



**UNIVERSIDADE FEDERAL DO CEARÁ**  
**CAMPUS DE QUIXADÁ**  
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**ED2 - TAREFA DE IMPLEMENTACAO - 2025.1: VISUALIZAÇÃO DE ÁRVORES**  
**AVL**

**QUIXADÁ**  
**2025**

## ED2 - TAREFA DE IMPLEMENTACAO - 2025.1: VISUALIZAÇÃO DE ÁRVORES AVL

Trabalho final da Disciplina de Estrutura de  
Dados Avançada.

Professor: Prof. Me. Arthur Rodrigues Araruna

QUIXADÁ

2025

## LISTA DE FIGURAS

Figura 1- Estrutura do projeto. ....	15
Figura 2 - Código fonte do arquivo Main.java .....	16
Figura 3 - Código fonte do arquivo Node.java.....	17
Figura 4 - Código fonte do arquivo AvlTree.java.....	18
Figura 5 - Código fonte do arquivo AVLMetodos.java .....	19
Figura 6 - Adicionando elementos.....	21
Figura 7 - Todos os elementos adicionados.....	22
Figura 8 - Tentativa de inserir elemento repetido.....	22
Figura 9 - AVL Tree visualization.....	23
Figura 10 - Busca por uma chave existente.....	24
Figura 11- Busca por uma chave não existente. ....	24
Figura 12 - Excluindo elemento da árvore. ....	25
Figura 13- Função que exibe a árvore no terminal.....	26

## **LISTA DE TABELAS**

Tabela 1 - Listagem das opções do menu em Main.java .....	16
Tabela 2 - Listagem com as principais funções e sua descrição.....	19
Tabela 3 - Continuação da listagem com as principais funções e sua descrição. ....	20

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>14</b>
<b>2 FUNCIONALIDADES IMPLEMENTADAS.....</b>	<b>15</b>
<b>2.1 Estrutura do projeto.....</b>	<b>15</b>
<i>2.1.1 Main.java – Interface interativa.....</i>	<i>16</i>
<i>2.1.2 Node.java – Definição dos nós da árvore .....</i>	<i>17</i>
<i>2.1.3 AvlTree.java — Implementação da Estrutura AVL .....</i>	<i>18</i>
<i>2.1.4 AVLMetodos.java – Classe abstrata.....</i>	<i>19</i>
<b>2.2 Principais funções e sua descrição .....</b>	<b>19</b>
<b>3 FUNCIONAMENTO DA APLICAÇÃO .....</b>	<b>21</b>
<b>3.1 Sequência de inserções .....</b>	<b>21</b>
<b>3.2 Função adicionar elemento .....</b>	<b>21</b>
<i>3.2.1 Ferramenta de validação.....</i>	<i>23</i>
<b>3.3 Função buscar elemento.....</b>	<b>24</b>
<b>3.4 Função excluir elemento .....</b>	<b>25</b>
<b>3.5 Função mostrar árvore.....</b>	<b>26</b>
<b>4 RELATO DE DESENVOLVIMENTO .....</b>	<b>27</b>
<b>5 PONTOS DE DIFICULDADE ENCONTRADOS.....</b>	<b>27</b>

## 1 INTRODUÇÃO

Este documento apresenta o relatório de implementação referente à segunda tarefa da disciplina de Estrutura de Dados Avançada (QXD0115), do curso de Ciência da Computação da Universidade Federal do Ceará, campus Quixadá. O objetivo principal do trabalho foi projetar e desenvolver em Java um sistema interativo para a visualização de uma Árvore AVL, uma das mais eficientes estruturas de árvore de busca auto-balanceável.

A solução implementada é baseada em uma estrutura de dados do tipo Árvore AVL, que é uma árvore binária de busca balanceada. O principal desafio e característica desta estrutura é a sua capacidade de se reorganizar após modificações. A cada inserção ou remoção de um nó, a árvore é ajustada automaticamente por meio de rotações, sendo elas simples ou duplas, para garantir que o fator de balanceamento de cada nó permaneça entre -1 e 1. Isso assegura que a complexidade de tempo para as operações de busca, inserção e remoção seja mantida em  $O(\log n)$ .

O sistema oferece ao usuário as funcionalidades de inserir, remover e buscar valores, fornecendo um feedback claro sobre o sucesso de cada operação e exibindo o estado atualizado da árvore de forma visual.

