Implementação de Árvore Binária em Java

João Victor de Souza Gonçalves – 92320247
Lucas Gabriel Rodrigues Valadares – 92310851
Marcus Vinícius Fernandes Lima – 92311773
Natan Rodrigo Faria Vaz – 92310556

Disciplina: Estruturas de Dados e Análise de Algoritmos

1 Introdução

Este trabalho tem como objetivo a implementação de uma Árvore Binária de Busca (BST), que é uma estrutura de dados fundamental em algoritmos eficientes de ordenação, busca e manipulação hierárquica de dados (Cormen et al., 2009). O projeto foi desenvolvido na linguagem Java, adotando práticas de programação orientada a objetos conforme recomendado por Goodrich et al. (2014).

A aplicação permite realizar operações como inserção, remoção, busca e travessias da árvore, além de avaliar propriedades estruturais como ser completa, cheia, estritamente binária ou perfeita. Todo o código desenvolvido pode ser consultado no seguinte repositório: `https://github.com/joaokoa/binary-tree-project`.

2 Implementação

2.1 Estrutura de Dados

A estrutura implementada segue os princípios de uma Árvore Binária de Busca, onde cada nó possui no máximo dois filhos, e a inserção é feita de forma ordenada: valores menores à esquerda e maiores à direita (Cormen et al., 2009).

Foram utilizadas três classes principais:

- **BinaryTreeNode**: representa um nó da árvore, contendo o valor, ponteiros para os filhos esquerdo e direito e um método que retorna seu grau.
- **BinaryTree**: contém a lógica principal da árvore binária, como inserção, remoção, travessia e verificações estruturais.
- **Main**: realiza testes com inserção de nós, exibição das travessias e chamadas às funções de verificação.

2.2 Decisões de Projeto

A escolha por Java foi feita por ser uma linguagem amplamente utilizada para ensino de estruturas de dados e possuir forte suporte a generics, o que permite reutilização do código com diferentes tipos de dados (Weiss, 2013).

Todos os métodos foram implementados de forma recursiva, com ênfase na clareza e legibilidade do código. O método de remoção, por exemplo, trata separadamente os casos de remoção de folhas, nós com um filho ou dois filhos, como proposto por Goodrich et al. (2014).

2.3 Formato de Entrada e Saída

A entrada é feita via código-fonte no método `main`, com inserção de valores inteiros. A saída é impressa no console em forma textual, exibindo os percursos da árvore e propriedades avaliadas.

3 Testes Executados

Foram realizados testes unitários com as seguintes operações:

- Inserção dos elementos: 50, 30, 70, 20, 40, 60, 80;
- Impressão das travessias: In-Order, Pre-Order e Post-Order;
- Avaliação das propriedades: altura da árvore, se é cheia, completa, estritamente binária e perfeita;
- Remoção do elemento 20 (folha) e nova travessia In-Order.

Os resultados foram consistentes com o comportamento esperado para árvores binárias de busca, conforme descrito por Cormen et al. (2009).

4 Conclusão

A implementação da árvore binária proporcionou uma melhor compreensão de estruturas hierárquicas e estratégias recursivas para manipulação de dados. A principal dificuldade foi garantir o correto funcionamento das verificações estruturais da árvore (cheia, completa, perfeita), o que exigiu revisões conceituais detalhadas (Goodrich et al., 2014).

O uso da orientação a objetos e dos generics em Java facilitou a organização e reutilização do código. Além disso, o projeto também serviu como base para o uso de controle de versão via GitHub, onde o repositório completo está disponível: `https://github.com/joaokoa/binary-tree-project`.

