



UNIVERSIDADE FEDERAL DE SÃO PAULO
Instituto de Ciência e Tecnologia
Bacharelado Interdisciplinar em Ciência e Tecnologia

BALANCEAMENTO DE ÁRVORES

Árvores AVL, Rubro-Negras e B

Trabalho apresentado como recuperação da disciplina
de Algoritmos e Estruturas de Dados I

Autora:
Gabriely Di Folco Rocha

São José dos Campos
2026

1 Introdução

Estruturas de dados baseadas em árvores desempenham um papel central no projeto de algoritmos eficientes, especialmente quando se deseja realizar operações de busca, inserção e remoção de forma rápida. Entre essas estruturas, as árvores de busca binária (Binary Search Trees – BSTs) ocupam posição de destaque por sua simplicidade e eficiência média. No entanto, seu desempenho depende fortemente da forma da árvore, podendo degradar-se significativamente em situações desfavoráveis.

Quando uma árvore de busca binária degenera em uma estrutura linear, suas operações passam a ter custo linear, comprometendo a eficiência esperada. Surge, portanto, a necessidade de mecanismos que garantam que a altura da árvore permaneça proporcional a $\log n$, independentemente da ordem de inserção dos elementos. Esse é o objetivo das chamadas árvores balanceadas.

Este trabalho apresenta os principais conceitos relacionados ao balanceamento de árvores de busca, com ênfase em árvores AVL, árvores rubro-negras e árvores B. Busquei referências da literatura (Cormen, Knuth e Weiss), de artigos da Wikipedia e de slides da disciplina de Algoritmos e Estruturas de Dados II do IC-Unicamp, coursera e IME-USP.

2 Árvores de Busca e o problema do desbalanceamento

Uma árvore de busca binária é definida pela propriedade de que, para cada nó, todos os elementos da subárvore esquerda são menores que sua chave, e todos os da subárvore direita são maiores. Essa propriedade permite realizar buscas de forma semelhante à busca binária em vetores ordenados.

Entretanto, diferentemente de vetores, a forma da árvore depende da ordem de inserção dos elementos. Inserções ordenadas podem gerar uma árvore degenerada, cuja altura é n , fazendo com que operações de busca, inserção e remoção tenham custo $O(n)$. Abaixo, percebe-se que a árvore degenerada mais se aproxima de uma lista encadeada.

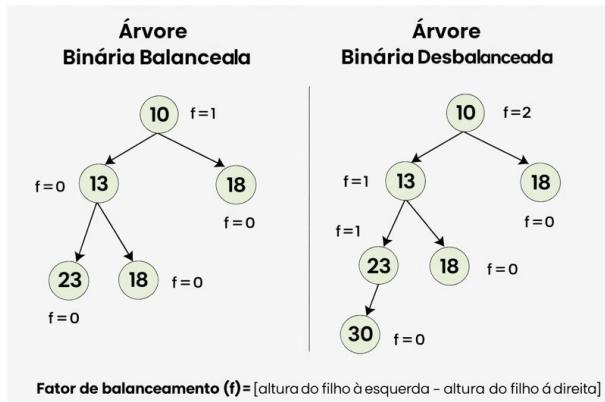


Figura 1: Exemplo de árvores quanto ao balanceamento.

Assim, a eficiência das árvores de busca está diretamente relacionada à sua altura. O objetivo das árvores平衡adas é garantir que essa altura seja mantida em $O(\log n)$, assegurando desempenho previsível mesmo no pior caso.

3 Árvores AVL

As árvores AVL, propostas por Adelson-Velsky e Landis, foram as primeiras estruturas de dados a implementar balanceamento automático em árvores binárias de busca. A ideia central consiste em restringir a diferença de altura entre as subárvores esquerda e direita de qualquer nó, de modo a impedir o crescimento degenerado da árvore.

Define-se o *fator de balanceamento* de um nó como a diferença entre as alturas de suas subárvores. Para que a árvore seja considerada AVL, esse fator deve assumir exclusivamente os valores -1 , 0 ou 1 . Sempre que uma inserção ou remoção viola essa condição, o balanceamento é restaurado por meio de rotações locais, que podem ser simples (à esquerda ou à direita) ou duplas (esquerda-direita ou direita-esquerda). A figura 2 ilustra a árvore mais desbalanceada à esquerda que se pode gerar com as novas regras.