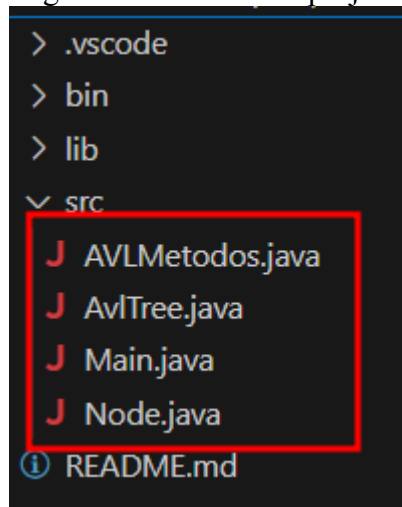
## 2 FUNCIONALIDADES IMPLEMENTADAS

Nesta seção, apresentamos as principais funções implementadas na estrutura da árvore AVL. Cada função tem um papel fundamental no funcionamento correto da árvore, garantindo que ela se mantenha balanceada após inserções ou remoções. Abaixo, estão descritas as funções essenciais, com um breve resumo de suas responsabilidades e comportamentos.

### 2.1 Estrutura do projeto

O projeto foi organizado de forma modular, separando responsabilidades em diferentes arquivos Java. A seguir, são descritas as funções de cada classe localizada na pasta *src*.

Figura 1- Estrutura do projeto.



Fonte: elaborado pelo o autor.

### 2.1.1 Main.java – Interface interativa

Este arquivo contém o ponto de entrada do programa (public static void main). Ele é responsável por apresentar um menu interativo ao usuário, com as principais funcionalidades implementadas na árvore AVL:

Tabela 1 - Listagem das opções do menu em Main.java

Opções presentes no menu	
Adicionar elemento	
Buscar elemento	
Excluir elemento	
Mostrar árvore	
Sair	
<b>Total</b>	<b>5</b>

Fonte: elaborado pelo o autor.

Segue abaixo um trecho do código de Main.java

Figura 2 - Código fonte do arquivo Main.java

```
import java.util.Scanner;

public class Main {

    public static void menu() {
        System.out.println("\nMENU");
        System.out.println("1 - Adicionar elemento");
        System.out.println("2 - Buscar elemento");
        System.out.println("3 - Excluir elemento");
        System.out.println("4 - Mostrar árvore");
        System.out.println("5 - Sair");
        System.out.print("Escolha uma opção: ");
    }

    public static void main(String[] args) {
        AVLTree arvore = new AVLTree();
        Scanner scanner = new Scanner(System.in);
        int opcao, chave;

        do {
            menu();
            opcao = scanner.nextInt();

            switch (opcao) {
                case 1:
                    System.out.print("Digite a chave a ser inserida: ");
                    chave = scanner.nextInt();

                    if (arvore.contains(chave)) {
                        System.out.println("Chave já existente na árvore! Não será inserida novamente.");
                    } else {
                        System.out.println("Inserindo " + chave);
                    }
                // ... (other cases)
            }
        } while (opcao != 5);
    }
}
```

Fonte: elaborado pelo o autor.



### 2.1.2 Node.java – Definição dos nós da árvore

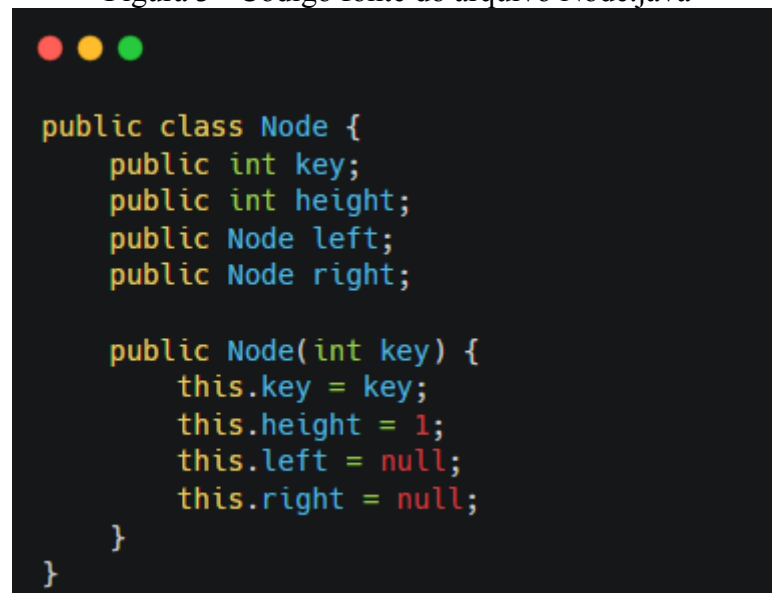
Este arquivo define a estrutura de um nó da árvore AVL. Cada nó possui:

- Uma chave (key) do tipo inteiro;
- Um campo de altura (height);
- Ponteiros para os nós filho esquerdo e direito.

