Algoritmos e Estruturas de Dados II

Capítulo da Aula 6: Árvores Binárias de Busca (BST)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br
CEFET-MG - Campus Timóteo

Fevereiro de 2026

1. O Colapso das Estruturas Lineares

Até o momento arquitetural do curso, operávamos restritos ao rotineiro limiar estrito de matrizes 1D enfileiradas e puramente lineares isoladas contíguas laterais transversais (um bloco atrás do outro), enraizadas nos Vetores ou nas Listas Encadeadas transversais. O engasgo fatal da estagnação vital computacional ocorre exatamente nos limites destas matrizes simples na hora do "rush"; quando o ecossistema estilhaçado sofre disparos exigindo buscas brutais de resgastes milissegundários em engarrafamentos de base de Big Data massivos gigantes reais pulsantes.

Para arrebentar e transpor as portas trancadas com a pesada e intolerante âncora de tempo linear contínuo do $O(N)$ limitador asfíxiador engessador da lista em cruzamentos, tracionamos a ciência da computação invadindo fronteiras complexas da matriz do terreno limpo atômico estrutural **Hierárquico**. As Árvores Binárias de Busca (BST - *Binary Search Trees*) despontam com uma proposta arquitetural hibrida e formidável base flexível: tentar amalgamar as inserções fluídas rápidas quebradas da conectividade dispersa dinâmica das **Listas Encadeadas** ao imponente poder destruidor limitante de funil rápido $\mathcal{O}(\log N)$ regido pelos saltos matemáticos cortados fracionários da matriz das estáticas brutas **Buscas Binárias**.

2. Definição Formal de BST

Uma árvore binária é uma **Árvore Binária de Pesquisa** se, e somente se, para **qualquer** nó `u`: 1. Todas as chaves na subárvore **Esquerda** de `u` são **menores** que `u.chave`. 2. Todas as chaves na subárvore **Direita** de `u` são **maiores** que `u.chave`. 3. As subárvores esquerda e direita também são BSTs.

Essa propriedade garante que, se percorrermos a árvore em ordem simétrica (**In-Order**), visitaremos os elementos de forma crescente.

▷ Exemplo

Exemplo de Validação: Considere uma árvore com Raiz=10.

- Se o filho esquerdo for 8, está OK ($8 < 10$).
- Se o filho direito for 15, está OK ($15 > 10$).
- Agora, se o filho direito (15) tiver um filho esquerdo 9... **Problema!** O 9 está na subárvore **direita** do 10, mas $9 < 10$. Isso viola a regra global: *todos* os descendentes à direita devem ser maiores que a raiz.

