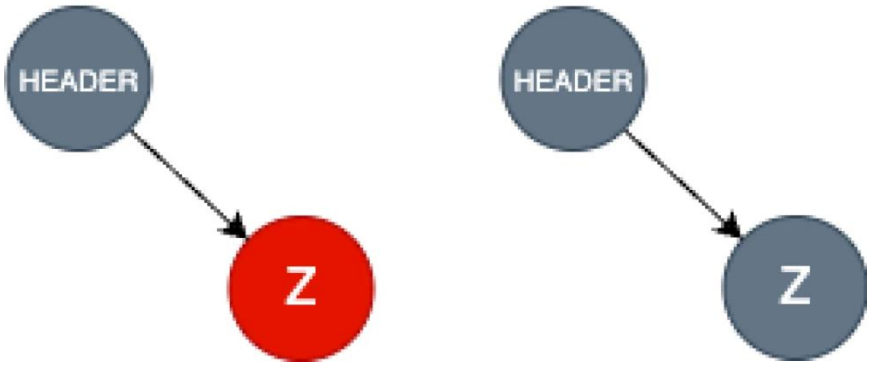


Figura 14 – Tipos de desbalanceamento
(Header aponta para a raiz)

Os quatro casos serão listados mais a fundo abaixo:

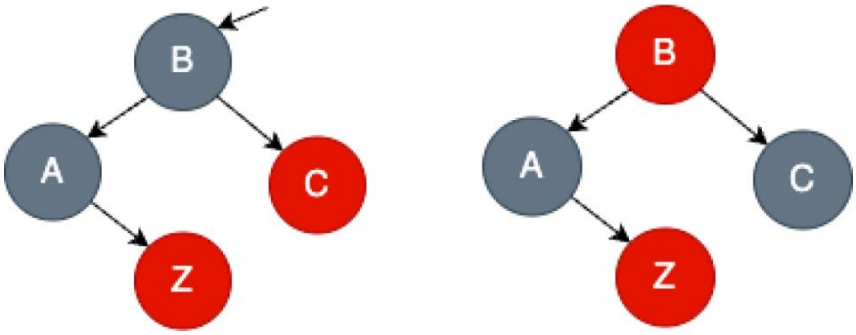
Caso 1 (Z é a raiz): para que as propriedades sejam respeitadas, basta mudar a cor de Z para preto como na figura 15.



(a) Antes (b) Depois

Figura 15 – Solução Caso 1
(Header aponta para a raiz)

Caso 2 (tio de Z é vermelho): para que as propriedades sejam respeitadas, pinta-se o avô de vermelho e o tio de preto como na figura 16. Se o avô for a raiz, pinta-se a raiz de preto novamente.

