Trabalho Prático: Árvores Binárias

Marcos Vinícius de Oliveira - 922143517

17/03/2025

1 Introdução

O presente trabalho tem como objetivo a implementação de uma árvore binária com as funcionalidades básicas e avançadas de manipulação de dados. A estrutura permite inserção, remoção, pesquisa, verificação de tipos (árvore cheia, completa e estritamente binária), travessias (in-order, pre-order e post-order), grau de um nó e nível da árvore. O projeto foi desenvolvido em linguagem Java, utilizando os princípios da programação orientada a objetos.

2 Implementação

A implementação foi dividida em três classes principais: `Node`, `BinaryTree` e `Main`.

2.1 Estrutura de Dados

A classe `Node` representa cada nó da árvore, contendo o valor inteiro e referências para seus filhos esquerdo e direito. A classe `BinaryTree` contém os métodos de manipulação e verificação da árvore, e a classe `Main` serve como interface de linha de comando para testes.

2.2 Funcionalidades

- **Inserção e remoção**: métodos recursivos que mantêm as propriedades da árvore binária de busca.
- **Busca de elemento**: retorna verdadeiro se o elemento estiver presente.
- **Travessias**: in-order, pre-order e post-order.
- **Verificação do tipo da árvore**:
  - Completa: todos os níveis, exceto o último, estão completamente preenchidos.
  - Cheia: todos os nós têm 0 ou 2 filhos.
  - Estrita: todos os nós internos têm exatamente dois filhos.
- **Grau de um nó**: número de filhos de um nó especificado.
- **Nível da árvore**: altura da árvore.

2.3 Entrada e Saída

A entrada dos dados é feita via terminal interativo. O usuário pode escolher operações por um menu, inserir valores e ver os resultados diretamente na tela.

2.4 Decisões de Implementação

Optou-se por não permitir a inserção de valores duplicados. O menu textual facilita testes manuais e simulações.

3 Testes Executados

Foram realizados testes com as seguintes operações:

- Inserção dos valores: 50, 30, 70, 20, 40, 60, 80
- Impressões in-order, pre-order e post-order
- Verificação de árvore cheia, completa e estrita
- Remoção dos nós: 20 (folha), 30 (com um filho), 50 (com dois filhos)
- Cálculo do grau do nó 70 (esperado: 2)
- Cálculo do nível da árvore (esperado: 3)

Os testes foram todos bem-sucedidos.

4 Conclusão

O trabalho permitiu o aprofundamento dos conceitos de árvores binárias e a aplicação prática de estruturas de dados e recursão. A maior dificuldade foi garantir o correto funcionamento dos métodos de remoção e verificação de tipos da árvore.

5 Repositório do Projeto

O código-fonte completo do projeto, juntamente com uma versão resumida desta documentação, está disponível no seguinte repositório público:

- `https://github.com/M4rcosVO/ArvoreBinaria`

6 Bibliografia

- Jovana, M. (2019). *Estruturas de Dados em Java.*
- Geeks for Geeks. `https://www.geeksforgeeks.org`
- Overleaf Team. (2025). *Documentação em LaTeX.* `https://www.overleaf.com`

Anexo: Código-fonte

Node.java

```
public class Node {
    int data;
    Node left, right;

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

BinaryTree.java

```
public class BinaryTree {
    private Node root;

    // Inser  o
    public void insert(int value) {
        root = insertRec(root, value);
    }

    private Node insertRec(Node root, int value) {
        if (root == null) {
            root = new Node(value);
            return root;
        }
        if (value < root.data)
            root.left = insertRec(root.left, value);
        else if (value > root.data)
            root.right = insertRec(root.right, value);
        return root;
    }

    // Remo  o
    public void remove(int value) {
        root = removeRec(root, value);
    }

    private Node removeRec(Node root, int value) {
        if (root == null) return root;
        if (value < root.data) {
            root.left = removeRec(root.left, value);
        } else if (value > root.data) {
            root.right = removeRec(root.right, value);
        } else {
            // N  com um ou nenhum filho
            if (root.left == null)
                return root.right;
            else if (root.right == null)
                return root.left;

            // N  com dois filhos
            root.data = minValue(root.right);
```

```
            root.right = removeRec(root.right, root.data);
        }
        return root;
    }

    private int minValue(Node root) {
        int minv = root.data;
        while (root.left != null) {
            minv = root.left.data;
            root = root.left;
        }
        return minv;
    }