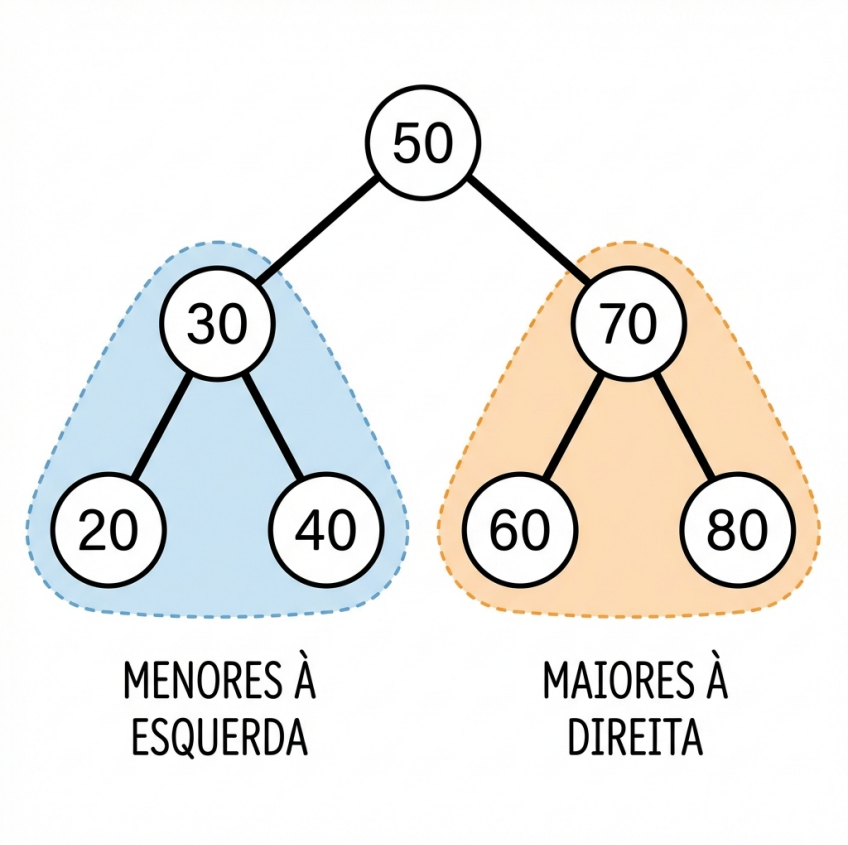


Figure 1: Propriedade da BST: à esquerda de um nó todos são menores; à direita, maiores.

3. Implementação em Java

Vamos definir a classe do Nó e a estrutura da BST.

```
1 public class BST {
2
3     private class Node {
4         int key;
5         Node left, right;
6
```

```
7           public Node(int item) {
8               key = item;
9               left = right = null;
10          }
11      }
12
13      private Node root;
14
15      public BST() {
16          root = null;
17      }
18
19      // Métodos públicos (API)
20      public void insert(int key) { root = insertRec(root, key);
            }
21      public boolean search(int key) { return searchRec(root, key
            ); }
22      public void delete(int key) { root = deleteRec(root, key);
            }
23  }
```

3.1. Inserção

A inserção sempre ocorre nas folhas (exceto se a chave já existir). Descemos pela árvore comparando: se é menor, vai pra esquerda; se maior, direita. Ao atingir `null`, inserimos.

▷ Exemplo

Rastreamento da Inserção: Vamos inserir 45 na árvore abaixo: Raiz: 50 / 30 70 1. Comparar 45 com 50: $45 < 50 \rightarrow$ vá para Esquerda. 2. Comparar 45 com 30: $45 > 30 \rightarrow$ vá para Direita. 3. O nó 30 não tem filho direito (é `null`). 4. **Inserir** 45 como filho direito de 30.

```
1  private Node insertRec(Node root, int key) {
2      // Caso Base: árvore vazia ou chegou na folha
3      if (root == null) {
4          return new Node(key);
5      }
6
7      // Case Recursivo: navegação
8      if (key < root.key)
9          root.left = insertRec(root.left, key);
10     else if (key > root.key)
11         root.right = insertRec(root.right, key);
12
13     // Retorna o nó (inalterado) para manter o encadeamento
14     return root;
15 }
```