Algoritmo 1 – Inserção em árvore AVL segundo Knuth

```

1: function INSERTAVL( $T, x$ )
   ▷ Etapa 1: inserção padrão de BST
2:   if  $T = \text{NULL}$  then
3:     return novo nó com chave  $x$ 
4:   end if
5:   if  $x < T.\text{chave}$  then
6:      $T.\text{esq} \leftarrow \text{INSERTAVL}(T.\text{esq}, x)$ 
7:   else
8:      $T.\text{dir} \leftarrow \text{INSERTAVL}(T.\text{dir}, x)$ 
9:   end if
   ▷ Etapa 2: atualização da altura
10:  Atualiza altura de  $T$ 
11:   $fb \leftarrow \text{altura}(T.\text{esq}) - \text{altura}(T.\text{dir})$ 
   ▷ Etapa 3: correção do balanceamento
12:  if  $fb > 1$  e  $x < T.\text{esq}.\text{chave}$  then
13:    return ROTACAO DIREITA( $T$ )
14:  end if
15:  if  $fb < -1$  e  $x > T.\text{dir}.\text{chave}$  then
16:    return ROTACAO ESQUERDA( $T$ )
17:  end if
18:  if  $fb > 1$  e  $x > T.\text{esq}.\text{chave}$  then
19:     $T.\text{esq} \leftarrow \text{ROTACAOESQUERDA}(T.\text{esq})$ 
20:    return ROTACAO DIREITA( $T$ )
21:  end if
22:  if  $fb < -1$  e  $x < T.\text{dir}.\text{chave}$  then
23:     $T.\text{dir} \leftarrow \text{ROTACAO DIREITA}(T.\text{dir})$ 
24:    return ROTACAOESQUERDA( $T$ )
25:  end if
   ▷ Caso direita–direita
   ▷ Caso esquerda–esquerda
   ▷ Caso esquerda–direita
   ▷ Caso direita–esquerda
26:  return  $T$ 
27: end function

```

Mesmo nesse pior caso, a diferença de altura entre as subárvore de cada nó permanece limitada aos valores $-1, 0$ ou 1 , o que impede o crescimento linear da árvore. O impacto desse balanceamento pode ser analisado observando-se o crescimento do número de nós em função da altura da árvore. Seja N_h o número mínimo de nós de uma árvore AVL de altura h . Para os primeiros valores, tem-se $N_0 = 1$, $N_1 = 2$ e, de forma geral,

$$N_h = 1 + N_{h-1} + N_{h-2}.$$

Essa recorrência é análoga à da sequência de Fibonacci, o que permite estabelecer uma

relação direta entre a altura da árvore e essa sequência. Em particular, vale que

$$N_h = F_{h+2} - 1,$$

onde F_i denota o i -ésimo número de Fibonacci. Utilizando a aproximação assintótica

$$F_i \approx \frac{\Phi^i}{\sqrt{5}}, \quad \text{com } \Phi = \frac{1 + \sqrt{5}}{2},$$

obtém-se

$$N_h \approx \frac{\Phi^{h+2}}{\sqrt{5}} - 1.$$

Reorganizando a expressão e aplicando logaritmos, conclui-se que a altura h da árvore cresce proporcionalmente ao logaritmo do número de nós:

$$h \leq 1,44 \log N.$$

Esse resultado mostra que o balanceamento imposto pelas árvores AVL controla rigidamente sua altura, impedindo degradações estruturais.

Como consequência direta, todas as operações fundamentais — busca, inserção e remoção — possuem complexidade $O(\log n)$ tanto no pior quanto no caso médio, conforme destacado por Weiss. A principal vantagem das árvores AVL reside justamente nessa garantia forte de altura mínima, que resulta em buscas ligeiramente mais rápidas. Em contrapartida, o custo adicional de rotações pode tornar inserções e remoções mais onerosas quando comparadas a estruturas com balanceamento mais flexível.

4 Árvores Rubro-Negras

As árvores rubro-negras constituem uma alternativa menos rígida ao balanceamento estrito das árvores AVL. Nelas, cada nó possui uma cor — vermelha ou preta — e obedece a um conjunto de propriedades que garantem uma altura limitada.

As principais propriedades são:

- Todo nó é vermelho ou preto;
- A raiz é sempre preta;
- Nenhum nó vermelho pode ter filho vermelho;
- Todo caminho de um nó até uma folha nula contém o mesmo número de nós pretos.

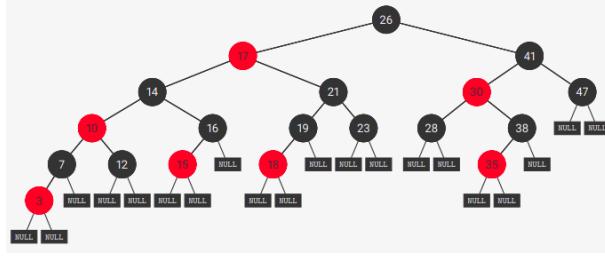


Figura 2: Exemplo de árvore rubro-negra esquerdistra, IC-Unicamp.