Ele também inclui um construtor para facilitar a criação e inicialização dos nós.

Abaixo o código de Node.java:

Figura 3 - Código fonte do arquivo Node.java

A screenshot of a code editor with a dark background and syntax-highlighted Java code. The code defines a public class Node with attributes key, height, left, and right, and a constructor Node(int key) that initializes these attributes.

```
public class Node {  
    public int key;  
    public int height;  
    public Node left;  
    public Node right;  
  
    public Node(int key) {  
        this.key = key;  
        this.height = 1;  
        this.left = null;  
        this.right = null;  
    }  
}
```

Fonte: elaborado pelo o autor.

### 2.1.3 *AvlTree.java* — Implementação da Estrutura AVL

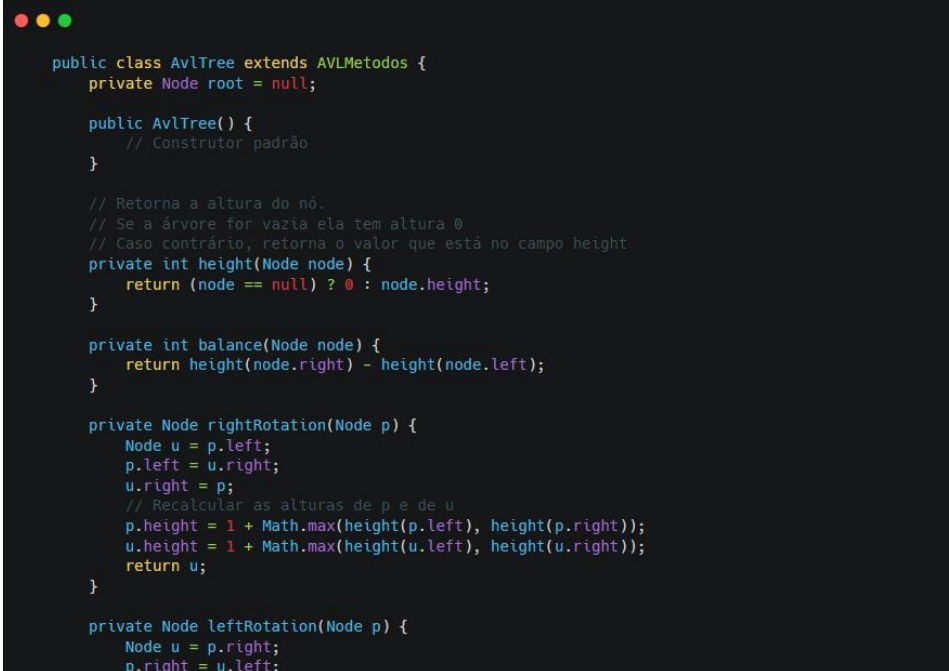
Contém os métodos principais da árvore AVL, como:

- Inserção de elementos com balanceamento automático;
- Remoção de nós;
- Cálculo e atualização de alturas;
- Rotações simples e duplas;
- Verificação do balanceamento.

É o núcleo da estrutura de dados utilizada no projeto.

Abaixo um trecho código de *AvlTree.java*

Figura 4 - Código fonte do arquivo *AvlTree.java*



```

public class AvlTree extends AVLMetodos {
    private Node root = null;

    public AvlTree() {
        // Construtor padrão
    }

    // Retorna a altura do nó.
    // Se a árvore for vazia ela tem altura 0
    // Caso contrário, retorna o valor que está no campo height
    private int height(Node node) {
        return (node == null) ? 0 : node.height;
    }

    private int balance(Node node) {
        return height(node.right) - height(node.left);
    }

    private Node rightRotation(Node p) {
        Node u = p.left;
        p.left = u.right;
        u.right = p;
        // Recalcular as alturas de p e de u
        p.height = 1 + Math.max(height(p.left), height(p.right));
        u.height = 1 + Math.max(height(u.left), height(u.right));
        return u;
    }

    private Node leftRotation(Node p) {
        Node u = p.right;
        p.right = u.left;
    }

```

Fonte: elaborado pelo o autor.

