

**ALEXANDRE ROCHA FONTE
JHENIFER PATRICIA GOMES DA SILVA
YAN VINICIUS SILVA**



FUNDAÇÃO HERMÍNIO OMETTO

**DESENVOLVIMENTO DE UMA ÁRVORE AVL EM C#
UTILIZANDO PROGRAMAÇÃO ORIENTADA A OBJETOS**

ARARAS/SP

Junho/2025

SUMÁRIO

1.INTRODUÇÃO.....	2
2.DIAGRAMA DE CLASSES (UML).....	2
3. DESCRIÇÃO DAS CLASSES.....	3
3.1 CLASSE PROGRAM.....	3
3.2 CLASSE NO.....	4
3.3 CLASSE ARVOREAVL.....	4
3.4 CLASSE ARQUIVOAVL.....	5
4. FUNCIONAMENTO GERAL DO SISTEMA.....	6
5. COMO UTILIZAR O PROGRAMA.....	6
5.1. MODO COM ARQUIVO PADRÃO.....	6
5.2. MODO COM ARQUIVO PERSONALIZADO (CAMINHO ESPECÍFICO).....	7
5.3. ESTRUTURA DO ARQUIVO DE COMANDOS.....	7
5.4. EXIBIÇÃO NO PROMPT.....	8
6. CONSIDERAÇÕES FINAIS.....	9
7. REFERÊNCIAS.....	9

1.INTRODUÇÃO

Este relatório tem como objetivo apresentar o desenvolvimento e a implementação de uma estrutura de dados do tipo Árvore AVL, utilizando a linguagem de programação C#. Este trabalho foi desenvolvido como parte dos requisitos da disciplina de Programação Orientada a Objetos, com o intuito de aplicar na prática os conceitos aprendidos ao longo do semestre.

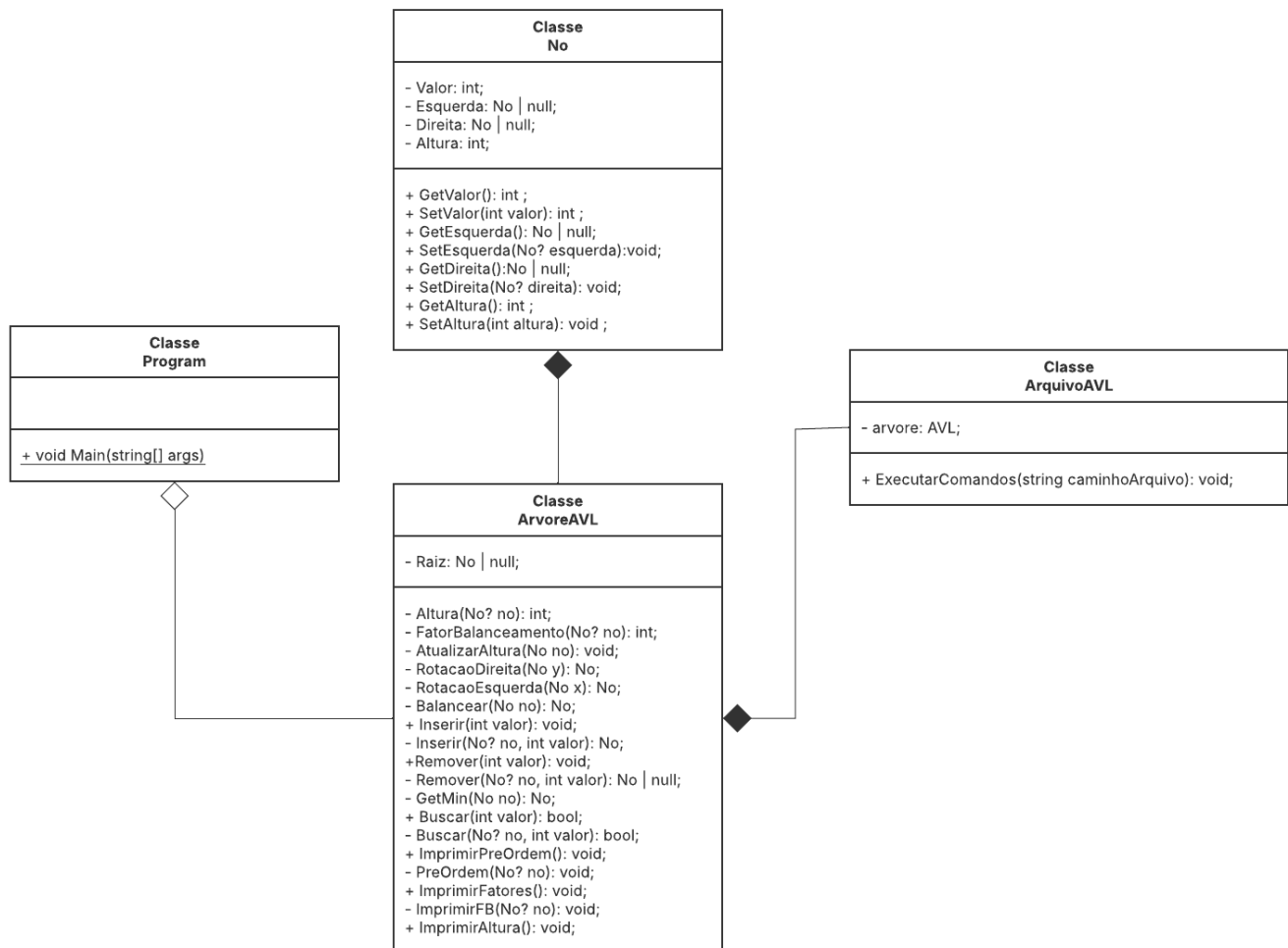
A escolha da árvore AVL se dá por ela ser uma árvore binária de busca que se mantém sempre balanceada. O balanceamento garante que as operações de busca, inserção e remoção sejam realizadas com alta eficiência, mantendo a complexidade de tempo no pior caso em $O(\log n)$. Esse tipo de estrutura é fundamental em sistemas que exigem consultas rápidas e manipulação eficiente de dados.