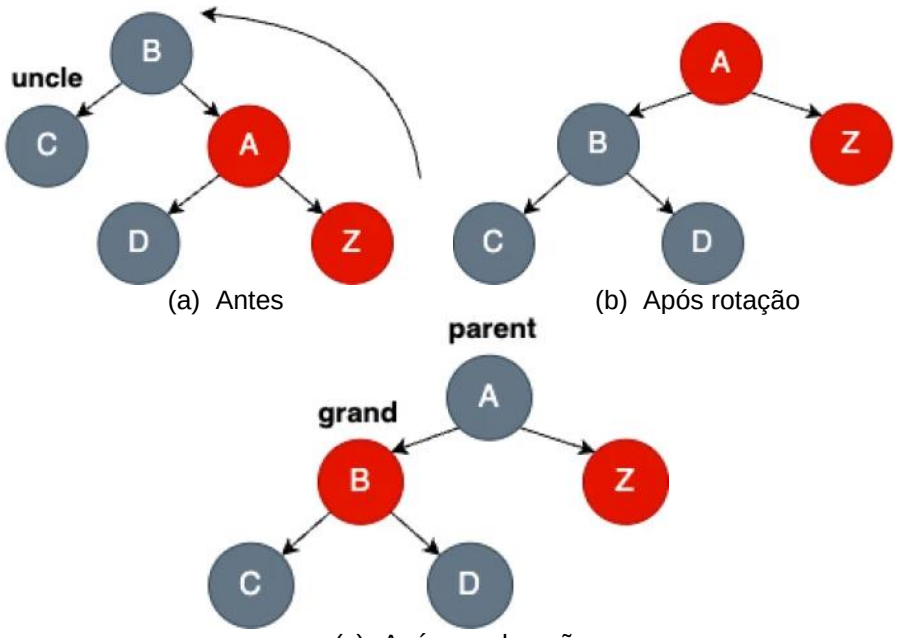


(a) Antes (b) Depois

Figura 16 – Solução Caso 2
(note que B não é raiz, não tem Header apontando para ele)

Caso 3 (tio de Z é preto formando uma linha): para que as propriedades sejam respeitadas, rotaciona-se para a esquerda,

pinta-se o “antigo” avô de vermelho e o pai de preto como na figura 17.

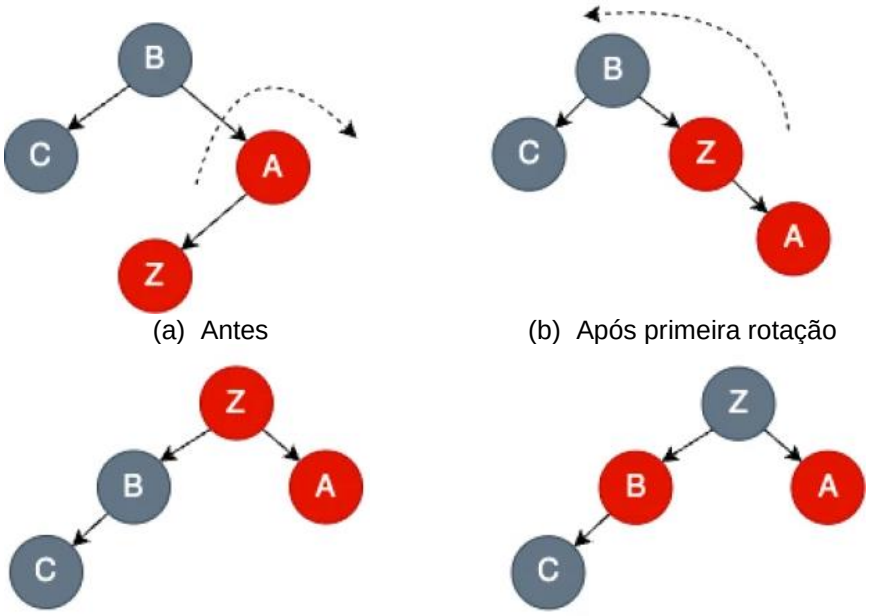


(a) Antes (b) Após rotação

(c) Após recoloração

Figura 17 – Solução Caso 3
(note que B não é raiz, não tem Header apontando para ele)

Caso 4 (tio de Z é preto formando um triângulo): para que as propriedades sejam respeitadas, rotaciona-se primeiro para a direita, depois para a esquerda e por fim, pinta-se o “antigo” avô de vermelho e o pai de preto como na figura 18.



(a) Antes (b) Após primeira rotação

(c) Após segunda rotação (d) Após recoloração

Figura 18 – Solução Caso 4
(note que B não é raiz, não tem Header apontando para ele)

Ademais, existem variações desses quatro casos base nas quais é preciso se atentar a rotação seguindo as não só a regra das cores, e sim na formação de uma linha ou um triângulo, utilizando as quatro rotações de AVL.

Por fim, a Árvore Rubro-Negra é também pode ser aplicada a banco de dados e em sistemas operacionais para gerenciar processos e memória.