### 2.1.4 AVLMetodos.java – Classe abstrata

O arquivo AVLMetodos.java define uma classe abstrata que estabelece a estrutura básica para qualquer implementação de uma árvore AVL. Essa classe funciona como um contrato: todas as classes que a estendem devem obrigatoriamente implementar os métodos definidos nela.

Abaixo o código de AVLMetodos.java

Figura 5 - Código fonte do arquivo AVLMetodos.java

```
public abstract class AVLMetodos {
    public abstract void add(int key);
    public abstract void clear();
    public abstract boolean remove(int key);
    public abstract boolean empty();
    public abstract boolean contains(int key);
}
```

Fonte: elaborado pelo o autor.

## 2.2 Principais funções e sua descrição

Tabela 2 - Listagem com as principais funções e sua descrição.

Nome da função:	Descrição:
add(int key)	Insere uma nova chave na árvore, apenas se ela ainda não existir. Após a inserção, garante que a árvore continue balanceada usando rotações apropriadas (LL, LR, RR, RL).
remove(int key)	Remove uma chave da árvore se ela existir. Após a remoção, executa rotações se necessário para manter a árvore balanceada.
height()	Retorna a altura da árvore (quantidade de níveis do nó raiz até uma folha).
height_rec(Node node)	Percorre a árvore recursivamente e ajusta o valor do campo height de todos os nós. Útil quando a altura de alguns nós está desatualizada.

<code>balance(Node node)</code>	Calcula o fator de balanceamento de um nó, que é a diferença entre as alturas da subárvore direita e esquerda. Usado para detectar desequilíbrios.
<code>rightRotation(Node p) / leftRotation(Node p)</code>	Executam rotações simples à direita ou à esquerda, utilizadas para rebalancear a árvore quando necessário.
<code>fixup_node(Node p, int key)</code>	Corrige o nó após uma inserção, verificando o balanceamento e aplicando a rotação adequada se necessário.

Fonte: elaborada pelo autor.

Tabela 3 - Continuação da listagem com as principais funções e sua descrição.

<b>Nome da função:</b>	<b>Descrição:</b>
<code>fixup_deletion(Node node)</code>	Corrige a árvore após uma remoção, aplicando rotações se a altura das subárvores ficar desbalanceada.
<code>bshow()</code>	Imprime a estrutura da árvore de forma visual, mostrando as chaves e os fatores de balanceamento de cada nó.
<code>clear()</code>	Remove todos os nós da árvore, esvaziando-a completamente.
<code>contains(int key)</code>	Verifica se uma chave está presente na árvore.
<code>empty()</code>	Retorna true se a árvore estiver vazia, false caso contrário.
<code>search_path(int key)</code>	Imprime o caminho percorrido para encontrar uma chave na árvore. Se não encontrar, informa que a chave não existe.
<code>fixup_node(Node p, int key)</code>	Corrige o nó após uma inserção, verificando o balanceamento e aplicando a rotação adequada se necessário.
<b>Total</b>	<b>8</b>

Fonte: elaborada pelo autor.

### 3 FUNCIONAMENTO DA APLICAÇÃO

#### 3.1 Sequência de inserções

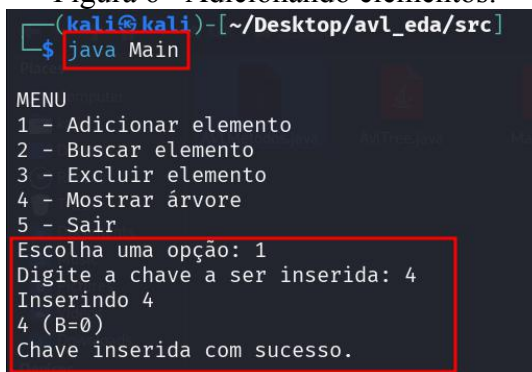
Para demonstrar o funcionamento da árvore AVL, utilizamos os seguintes valores, inseridos na ordem:

<4, 5, 20, 30, 50, 8, 3>

#### 3.2 Função adicionar elemento

O usuário seleciona a opção *1 - Adicionar elemento*, sendo então solicitado a digitar a chave a ser inserida. No exemplo, a *chave escolhida foi 4*. O programa confirma a inserção, informando o valor e o seu fator de balanceamento ( $B=0$ ), indicando que, até o momento, a árvore continua balanceada. Por fim, é exibida a mensagem “*Chave inserida com sucesso.*”, confirmando que a operação foi realizada corretamente.