5 Bibliografia

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Java* (6th ed.). Wiley.
- Weiss, M. A. (2013). *Data Structures and Problem Solving Using Java* (4th ed.). Pearson.
- GitHub Project: `https://github.com/joaokoa/binary-tree-project`

Anexo: Código-Fonte

BinaryTreeNode.java

```
public class BinaryTreeNode<T extends Comparable<T>> {
    T data;
    BinaryTreeNode<T> left;
    BinaryTreeNode<T> right;

    public BinaryTreeNode(T data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }

    /**
     * Retorna o grau do n   (n mero de filhos)
     * @return 0 (folha), 1 (um filho) ou 2 (dois filhos)
     */
    public int getDegree() {
        int degree = 0;
        if (left != null) degree++;
        if (right != null) degree++;
        return degree;
    }

    @Override
    public String toString() {
        return data.toString();
    }
}
```

BinaryTree.java

```
public class BinaryTree<T extends Comparable<T>> {
    private BinaryTreeNode<T> root;
    private int size;

    public BinaryTree() {
        this.root = null;
        this.size = 0;
    }

    // M todo de inser   o
    public void insert(T data) {
        root = insertRec(root, data);
        size++;
    }

    private BinaryTreeNode<T> insertRec(BinaryTreeNode<T> node, T data)
       {
        if (node == null) {
            return new BinaryTreeNode<>(data);
        }

        if (data.compareTo(node.data) < 0) {
```

```
            node.left = insertRec(node.left, data);
        } else if (data.compareTo(node.data) > 0) {
            node.right = insertRec(node.right, data);
        }

        return node;
    }

    // M todo de remo   o
    public boolean remove(T data) {
        int initialSize = size;
        root = removeRec(root, data);
        return size != initialSize;
    }

    private BinaryTreeNode<T> removeRec(BinaryTreeNode<T> node, T data)
       {
        if (node == null) {
            return null;
        }

        if (data.compareTo(node.data) < 0) {
            node.left = removeRec(node.left, data);
        } else if (data.compareTo(node.data) > 0) {
            node.right = removeRec(node.right, data);
        } else {
            if (node.left == null) {
                size--;
                return node.right;
            } else if (node.right == null) {
                size--;
                return node.left;
            }

            node.data = minValue(node.right);
            node.right = removeRec(node.right, node.data);
        }

        return node;
    }

    private T minValue(BinaryTreeNode<T> node) {
        T minValue = node.data;
        while (node.left != null) {
            minValue = node.left.data;
            node = node.left;
        }
        return minValue;
    }

    // M todo de busca
    public boolean contains(T data) {
        return containsRec(root, data);
    }

    private boolean containsRec(BinaryTreeNode<T> node, T data) {
        if (node == null) {
            return false;
```

```
        }
        if (data.compareTo(node.data) == 0) {
            return true;
        }
        return data.compareTo(node.data) < 0
            ? containsRec(node.left, data)
            : containsRec(node.right, data);
    }

    // M todos de travessia
    public void printInOrder() {
        inOrderRec(root);
        System.out.println();
    }

    private void inOrderRec(BinaryTreeNode<T> node) {
        if (node != null) {
            inOrderRec(node.left);
            System.out.print(node.data + " ");
            inOrderRec(node.right);
        }
    }

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

    private void preOrderRec(BinaryTreeNode<T> node) {
        if (node != null) {
            System.out.print(node.data + " ");
            preOrderRec(node.left);
            preOrderRec(node.right);
        }
    }

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

    private void postOrderRec(BinaryTreeNode<T> node) {
        if (node != null) {
            postOrderRec(node.left);
            postOrderRec(node.right);
            System.out.print(node.data + " ");
        }
    }

    // M todos de informa  o
    public int getHeight() {
        return calculateHeight(root);
    }

    private int calculateHeight(BinaryTreeNode<T> node) {
        if (node == null) {
            return 0;
        }
```

```
        return 1 + Math.max(calculateHeight(node.left), calculateHeight(
            node.right));
    }

    public int getNodeDegree(T data) {
        BinaryTreeNode<T> node = findNode(root, data);
        return node != null ? node.getDegree() : -1;
    }

    private BinaryTreeNode<T> findNode(BinaryTreeNode<T> node, T data) {
        if (node == null || data.compareTo(node.data) == 0) {
            return node;
        }
        return data.compareTo(node.data) < 0
            ? findNode(node.left, data)
            : findNode(node.right, data);
    }

    // M todos de verifica  o
    public boolean isStrictlyBinary() {
        return isStrictlyBinaryRec(root);
    }

    private boolean isStrictlyBinaryRec(BinaryTreeNode<T> node) {
        if (node == null) {
            return true;
        }
        if ((node.left == null && node.right != null) ||
            (node.left != null && node.right == null)) {
            return false;
        }
        return isStrictlyBinaryRec(node.left) && isStrictlyBinaryRec(
            node.right);
    }

    public boolean isComplete() {
        int height = getHeight();
        return isCompleteRec(root, 0, height);
    }

    private boolean isCompleteRec(BinaryTreeNode<T> node, int level, int
        height) {
        if (node == null) {
            return level >= height - 1;
        }
        if (level == height - 1) {
            return node.left == null && node.right == null;
        }
        return isCompleteRec(node.left, level + 1, height) &&
               isCompleteRec(node.right, level + 1, height);
    }

    public boolean isFull() {
        int height = getHeight();
        return isFullRec(root, 0, height);
    }
```

```
    private boolean isFullRec(BinaryTreeNode<T> node, int level, int
        height) {
        if (node == null) {
            return level >= height;
        }
        if (level == height - 1) {
            return true;
        }
        if (node.left == null || node.right == null) {
            return false;
        }
        return isFullRec(node.left, level + 1, height) &&
               isFullRec(node.right, level + 1, height);
    }

    public boolean isPerfect() {
        int height = getHeight();
        return size == (Math.pow(2, height) - 1);
    }

    public int size() {
        return size;
    }
}
```