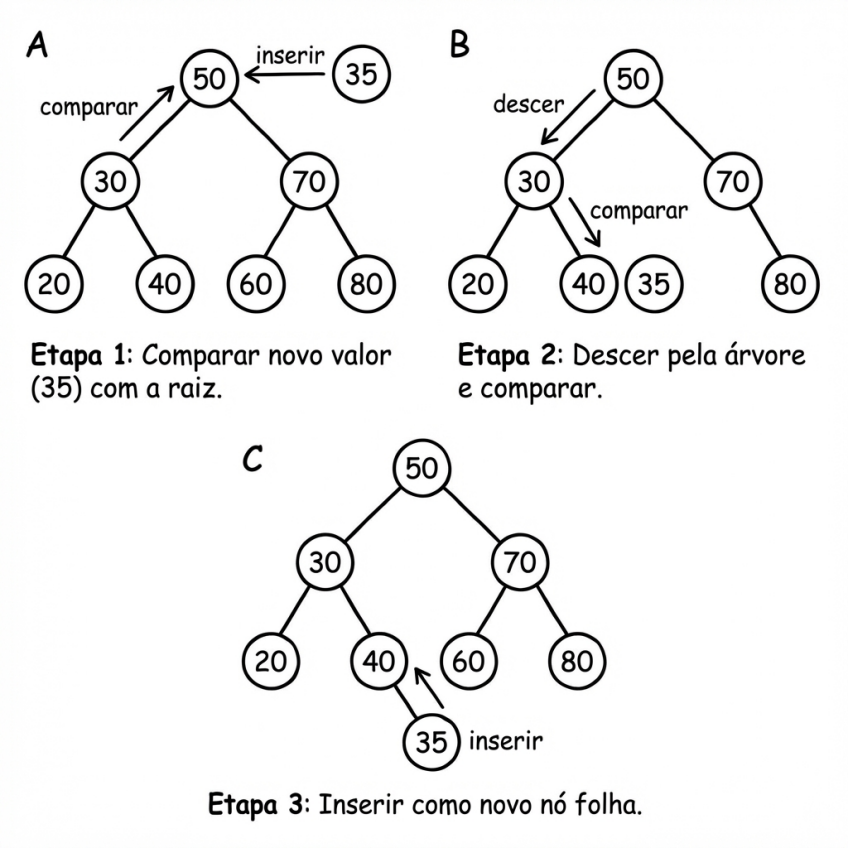


Figure 2: Inserção na BST: descer comparando até encontrar posição de folha.

3.2. Busca

A complexidade é proporcional à **altura** da árvore (h).

▷ Exemplo

Exemplo de Busca: Procurando o valor 60: 1. Raiz 50: $60 > 50 \rightarrow$ Direita. 2. Nó 70: $60 < 70 \rightarrow$ Esquerda. 3. Nó 60: $60 == 60 \rightarrow$ **Encontrado!**

- Melhor caso (balanceada): $O(\log N)$.
- Pior caso (degenerada em lista): $O(N)$.

```java
private boolean searchRec(Node root, int key) {
    if (root == null) return false;
    if (root.key == key) return true;

    if (key < root.key)
        return searchRec(root.left, key);
    else
        return searchRec(root.right, key);
}
```

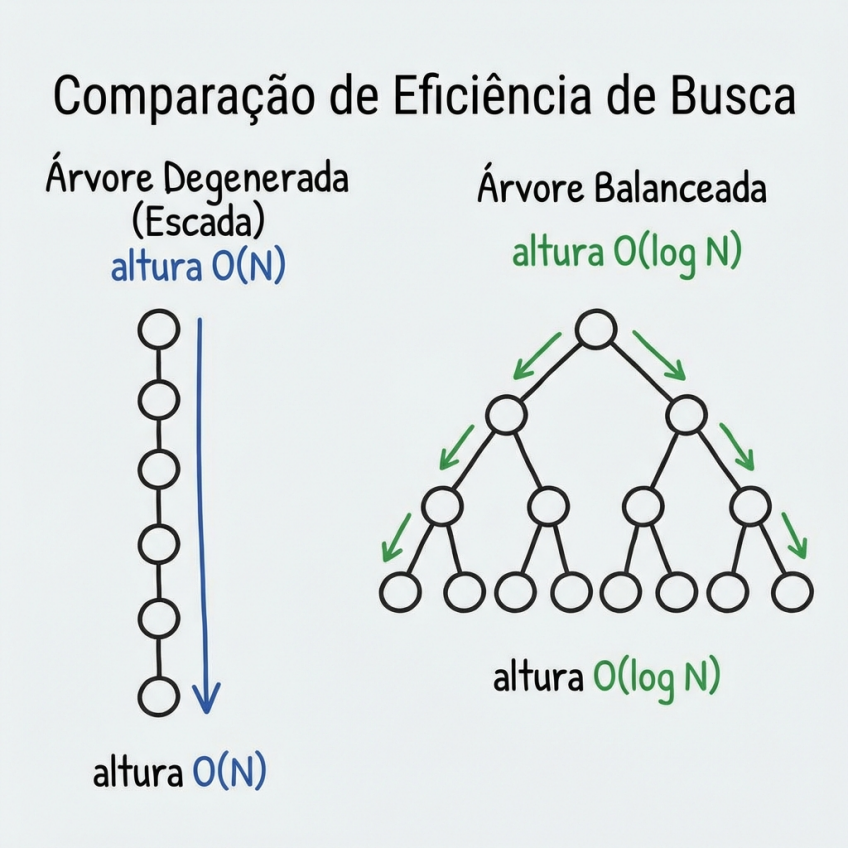


Figure 3: BST degenerada (lista) vs BST balanceada: impacto na altura e no tempo de busca.