Figura 6 - Adicionando elementos.



```
(kali@kali)-[~/Desktop/avl_edu/src]
$ java Main
MENU
1 - Adicionar elemento
2 - Buscar elemento
3 - Excluir elemento
4 - Mostrar árvore
5 - Sair
Escolha uma opção: 1
Digite a chave a ser inserida: 4
Inserindo 4
4 (B=0)
Chave inserida com sucesso.
```

Fonte: elaborado pelo o autor.

Para evitar que o relatório se tornasse excessivamente extenso, não foram incluídas capturas de tela de todas as inserções realizadas. No entanto, ao adicionar o último valor, é possível visualizar a árvore AVL completa. É importante destacar que, a cada nova inserção, o programa calcula automaticamente o fator de balanceamento do nó e exibe a estrutura atualizada da árvore, garantindo a manutenção das propriedades da árvore AVL.

Figura 7 - Todos os elementos adicionados.

```

MENU
1 - Adicionar elemento
2 - Buscar elemento
3 - Excluir elemento
4 - Mostrar árvore
5 - Sair
Escolha uma opção: 1
Digite a chave a ser inserida: 3
Inserindo 3
      50 (B=0)
     /
    30 (B=1)
   /  \
  #    #
 /
20 (B=-1)
 /
5  (B=-1)
 /  \
#    4 (B=-1)
     /
    3 (B=0)
Chave inserida com sucesso.

```

Fonte: elaborado pelo o autor.

A Figura 8 demonstra a execução de um caso de teste para a inserção de um elemento duplicado no sistema. O usuário seleciona a opção "1 - Adicionar elemento" e insere a chave "50". Como este valor já existe na árvore, o programa impede a inserção e exibe a mensagem de aviso: "Chave já existente na árvore! Não será inserida novamente."

Figura 8 - Tentativa de inserir elemento repetido.

```

MENU
1 - Adicionar elemento
2 - Buscar elemento
3 - Excluir elemento
4 - Mostrar árvore
5 - Sair
Escolha uma opção: 1
Digite a chave a ser inserida: 50
Chave já existente na árvore! Não será inserida novamente.

```

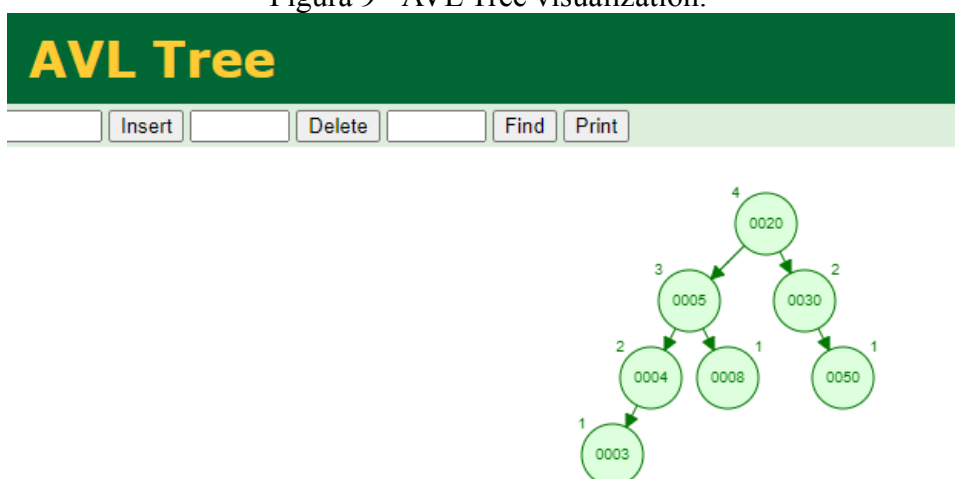
Fonte: elaborado pelo o autor.

### 3.2.1 Ferramenta de validação

Para validar se a estrutura da árvore estava sendo construída corretamente e se as rotações estavam sendo aplicadas como esperado, utilizamos a ferramenta online AVL Tree Visualization disponível em <https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>

Essa ferramenta permite acompanhar visualmente a inserção e o balanceamento de nós em uma árvore AVL. É possível observar rotações simples e duplas, e a estrutura final da árvore, facilitando a conferência da implementação.

Figura 9 - AVL Tree visualization.



Fonte: elaborado pelo o autor.

### 3.3 Função buscar elemento

A imagem abaixo exibe a funcionalidade de busca por um elemento. O usuário escolhe a opção 2 e digita a chave "50". O sistema percorre a árvore e localiza o valor com sucesso. Como resultado, é impresso na tela o caminho percorrido durante a busca ( $20 \Rightarrow 30 \Rightarrow \{50\}$ ) e uma mensagem de confirmação de que a chave foi encontrada.