Main.java

```
public class BinaryTree<T extends Comparable<T>> {
    private BinaryTreeNode<T> root;
    private int size;

    public BinaryTree() {
        this.root = null;
        this.size = 0;
    }

    // M todo de inser  o
    public void insert(T data) {
        root = insertRec(root, data);
        size++;
    }

    private BinaryTreeNode<T> insertRec(BinaryTreeNode<T> node, T data)
        {
        if (node == null) {
            return new BinaryTreeNode<>(data);
        }

        if (data.compareTo(node.data) < 0) {
            node.left = insertRec(node.left, data);
        } else if (data.compareTo(node.data) > 0) {
            node.right = insertRec(node.right, data);
        }

        return node;
    }
```

```
    // M todo de remo   o
    public boolean remove(T data) {
        int initialSize = size;
        root = removeRec(root, data);
        return size != initialSize;
    }

    private BinaryTreeNode<T> removeRec(BinaryTreeNode<T> node, T data)
       {
        if (node == null) {
            return null;
        }

        if (data.compareTo(node.data) < 0) {
            node.left = removeRec(node.left, data);
        } else if (data.compareTo(node.data) > 0) {
            node.right = removeRec(node.right, data);
        } else {
            if (node.left == null) {
                size--;
                return node.right;
            } else if (node.right == null) {
                size--;
                return node.left;
            }

            node.data = minValue(node.right);
            node.right = removeRec(node.right, node.data);
        }

        return node;
    }

    private T minValue(BinaryTreeNode<T> node) {
        T minValue = node.data;
        while (node.left != null) {
            minValue = node.left.data;
            node = node.left;
        }
        return minValue;
    }

    // M todo de busca
    public boolean contains(T data) {
        return containsRec(root, data);
    }

    private boolean containsRec(BinaryTreeNode<T> node, T data) {
        if (node == null) {
            return false;
        }
        if (data.compareTo(node.data) == 0) {
            return true;
        }
        return data.compareTo(node.data) < 0
            ? containsRec(node.left, data)
            : containsRec(node.right, data);
```

```
    }

    // M todos de travessia
    public void printInOrder() {
        inOrderRec(root);
        System.out.println();
    }

    private void inOrderRec(BinaryTreeNode<T> node) {
        if (node != null) {
            inOrderRec(node.left);
            System.out.print(node.data + " ");
            inOrderRec(node.right);
        }
    }

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

    private void preOrderRec(BinaryTreeNode<T> node) {
        if (node != null) {
            System.out.print(node.data + " ");
            preOrderRec(node.left);
            preOrderRec(node.right);
        }
    }

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

    private void postOrderRec(BinaryTreeNode<T> node) {
        if (node != null) {
            postOrderRec(node.left);
            postOrderRec(node.right);
            System.out.print(node.data + " ");
        }
    }

    // M todos de informa  o
    public int getHeight() {
        return calculateHeight(root);
    }

    private int calculateHeight(BinaryTreeNode<T> node) {
        if (node == null) {
            return 0;
        }