Essas restrições garantem que a altura da árvore seja, no máximo, $2 \log(n+1)$. Embora menos rigidamente balanceadas que as AVL, árvores rubro-negras realizam menos rotações durante atualizações, o que as torna eficientes na prática.

Altura e complexidade da árvore rubro-negra

- **Caso base ($bh = 0$):**

A árvore consiste apenas em uma folha `NULL`. Logo, possui exatamente $2^{bh} - 1 = 0$ nós não nulos.

- **Passo indutivo ($bh > 0$):**

Os filhos da raiz possuem altura-negra pelo menos $bh - 1$. Pela hipótese de indução, cada subárvore contém ao menos

$$2^{bh-1} - 1$$

nós não nulos. Somando as duas subárvores e o nó raiz, a árvore possui no mínimo

$$2(2^{bh-1} - 1) + 1 = 2^{bh} - 1$$

nós não nulos.

Portanto, uma árvore com altura-negra bh possui pelo menos $2^{bh} - 1$ nós não nulos. Em árvores rubro-negras, a altura-negra bh é pelo menos metade da altura total h da árvore, pois não existem dois nós vermelhos consecutivos em um caminho da raiz até uma folha.

Como o número de nós não nulos satisfaz

$$n \geq 2^{bh} - 1 \geq 2^{\frac{h}{2}} - 1,$$

segue que

$$h \leq 2 \lg(n + 1),$$

o que implica

$$h = O(\lg n).$$

Por essa razão, são amplamente utilizadas em implementações reais, como em bi-

bliotecas padrão de linguagens de programação e em sistemas operacionais. Conforme apresentado por Cormen et al., as operações de busca, inserção e remoção mantêm complexidade $O(\log n)$.

Algoritmo 2 — Inserção em árvore rubro-negra segundo Knuth

```

1: function FIXINSERTRB( $T, z$ )
2:   while  $z.pai$  é vermelho do
3:     if  $z.pai$  é filho esquerdo de  $z.avos$  then
4:        $y \leftarrow z.avos.dir$                                  $\triangleright$  tio de  $z$ 
5:       if  $y$  é vermelho then                                $\triangleright$  Caso 1: recoloração
6:         Recoloce  $z.pai$ ,  $y$  e  $z.avos$ 
7:          $z \leftarrow z.avos$ 
8:       else
9:         if  $z$  é filho direito then                       $\triangleright$  Caso 2: rotação preparatória
10:         $z \leftarrow z.pai$ 
11:        ROTACAOESQUERDA( $T, z$ )
12:      end if                                          $\triangleright$  Caso 3: rotação principal
13:      Recoloce  $z.pai$  e  $z.avos$ 
14:      ROTACAO DIREITA( $T, z.avos$ )
15:    end if
16:  end if
17: end while
18: Define a raiz como preta
19: end function
```

5 Árvores B e B+

Enquanto árvores AVL e rubro-negras são adequadas para memória principal, árvores B foram projetadas para ambientes onde o custo dominante é o acesso à memória secundária, como discos.

Nessas estruturas, cada nó pode armazenar múltiplas chaves e possuir vários filhos, reduzindo significativamente a altura da árvore. Isso diminui o número de acessos a disco necessários para realizar operações.

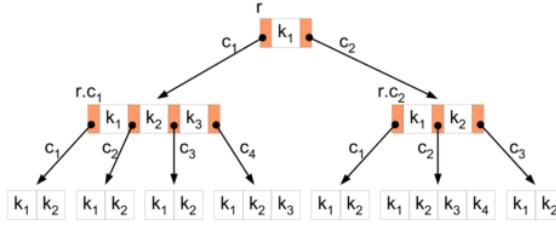


Figura 3: Exemplo de árvore B, Wikipedia.

Algoritmo 3 – Divisão de nó em árvore B segundo Knuth

```

1: function SPLITCHILD( $x, i$ ) ▷ Etapa 1: preparação
2:    $y \leftarrow x.filho[i]$ 
3:   Criar novo nó  $z$ 
4:    $z.n \leftarrow t - 1$  ▷ Etapa 2: mover chaves
5:   for  $j \leftarrow 1$  to  $t - 1$  do
6:      $z.chave[j] \leftarrow y.chave[j + t]$ 
7:   end for ▷ Etapa 3: mover filhos
8:   if  $y$  não é folha then
9:     for  $j \leftarrow 1$  to  $t$  do
10:       $z.filho[j] \leftarrow y.filho[j + t]$ 
11:    end for
12:  end if ▷ Etapa 4: ajustar nó pai
13:   $y.n \leftarrow t - 1$ 
14:  Deslocar filhos de  $x$  para abrir espaço
15:   $x.filho[i + 1] \leftarrow z$ 
16:  Deslocar chaves de  $x$ 
17:   $x.chave[i] \leftarrow y.chave[t]$ 
18:   $x.n \leftarrow x.n + 1$ 
19: end function

```

As árvores B mantêm todas as folhas no mesmo nível e garantem que cada nó (exceto a raiz) esteja parcialmente cheio, assegurando um bom aproveitamento de espaço. As árvores B+ são uma variação em que todas as chaves são armazenadas nas folhas, facilitando operações de varredura sequencial.