2.3.3 Árvores B-Tree

Esse tipo de árvore é uma estrutura de dados projetada para sistemas que lidam com grande quantidade de dados. As B-trees, diferentemente dos outros dois tipos, não são binárias, isso é, permitem que que os nós tenham vários filhos. Cada nó – também chamado de página – pode ter várias chaves e filhos, além disso, as chaves de cada nó estão em ordem crescente. Uma B-tree balanceada tem todas suas folhas no mesmo nível, como representado na figura 19.

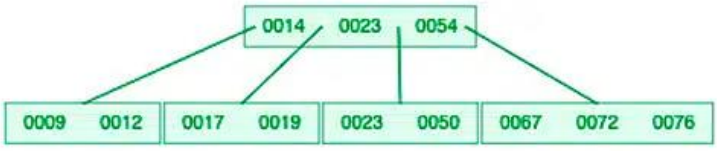


Figura 19 – Exemplo de B-Tree

Uma das características mais importantes desse tipo de árvore é que todos os caminhos das folhas até a raiz tem o mesmo comprimento, ou seja, a árvore é perfeitamente balanceada. Ademais, o número máximo de chaves em cada nó da página é restrito de forma que ao escolher um número par M, a árvore será organizada para que cada nó tenha no máximo M-1 chaves e no mínimo M/2 chaves.

Dessa forma, para realizar inserção e manter a árvore balanceada existem duas situações:

Página folha está com menos chaves que o máximo permitido: nesse caso, a inserção é feita de forma ordenada na página.

Página folha completa (com as M-1 chaves preenchidas): nesse caso, é necessário separar a página – operação de split.

Para isso, segue-se os seguintes passos: identificar qual chave fica no meio da sequência ordenada já considerando a nova chave; separar as chaves de forma que as menores continuam na página original e as maiores vão para uma nova página; por fim a página do meio sobe para a página pai e esse processo se repete até que a árvore esteja balanceada.

A figura 20 abaixo mostra a criação de uma árvore com chaves de 1 a 7 com no máximo três filhos.

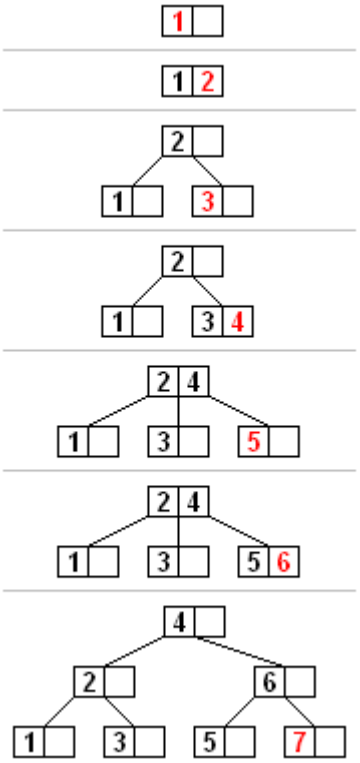


Figura 20 – Exemplo de B-Tree

Também existem variações desse tipo de árvore com a B+ e a B*. A B+ é uma variação em que todos os valores ficam armazenados nas folhas e as folhas são ligadas entre si, como representada na figura 21. Já as B* são uma melhoria da B+, que força nós a estarem bastante preenchidos antes de dividir, gerando uma árvore mais compacta, como na figura 22.