4. A Engenharia da Deleção: O Abismo Estrutural

Extirpar e explodir a base estática enraizada conectiva logada das passagens vitais de dados cruzados estruturais e esvaziar pontes numa Árvore mantendo intactos cruzamentos engessados nos nós conectivos filhos remanescentes desponta disparadamente entre as reestruturações cruéis emaranhadas complexas críticas engessadas brutas cruas das passagens em algoritmos primários base. Classificamos triplamente intervenções de escopo:

1. **A Passividade Limpa (Nó Folha Morto terminal vazio cego nulo):** Extirpar a ponta alocando anulação limpa unicamente desvinculando limpo engate referencial raiz do ascendente nula sem remanescentes sem efeitos físicos posteriores passivos. 2. **O Salto Direto (O Engate Desamparado na Órfandade Unifilar Única):** Ao eliminar a matriz paternal local o fluxo resgata varre contínua contornando isolador o nó extraído soldando diretamente a aresta avulsa raiz cravando isoladoramente linear os descendentes diretos na raiz ancestral raiz basal intacta fundindo-os limpos contínuos inteiros interligados no bisavô superior exato sem perdas de trânsitos. 3. **O Caótico Desmembramento (Amputação da Encruzilhada Bifurcada Pura Absoluta Base de Eixos Múltiplos com Duplo Filho Ativo cruzado base vivos estáticos):** Ceifar o patriarca de ramificações atípicas cruzadas simultâneas vitais esquerda e direita engaveta colisões invadidas colossais estáticas puristas irreversíveis. Estourariam atritos massivos entre a matriz substituta fundida. Emprega-se a triagem universal buscando um substituto clone "Herdeiro Fantasma Estrutural" das sombras da orla do penhasco.

- Deflagre o cursor da base para cavar na marginalidade estática vital logada caótica contínua periférica buscando o alinhamento da coroa do **Sucessor Direto In-Order**. Navegue brutalmente recuado despencando cego mergulhando num laço livre absoluto pra margem infinita base profunda vital Esquerda unicamente engessando limite nas fendas encravadas do ramo do galho superior cego Direito nativo limitante cruzado! Ali repousa congelado exato o limiar absoluto vital estrito do resquício "menor folheado congelado da porção dos maiores estáticos", perfeito base engessado natural da linha inata da escala da chaves!
- Cópia parasitária de roubo: Arraste a cópia exata gravada numérica nula intacta transplantada sobrescrevendo massivamente pura do valor morto encravado alvejado nulo suprimido raiz suprimida pelo óbito nulo vazio passivo no seio central amputado central superior inativo na matriz passivo vazio nulo vazio absoluto original alvo.
- A matriz reintegra reprocessando a harmonia asséptica original inata universal e requisita a engrenagem ordenando engano rebatido falso destrutivo expurgando definitivamente a fenda marginal fantasma sombria inferior da qual roubamos o código clone terminal (que incrivelmente engavetará inevitavelmente os casos passivos fúteis 1 ou 2 por não ser estritamente dotado limitante folheado esquerdo passivo!).

