AED2 - Algoritmos e Estr. de Dados II

Aula 6: Árvores Binárias de Busca (BST)

Prof. Alléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Fevereiro de 2026

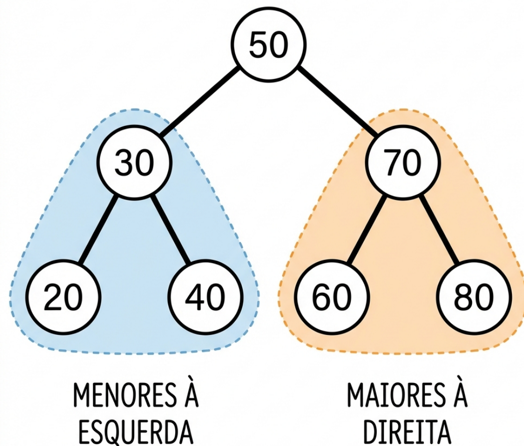
Sumário

1. Introdução às Árvores

Até agora, nossas estruturas eram lineares (um após o outro). Agora, entramos no mundo **hierárquico**. Uma Árvore é uma coleção de nós, onde existe um nó especial chamado **Raiz** e os demais são divididos em subconjuntos disjuntos (subárvores). As BSTs (Binary Search Trees) combinam a flexibilidade de inserção das listas encadeadas com a eficiência de busca dos vetores ordenados.

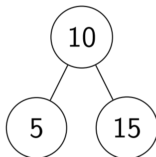
2. Definição Formal de BST

Uma árvore binária é **BST** se: esquerda $<$ raiz $<$ direita (para todo nó). In-Order $\Rightarrow$ ordem crescente.

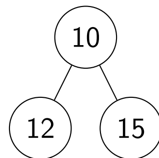


Exemplo Visual

É BST:



NÃO é BST:



12 está na esquerda de 10!

3. Implementação em Java

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

```java
public class BST {

    private class Node {
        int key;
        Node left, right;

        public Node(int item) {
            key = item;
            left = right = null;
        }
    }

    private Node root;

    public BST() {
        root = null;
    }
```

Remover é mais complexo pois não podemos quebrar a estrutura da árvore. Existem 3 casos: 1. **Nó Folha:** Basta setar o ponteiro do pai para `null`. 2. **Nó com 1 Filho:** O pai aponta direto para o filho único (pula o nó removido). 3. **Nó com 2 Filhos:** Não podemos remover simplesmente. Estratégia:

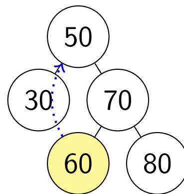
- Encontre o **Sucessor** (o menor valor da subárvore direita).
- Copie o valor do sucessor para o nó atual.
- Remova o sucessor (que cairá no caso 1 ou 2, pois o menor da direita nunca tem filho esquerdo!).

```
1  private Node deleteRec(Node root, int key) {
2      if (root == null) return root;
3  
4      // 1. Procurar o nó
5      if (key < root.key)
6          root.left = deleteRec(root.left, key);
7      else if (key > root.key)
8          root.right = deleteRec(root.right, key);
9      else {
10         // Encontrou o nó a ser removido
```

Exemplo Remoção (Nó com 2 filhos)

Remover **50** (Raiz):

1. Nó 50 tem 2 filhos. 2. Sucessor = Mínimo da dir (**60**). 3. Copia 60 p/ raiz. 4. Remove antigo 60 (nó folha).



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.

```
public void inOrder() { inOrderRec(root); }

private void inOrderRec(Node root) {
    if (root != null) {
        inOrderRec(root.left);
        System.out.print(root.key + " ");
        inOrderRec(root.right);
    }
}
```

6. O Problema do Balanceamento

Se inserirmos os números na ordem `1, 2, 3, 4, 5`, a BST vira uma "escada" para a direita (lista encadeada). A altura h vira N, e a busca $O(N)$. Para garantir eficiência $O(\log N)$, a árvore precisa estar **Balanceada** (altura próxima de $\log_2 N$). Isso nos leva ao próximo tópico: Árvores AVL.

Prof. Aléssio Miranda Júnior - alessio@cefetmg.br

CEFET-MG - Campus Timóteo