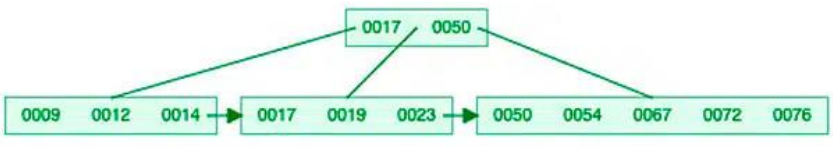


Figura 21 – Exemplo B+

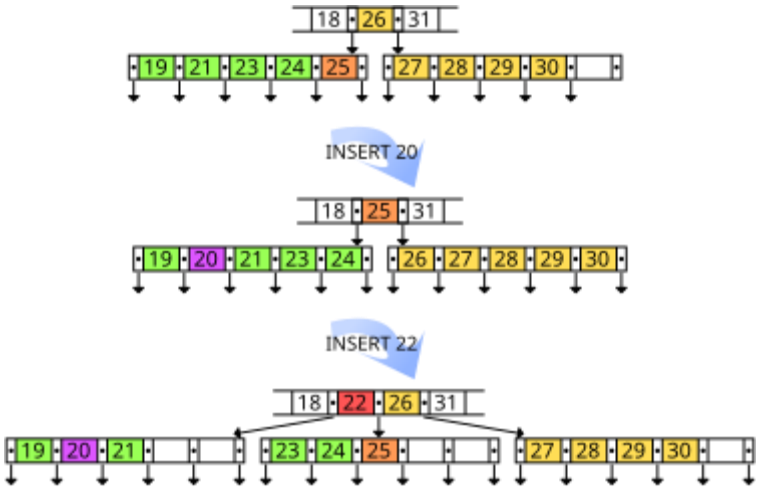


Figura 22 – Exemplo B*

Por fim, esse sistema é muito utilizado em banco de dados e sistema de arquivos, principalmente em disco.

2.3.4 Outros métodos de balanceamento

2.3.4.1 Árvores Splay

As árvores splay é uma estrutura de dados que mantém os itens mais acessados mais próximo a raiz para mais fácil acesso. Por essa razão, esse tipo de árvore é bastante aplicado em políticas de cache.

Para manter essa propriedade, é necessário fazer rotações. Existem seis tipos de rotação que serão listados abaixo.

Zig: quando o nó é filho esquerdo da raiz, aplica-se uma rotação a direita, como na figura 23.

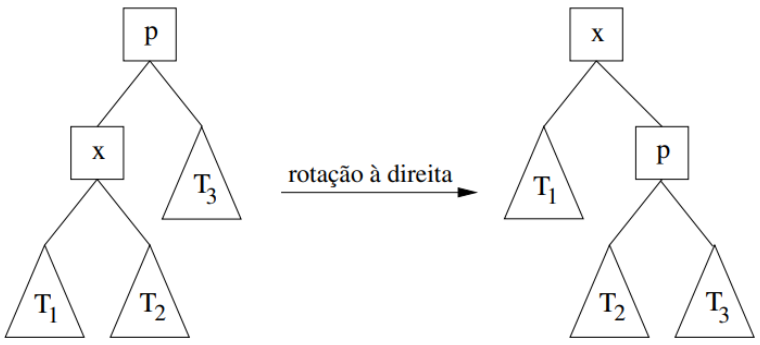


Figura 23 – Situação de Zig

Zag: quando o nó é filho direito da raiz, aplica-se uma rotação a esquerda, como na figura 24.

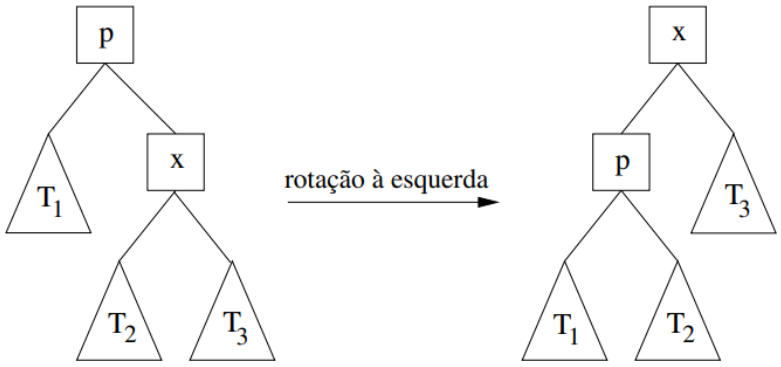


Figura 24 – Situação de Zag

Zig-Zig: quando o nó é filho esquerdo e o pai também é filho esquerdo, aplica-se duas rotações para a direita, como na figura 25.