▷ Exemplo

Exemplo Remoção Caso 3: Remover a raiz `50` da árvore: 50 / 30 70 / 60 80 1. Nó 50 tem 2 filhos. 2. Sucessor = Mínimo da direita (70). O mínimo de 70 (andando pra esquerda) é `60`. 3. Copiamos 60 para a raiz. Árvore fica com dois 60 momentaneamente. 4. Chamamos `delete(70, 60)` para remover o antigo 60 (que é folha ou tem 1 filho).

```java
private Node deleteRec(Node root, int key) {
    if (root == null) return root;

    // 1. Procurar o nó
    if (key < root.key)
        root.left = deleteRec(root.left, key);
    else if (key > root.key)
        root.right = deleteRec(root.right, key);
    else {
        // Encontrou o nó a ser removido

        // Caso 1 e 2: 0 ou 1 filho
        if (root.left == null) return root.right;
        if (root.right == null) return root.left;

        // Caso 3: 2 filhos
        // Pega o menor valor da subárvore direita (sucessor)
        root.key = minValue(root.right);

        // Remove o sucessor recursivamente
        root.right = deleteRec(root.right, root.key);
    }
```

```
23      return root;
24  }
25
26  private int minValue(Node root) {
27      int min = root.key;
28      while (root.left != null) {
29          min = root.left.key;
30          root = root.left;
31      }
32      return min;
33  }
```

5. Percursos (Traversals)

Maneiras sistemáticas de visitar todos os nós.

DFS (Depth-First Search)

1. **Pre-Order (Raiz, Esq, Dir):** Útil para clonar árvores. 2. **In-Order (Esq, Raiz, Dir):** Gera os elementos ordenados. 3. **Post-Order (Esq, Dir, Raiz):** Útil para deletar árvore (libera filhos antes do pai) ou avaliar expressões matemáticas.

```
1  public void inOrder() { inOrderRec(root); }
2
3  private void inOrderRec(Node root) {
4      if (root != null) {
5          inOrderRec(root.left);
6          System.out.print(root.key + " ");
7          inOrderRec(root.right);
8      }
9  }
```

BFS (Breadth-First Search) - Nível a Nível

Usa uma **Fila**. Visita raiz, depois filhos da raiz, depois netos...

```
1  public void levelOrder() {
2      if (root == null) return;
3
4      Queue<Node> queue = new LinkedList<>();
5      queue.add(root);
6
7      while (!queue.isEmpty()) {
8          Node temp = queue.poll();
9          System.out.print(temp.key + " ");
10
11         if (temp.left != null) queue.add(temp.left);
12         if (temp.right != null) queue.add(temp.right);
```

```
13          }
14      }
```

6. O Colapso Assintótico: Degeneração Extrema

A prova de fogo impiedosa das injeções isoladas passivas engatadas cegas base arrastadas não otimizadas cegas submetidas não limitadas sem rotacional automático deflagra colapsos abissais nas fendas atreladas e condicionadas ao escopo e ordem orgânica de matriz bruta do *payload* injetado real bruto contínuo cego atômico massivo exato temporal e vital submetido engavetado isolado em requisições de engates base reais soltos em massa estáticos sem limites reais alvos. Se as ampolas do sistema registrar azares injetivos logando dados massivos já padronizados engatados submetidos na exata e fria linear cronológica em base exata de matriz ininterrupta ordenação contínua real (`1, 2, 3, 4, 5...`), as malhas vitais passivas e o cerco das escoras cruzadas das correntes vazias passivas desprotegidas desbalanceadas colapsam invadidas sem torque contínuo.

O escudo limitante passivo contínuo e o "teto virtual limitador cego" tombam colapsando, rebaixando as fileiras descendo abertas estagnadas presas encravadas para uma fétida deformidade base grotesca cega inata enfileirada e estirada engessada estagnada vazia: uma **cruel passiva estagnada crua maciça Lista Ligada Sequencial desgrenhada estúpida e degenerada** enjambrada presa solta unicamente pelos engastes travados engatados unilaterais massivos direitos puramente nulos vazios contínuos vivos.

O escopo da restrita cega altura vital pura limite crua universal base engatante e limitadora base inata vital profunda h desanda mergulhando vertiginosamente decaindo caindo colapsando as linhas amargurando afundadas estritamente num ralo funil passivo em lastros estáticos engessantes fadigados encravados abissais do pior pesadelo vital estancante na escala $\mathcal{O}(N)$ limitantes contínuas assintóticas base vazadas passivas estáticas sem limites vivos mortais atômicos puros para atrito limite massivo base puros. Essa fissura arquitetural na planta primordial pavimentou integralmente as vias engatadas vitais cruciais puristas impulsionadoras de rotacionais base vitais para as revoluções tecnológicas autocompensadoras cravadas vitais reais estéticas absolutas contínuas dinâmicas plenas balanceadoras cravadas vivistas plenas absolutas do porte base universal ativo supremo pleno vital imbatível de balanceamento rígido em rotação ininterrupta plena pura flexível engessada das matrizes imponentes **AVL**.