        return 1 + Math.max(calculateHeight(node.left), calculateHeight(
           node.right));
    }

    public int getNodeDegree(T data) {
        BinaryTreeNode<T> node = findNode(root, data);
        return node != null ? node.getDegree() : -1;
```

```
    }

    private BinaryTreeNode<T> findNode(BinaryTreeNode<T> node, T data) {
        if (node == null || data.compareTo(node.data) == 0) {
            return node;
        }
        return data.compareTo(node.data) < 0
            ? findNode(node.left, data)
            : findNode(node.right, data);
    }

    // M todos de verifica  o
    public boolean isStrictlyBinary() {
        return isStrictlyBinaryRec(root);
    }

    private boolean isStrictlyBinaryRec(BinaryTreeNode<T> node) {
        if (node == null) {
            return true;
        }
        if ((node.left == null && node.right != null) ||
            (node.left != null && node.right == null)) {
            return false;
        }
        return isStrictlyBinaryRec(node.left) && isStrictlyBinaryRec(
           node.right);
    }

    public boolean isComplete() {
        int height = getHeight();
        return isCompleteRec(root, 0, height);
    }

    private boolean isCompleteRec(BinaryTreeNode<T> node, int level, int
        height) {
        if (node == null) {
            return level >= height - 1;
        }
        if (level == height - 1) {
            return node.left == null && node.right == null;
        }
        return isCompleteRec(node.left, level + 1, height) &&
               isCompleteRec(node.right, level + 1, height);
    }

    public boolean isFull() {
        int height = getHeight();
        return isFullRec(root, 0, height);
    }

    private boolean isFullRec(BinaryTreeNode<T> node, int level, int
       height) {
        if (node == null) {
            return level >= height;
        }
        if (level == height - 1) {
            return true;
        }
```

```
        if (node.left == null || node.right == null) {
            return false;public class Main {
    public static void main(String[] args) {
        BinaryTree<Integer> tree = new BinaryTree<>();

        // Testes de inser  o
        tree.insert(50);
        tree.insert(30);
        tree.insert(70);
        tree.insert(20);
        tree.insert(40);
        tree.insert(60);
        tree.insert(80);

        // Testes de travessia
        System.out.println("In-Order:");
        tree.printInOrder();

        System.out.println("Pre-Order:");
        tree.printPreOrder();

        System.out.println("Post-Order:");
        tree.printPostOrder();

        // Testes de propriedades
        System.out.println("Altura: " + tree.getHeight());
        System.out.println("   estritamente bin ria? " + tree.
           isStrictlyBinary());
        System.out.println("   completa? " + tree.isComplete());
        System.out.println("   cheia? " + tree.isFull());
        System.out.println("   perfeita? " + tree.isPerfect());

        // Testes de remo  o
        tree.remove(20);
        System.out.println("Ap s remover 20 (In-Order):");
        tree.printInOrder();
    }
}
        }
        return isFullRec(node.left, level + 1, height) &&
               isFullRec(node.right, level + 1, height);
    }

    public boolean isPerfect() {
        int height = getHeight();
        return size == (Math.pow(2, height) - 1);
    }

    public int size() {
        return size;
    }
}
```