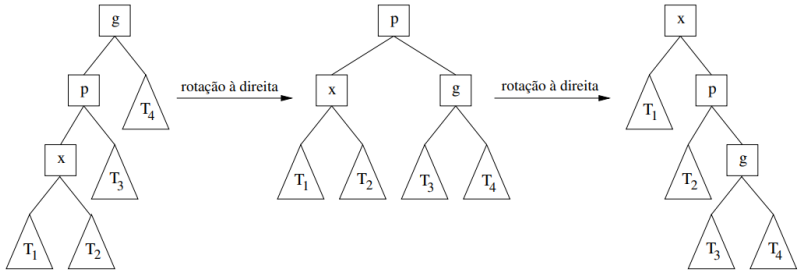


Figura 25 - Situação de Zig-Zig

Zag-Zag: quando o nó é filho direito e o pai também é filho direito, aplica-se duas rotações para a esquerda, como na figura 26.

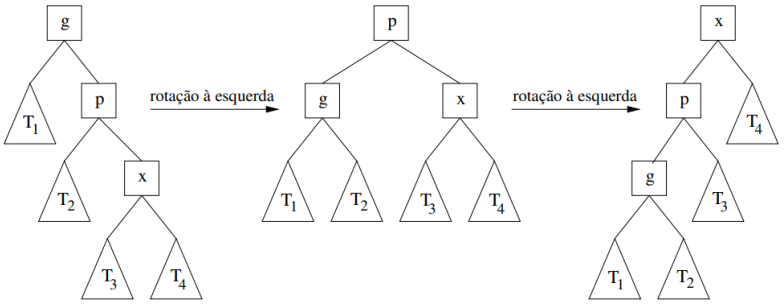


Figura 26 – Situação de Zag-Zag

Zig-Zag: quando o nó é filho esquerdo e o pai é filho direito, aplica-se primeiro uma rotação a direita e depois uma rotação a esquerda, como na figura 27.

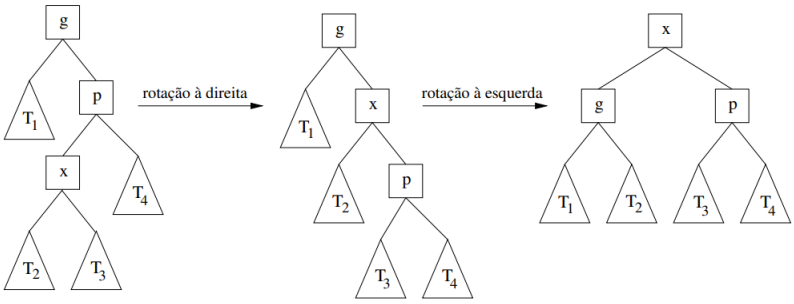


Figura 27 – Situação de Zig-Zag

Zag-Zig: quando o nó é filho direito e o pai é filho esquerdo, aplica-se primeiro uma rotação a esquerda e depois uma rotação a direita, como na figura 28.

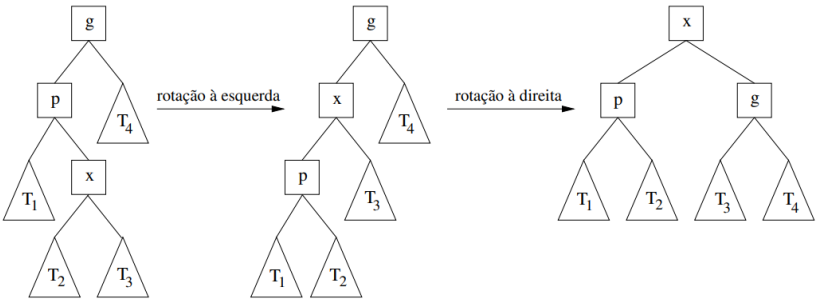


Figura 28 – Situação de Zag-Zig

2.3.4.2 Árvores Treap

As árvores treap são uma combinação de árvore de busca binária com heap, que os nós têm um campo extra indicando a prioridade. A principal característica é que todos os pais devem ter obrigatoriamente prioridade maior que a de seus filhos, isso pode ser visto na figura 29.