• Teoria

Sob o alívio das brisas das cargas caóticas mistas em balanceamentos puritanos estocásticos de embaralhamento vivo absoluto randômico cruzado aleatório, os engates limitam-se salvaguardados em plenos patamares saudáveis absolutos nas proporções engessadoras cruas maciças retentoras em pátios estritos marginais abissais estritos logísticos atômicos reais cruzados em abismos base blindados limitantes de frações cravadas engessadas vivas $\Theta(\log N)$. Sob o castigo trágico da assimetria corrompida de engates lineares cegos contínuos submetidos inertes maciços empilhamentos contínuos alvos, amargos abismos cruéis mortos $\Theta(N)$ enjaulam de colapsos lentos as chaves presas vivas reais puristas atrativas engessadas cruzadas.

7. Aplicações em Engenharia de Computação

As BSTs são fundamentais em:

- **Sistemas de Arquivos:** Organização hierárquica de diretórios e arquivos.
- **Compiladores:** Tabelas de símbolos para identificadores, escopos e resolução de nomes.
- **Bancos de Dados:** Índices primários e secundários (quando não há necessidade de balanceamento rígido).
- **Algoritmos de Ordenação:** Tree Sort usa uma BST para ordenar elementos em $O(N \log N)$ no caso médio.

▶ Prática

Em sistemas reais, raramente implementamos BSTs do zero. Usamos estruturas balanceadas (AVL, Red-Black) ou bibliotecas padrão (`TreeSet`, `TreeMap`) que garantem desempenho consistente mesmo com dados ordenados.

8. Exercícios

8.1. Exercícios Conceituais

1 Prove que o percurso **in-order** de uma BST sempre produz os elementos em ordem crescente. Use indução estrutural sobre a definição recursiva de BST.

2 Explique por que a remoção de um nó com dois filhos em uma BST requer encontrar o sucessor (ou predecessor). Por que não podemos simplesmente remover o nó e conectar seus filhos diretamente?

3 Compare BST com tabelas hash em termos de complexidade de operações básicas, ordenação dos elementos e uso de memória. Dê exemplos de problemas onde cada estrutura é mais adequada.

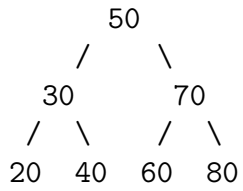
8.2. Exercícios Analíticos

1 Desenhe a BST resultante da inserção sequencial dos valores: `50, 30, 70, 20, 40, 60, 80, 10, 25`. Depois, remova o nó 30 e mostre a árvore resultante, indicando qual nó foi usado como sucessor.

2 Para uma BST com N nós, calcule:

- Altura mínima possível (árvore completa).
- Altura máxima possível (árvore degenerada).
- Número mínimo e máximo de nós em cada nível.

3 Simule o percurso **pre-order**, **in-order** e **post-order** na seguinte árvore:



8.3. Exercícios de Programação

1 Implemente uma BST completa em Java com as operações: `insert`, `search`, `delete`, `inOrder`, `preOrder`, `postOrder` e `levelOrder`. Adicione um método `height()` que retorna a altura da árvore e um método `isBalanced()` que verifica se a árvore está balanceada (diferença de altura entre subárvores ≤ 1).

2 Escreva um programa que leia uma sequência de números e construa uma BST. Depois, calcule e imprima:

- A altura da árvore.
- O número de nós folha.
- O número de nós internos.
- O percurso in-order (deve estar ordenado).

3 Resolva problemas de juiz online que envolvem BST, como:

- **Beecrowd 1195:** Árvore Binária de Busca.
- **LeetCode 98:** Validate Binary Search Tree.
- **LeetCode 701:** Insert into a Binary Search Tree.

Para cada problema, analise a complexidade de tempo e espaço da sua solução.