Segundo Cormen, essas estruturas são amplamente empregadas em sistemas de arquivos e bancos de dados devido à sua eficiência em ambientes de armazenamento secundário.

6 Comparação e considerações finais

Os algoritmos estudados constituem soluções fundamentais para o problema do desbalanceamento em árvores de busca binária, garantindo que operações de busca, inserção e remoção mantenham complexidade assintótica logarítmica mesmo no pior caso. Embora compartilhem esse objetivo, as estruturas analisadas diferem nas estratégias de平衡amento e nos compromissos assumidos entre desempenho e custo de manutenção.

As árvores AVL impõem um balanceamento rigoroso, limitando estritamente a altura da árvore e favorecendo buscas mais eficientes, à custa de um maior número de rotações em atualizações. As árvores rubro-negras adotam restrições mais flexíveis, preservando a altura em ordem logarítmica com menor custo de reestruturação, o que justifica sua ampla utilização em bibliotecas padrão e sistemas operacionais. As árvores B e B+, por sua vez, são projetadas para ambientes com armazenamento em disco, reduzindo a altura da árvore por meio de nós com múltiplas chaves e minimizando acessos à memória secundária.

A tabela a seguir apresenta uma comparação sintética entre essas estruturas:

Tabela 1: Comparação entre estruturas de árvores balanceadas

Estrutura	Altura	Busca	Inserção/Remoção	Aplicações típicas
AVL	$O(\log n)$	$O(\log n)$	$O(\log n)$	Sistemas com predominância de operações de busca
Rubro-negra	$O(\log n)$	$O(\log n)$	$O(\log n)$	Bibliotecas padrão, kernels e aplicações gerais
B / B+	$O(\log_m n)$	$O(\log_m n)$	$O(\log_m n)$	Bancos de dados e sistemas de arquivos

A escolha da estrutura mais adequada depende das restrições do sistema e do perfil de operações predominantes. Em aplicações com predominância de buscas, árvores AVL podem ser vantajosas. Em cenários gerais de software, árvores rubro-negras oferecem um equilíbrio eficiente entre desempenho e custo de manutenção. Em sistemas de grande escala baseados em armazenamento secundário, árvores B e B+ constituem a escolha natural.

Pelo que discutimos, percebe-se que o estudo das árvores balanceadas é essencial para evitar a degradação estrutural das árvores de busca binária e garantir desempenho previsível. O balanceamento de árvores ocupa, portanto, um papel central na formação em Algoritmos e Estruturas de Dados, ao evidenciar como escolhas estruturais impactam diretamente a eficiência dos algoritmos.

7 Referências bibliográficas

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to Algorithms*. 3. ed. Cambridge: MIT Press, 2009.
- WEISS, M. A. *Data Structures and Algorithm Analysis*. 3. ed. Boston: Pearson, 2012.
- KNUTH, D. E. *The Art of Computer Programming, Volume 3: Sorting and Searching*. 2. ed. Reading: Addison-Wesley, 1998.
- DAHAB, R. *Árvores Balanceadas*. Handout da disciplina MC202 – Estruturas de Dados, Instituto de Computação, Universidade Estadual de Campinas, 2019. Disponível em: https://www.ic.unicamp.br/~rdahab/cursos/mc202-old/2019-2s/Welcome_files/a16-arvores-balanceadas-handout.pdf. Acesso em: jan. 2026.
- PANDA IME-USP. *Desempenho de Árvores AVL*. Material didático online do Instituto de Matemática e Estatística da USP. Disponível em: https://panda.ime.usp.br/pythonds/static/pythonds_pt/06-Arvores/AVLTreePerformance.html. Acesso em: jan. 2026.
- SEDGEWICK, R.; WAYNE, K. *Algorithms, Part II*. Curso online em Algoritmos e Estruturas de Dados, Universidade de Princeton, plataforma Coursera. Disponível em: <https://www.coursera.org/learn/algorithms-part2>. Acesso em: jan. 2026.
- WIKIPÉDIA. *Árvore B*. Wikipédia, a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/%C3%81rvore_B. Acesso em: jan. 2026.