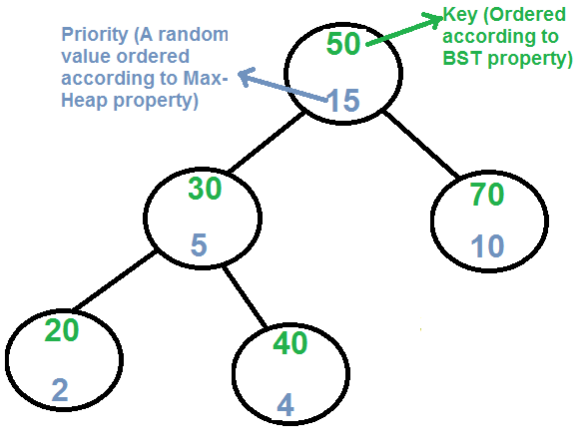


Figura 29 – Exemplo de Treap

A inserção de um nó segue as regras da árvore de busca binária e o balanceamento segue as rotações de um heap máximo. Essa árvore é aplicada apenas onde o balanceamento probabilístico é aceitável, como alguns algoritmos de ordenação e gerenciamento de dados.

3. Considerações Finais

Em suma, o balanceamento de árvores desempenha um papel crucial na eficiência das operações em estruturas de dados, e os métodos B-tree, AVL e Red-Black representam maneiras diferentes para manter essa eficiência. A B-tree se destaca em aplicações que utilizam muito o armazenamento em disco, enquanto as árvores AVL são ótimas em ambiente que demandam buscas rápidas. Por outro lado, as árvores Rubro-Negras oferecem um balanceamento mais flexível, resultando em inserções e deleções mais eficientes em comparação com as AVL.

Além dessas formas usuais, é importante mencionar também as árvores Treap e Splay, que são alternativas para o balanceamento dinâmico. As árvores Treap combinam as propriedades de uma árvore binária de busca com as de uma heap, oferecendo balanceamento probabilístico que pode ser vantajoso em certas aplicações. Já as árvores Splay, através de operações de splaying, garantem que os elementos recentemente acessados permaneçam próximos à raiz, o que pode melhorar o desempenho em cenários onde o acesso é não uniforme.

Portanto, escolha do método mais adequado depende do problema em questão e dos requisitos de desempenho.

4. Referências Bibliográfica

Árvores balanceadas. [s.l: s.n.]. Disponível em: <https://docente.ifrn.edu.br/robinsonalves/disciplinas/estruturas-de-dados/arvoreAVL.pdf>. Acesso em: 16 ago. 2024.

Árvores Rubro Negra. Disponível em: <https://pt.slideshare.net/skosta/rvores-rubro-negra>. Acesso em: 16 ago. 2024.

Árvores “Splay”. [s.l: s.n.]. Disponível em: <https://paginas.fe.up.pt/~rossetti/rrwiki/lib/exe/fetch.php?media=teaching:0910:aeda:aeda0910.18_splay.pdf>. Acesso em: 22 ago. 2024.

ÁRVORES, R.; NEGRAS. INF 1010 Estruturas de Dados Avançadas. [s.l: s.n.]. Disponível em: <https://www.inf.puc-rio.br/~noemi/eda-19.1/arvoresVN.pdf>. Acesso em: 16 ago. 2024.

Árvores: estrutura de dados - Algol.dev - com ilustrações e animações. Disponível em: <https://algol.dev/arvores-estrutura-de-dados/>. Acesso em: 16 ago. 2024.