    // Pesquisa
    public boolean search(int value) {
        return searchRec(root, value);
    }

    private boolean searchRec(Node root, int value) {
        if (root == null) return false;
        if (root.data == value) return true;
        return value < root.data ? searchRec(root.left, value) :
           searchRec(root.right, value);
    }

    // Traversals
    public void inOrder() {
        inOrderRec(root);
        System.out.println();
    }

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

    public void preOrder() {
        preOrderRec(root);
        System.out.println();
    }

    private void preOrderRec(Node root) {
        if (root != null) {
            System.out.print(root.data + "␣");
            preOrderRec(root.left);
            preOrderRec(root.right);
        }
    }

    public void postOrder() {
        postOrderRec(root);
        System.out.println();
    }
```

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

    // Verifica se    cheia
    public boolean isFull() {
        return isFullRec(root);
    }

    private boolean isFullRec(Node node) {
        if (node == null) return true;
        if (node.left == null && node.right == null) return true;
        if (node.left != null && node.right != null)
            return isFullRec(node.left) && isFullRec(node.right);
        return false;
    }

    // Verifica se    estritamente bin ria
    public boolean isStrict() {
        return isStrictRec(root);
    }

    private boolean isStrictRec(Node node) {
        if (node == null) return true;
        if ((node.left == null && node.right != null) || (node.left !=
           null && node.right == null))
            return false;
        return isStrictRec(node.left) && isStrictRec(node.right);
    }

    // Verifica se    completa
    public boolean isComplete() {
        int nodeCount = countNodes(root);
        return isCompleteRec(root, 0, nodeCount);
    }

    private boolean isCompleteRec(Node node, int index, int count) {
        if (node == null) return true;
        if (index >= count) return false;
        return isCompleteRec(node.left, 2 * index + 1, count) &&
               isCompleteRec(node.right, 2 * index + 2, count);
    }

    private int countNodes(Node node) {
        if (node == null) return 0;
        return 1 + countNodes(node.left) + countNodes(node.right);
    }

    // Grau de um n
    public int getGrau(int value) {
        Node node = findNode(root, value);
        if (node == null) return -1;
        int grau = 0;
        if (node.left != null) grau++;
```

```
        if (node.right != null) grau++;
        return grau;
    }

    private Node findNode(Node root, int value) {
        if (root == null || root.data == value) return root;
        return value < root.data ? findNode(root.left, value) : findNode
            (root.right, value);
    }

    // N vel da  rvore
    public int nivel() {
        return altura(root);
    }

    private int altura(Node node) {
        if (node == null) return 0;
        int left = altura(node.left);
        int right = altura(node.right);
        return Math.max(left, right) + 1;
    }
}
```

Main.java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        BinaryTree bt = new BinaryTree();
        Scanner sc = new Scanner(System.in);
        int opcao, valor;

        do {
            System.out.println("\n1. Inserir\n2. Remover\n3. Buscar\n4. 
                InOrder\n5. PreOrder\n6. PostOrder\n7. Verificar tipo da 
                 rvore \n8. Grau de um n  \n9. N vel da  rvore \n0. Sair
                ");
            System.out.print("Escolha: ");
            opcao = sc.nextInt();

            switch (opcao) {
                case 1:
                    System.out.print("Valor: ");
                    valor = sc.nextInt();
                    bt.insert(valor);
                    break;
                case 2:
                    System.out.print("Valor: ");
                    valor = sc.nextInt();
                    bt.remove(valor);
                    break;
                case 3:
                    System.out.print("Valor: ");
                    valor = sc.nextInt();
                    System.out.println(bt.search(valor) ? "Encontrado!"
                        : "N o encontrado.");
```

```
                    break;
                case 4:
                    bt.inOrder();
                    break;
                case 5:
                    bt.preOrder();
                    break;
                case 6:
                    bt.postOrder();
                    break;
                case 7:
                    System.out.println("Completa: " + bt.isComplete());
                    System.out.println("Cheia: " + bt.isFull());
                    System.out.println("Estrita: " + bt.isStrict());
                    break;
                case 8:
                    System.out.print("Valor: ");
                    valor = sc.nextInt();
                    int grau = bt.getGrau(valor);
                    if (grau == -1)
                        System.out.println("N  n  o encontrado.");
                    else
                        System.out.println("Grau: " + grau);
                    break;
                case 9:
                    System.out.println("N vel da  rvore : " + bt.nivel
                        ());
                    break;
            }
        } while (opcao != 0);
        sc.close();
    }
}
```