Figura 10 - Busca por uma chave existente.

```
MENU
1 - Adicionar elemento
2 - Buscar elemento
3 - Excluir elemento
4 - Mostrar árvore
5 - Sair
Escolha uma opção: 2
Digite a chave a ser buscada: 50
Caminho até a chave 50: 20  $\Rightarrow$  30  $\Rightarrow$  {50} | [OK] Chave encontrada!
```

Fonte: elaborado pelo o autor.

A imagem abaixo demonstra o resultado de uma busca por um elemento inexistente na árvore. O usuário seleciona a opção 2 para buscar a chave "80". O sistema percorre o caminho  $20 \Rightarrow 30 \Rightarrow 50$ , que é o mais longo possível na direção da chave procurada. Ao final do caminho, como a chave não é encontrada, o programa exibe a mensagem de falha "[X] Chave não encontrada!".

Figura 11- Busca por uma chave não existente.

```
MENU
1 - Adicionar elemento
2 - Buscar elemento
3 - Excluir elemento
4 - Mostrar árvore
5 - Sair
Escolha uma opção: 2
Digite a chave a ser buscada: 80
Caminho até a chave 80: 20  $\Rightarrow$  30  $\Rightarrow$  50  $\Rightarrow$  [X] Chave não encontrada!
```

Fonte: elaborado pelo o autor.



### 3.4 Função excluir elemento

A imagem exibe a remoção de um elemento que causa um desbalanceamento na Árvore AVL, exigindo uma rotação para corrigi-lo. O usuário escolhe a opção 3 e remove a chave "50". O sistema informa que o nó 30 permaneceu balanceado, mas que foi necessária uma Rotação Simples à Direita (LL) no nó 20 para rebalancear a árvore. Após a rotação, o sucesso da remoção é confirmado e a nova estrutura da árvore é exibida.

Figura 12 - Excluindo elemento da árvore.

```
MENU
1 - Adicionar elemento
2 - Buscar elemento
3 - Excluir elemento
4 - Mostrar árvore
5 - Sair
Escolha uma opção: 3
Digite a chave a ser removida: 50
[OK] Nó 30 está balanceado após remoção (B=0)
Rotação simples à direita (LL) no nó 20 após remoção
Chave removida com sucesso.
      30 (B=0)
     /  \
    20 (B=0)  8 (B=0)
   /  \
  5 (B=0)  #
 /  \
4 (B=-1) 3 (B=0)
```

Fonte: elaborado pelo o autor.

### 3.5 Função mostrar árvore

A imagem demonstra a funcionalidade "*Mostrar árvore*", acionada pelo usuário ao escolher a opção 4. Em resposta, o sistema imprime uma representação visual da estrutura atual da árvore. O formato utiliza indentação e linhas para mostrar a hierarquia dos nós, exibindo também o fator de balanceamento ( $B$ ) de cada um. A árvore mostrada é o resultado final após a operação de remoção e rotação do exemplo anterior.

Figura 13- Função que exibe a árvore no terminal.

```
1 - Adicionar elemento
2 - Buscar elemento
3 - Excluir elemento
4 - Mostrar árvore
5 - Sair
Escolha uma opção: 4

Árvore AVL:
      30 (B=0)
     /  \
    20 (B=0)  8 (B=0)
   /  \
  5 (B=0)  #
   \
    4 (B=-1)
     \
      3 (B=0)
Altura da arvore: 3
```

Fonte: elaborado pelo o autor.

#### 4 RELATO DE DESENVOLVIMENTO

O desenvolvimento começou pela estrutura básica do nó da árvore, com atributos *key*, *left*, *right* e *height*. Em seguida, foram implementadas as operações básicas de inserção e busca. O passo seguinte foi implementar as rotações e os métodos auxiliares para garantir o balanceamento após inserções. Por fim, foram adicionadas as funções visuais como *bshow()* e *search\_path()*

#### 5 PONTOS DE DIFICULDADE ENCONTRADOS

Um dos principais desafios foi entender e aplicar corretamente as rotações da árvore, especialmente as rotações duplas (esquerda-direita e direita-esquerda). Além disso, implementar a função *remove* exigiu cuidado especial para manter o balanceamento da árvore após a remoção de um nó com dois filhos. Mas foi bastante interessante e desafiador o trabalho, valeu a dor de cabeça.