AULA. Melhores momentos. [s.l: s.n.]. Disponível em: <https://www.ime.usp.br/~cris/aulas/19_2_121/slides/aula23.pdf>. Acesso em: 22 ago. 2024.

B-Árvores Siang Wun Song -Universidade de São Paulo -IME/USP MAC 5710 - Estruturas de Dados -2008. [s.l: s.n.]. Disponível em: <https://www.ime.usp.br/~song/mac5710/slides/09bt.pdf>. Acesso em: 16 ago. 2024.

B-Tree Visualization. Disponível em: <https://www.cs.usfca.edu/~galles/visualization/BTree.html>. Acesso em: 16 ago. 2024.

CORMEN, T.H. et al. Algoritmos: Teoria e Prática (Caps. 13). Campus. 2002

DOS, C. Árvore AVL. Disponível em: <https://pt.wikipedia.org/wiki/%C3%81rvore_AVL#:~:text=0%2F0).->. Acesso em: 16 ago. 2024.

DOS, C. Árvore B. Disponível em: <https://pt.wikipedia.org/wiki/%C3%81rvore_B>. Acesso em: 16 ago. 2024.

DOS, C. Árvore binária de busca balanceada. Disponível em: <https://pt.wikipedia.org/wiki/%C3%81rvore_bin%C3%A1ria_de_busca_balanceada>. Acesso em: 16 ago. 2024.

Estrutura de Dados: tipos de busca, árvore e tipos de árvores. Disponível em: <https://luizladeira.wordpress.com/2021/02/21/estrutura-de-dados-tipos-de-busca-arvore-e-tipos-de-arvores/>. Acesso em: 16 ago. 2024.

Estruturas de Dados: Árvores B. Disponível em: <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/B-trees.html>. Acesso em: 16 ago. 2024.

Estruturas de Dados: Conhecendo e dominando AVL Tree / Árvore AVL - Tiago Tartari. Disponível em: <https://tiagotartari.net/estruturas-de-dados-conhecendo-e-dominando-avl-tree-arvore-avl.html#:~:text=Balanceamento%2C%20no%20contexto%20de%20%C3%A1rvores>. Acesso em: 16 ago. 2024.

LANDIS, K. P., Exclusão, Prof, Rui Tramontin, Stout, Vel’skii. Árvores Binárias de Pesquisa e Balanceamento usando Árvores AVL - ppt carregar. Disponível em: <https://slideplayer.com.br/slide/14798097/>. Acesso em: 16 ago. 2024.

PEDRINI, H. Estruturas de Dados MC202. [s.l: s.n.]. Disponível em: <https://www.ic.unicamp.br/~helio/disciplinas/MC202/28_arvores_balanceadas_SPLAY.pdf>. Acesso em: 22 ago. 2024.

REALYTE. Árvore rubro-negra. Propriedades, princípios de organização, mecanismo de inserção. Disponível em: <https://javarush.com/pt/groups/posts/pt.4165.rvore-rubro-negra-propriedades-princpios-de-organizao-mecanismo-de-insero>. Acesso em: 16 ago. 2024.

Red/Black Tree Visualization. Disponível em: <https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>. Acesso em: 16 ago. 2024.

SONG, S. Árvore Rubro-Negra Siang Wun Song -Universidade de São Paulo - IME/USP MAC 5710 -Estruturas de Dados -2008. [s.l: s.n.]. Disponível em: <https://www.ime.usp.br/~song/mac5710/slides/08rb.pdf>. Acesso em: 16 ago. 2024.

Tudo o que você precisa saber sobre estruturas de dados em árvore. Disponível em: <https://www.freecodecamp.org/portuguese/news/tudo-o-que-voce-precisa-saber-sobre-estruturas-de-dados-em-arvore/>. Acesso em: 16 ago. 2024.