O desenvolvimento deste projeto permite ao aluno não só compreender os conceitos fundamentais da orientação a objetos, como também entender profundamente os princípios de estruturas de dados, algoritmos de balanceamento e a importância da modularização no desenvolvimento de software.

2.DIAGRAMA DE CLASSES (UML)

O diagrama de classes é uma representação visual da estrutura do sistema. Nele estão representadas as classes, seus atributos, métodos e os relacionamentos entre elas. Através do diagrama é possível visualizar claramente como as classes interagem, como estão organizadas e quais são as responsabilidades de cada uma.

O diagrama a seguir foi elaborado para representar a implementação da árvore AVL. Observa-se que cada classe possui uma função bem definida, respeitando os princípios da programação orientada a objetos, como abstração, encapsulamento, responsabilidade única e reutilização de código.



3. DESCRIÇÃO DAS CLASSES

3.1 CLASSE PROGRAM

A classe Program é a classe principal da aplicação, responsável por inicializar o sistema. Ela contém o método Main, que funciona como ponto de entrada do programa.

Nessa classe, o usuário pode interagir com a árvore AVL, seja por meio do console, inserindo comandos manualmente, ou utilizando arquivos de texto que contêm sequências de comandos. A classe faz uso dos métodos da árvore AVL para inserir, remover, buscar e visualizar os nós, além de permitir a execução de comandos automatizados.

Essa classe é essencial, pois coordena a execução de todo o sistema e garante que as funcionalidades estejam acessíveis ao usuário.

3.2 CLASSE NO

A classe No (ou Nó) é a representação de cada elemento dentro da árvore AVL. Cada nó contém dados e referências para seus filhos esquerdo e direito, além de uma informação crucial para o balanceamento da árvore: sua altura.

Atributos:

- **Valor (int):** Armazena o valor inteiro do nó.
- **Esquerda (No):** Referência para o nó filho da esquerda.
- **Direita (No):** Referência para o nó filho da direita.
- **Altura (int):** Altura do nó, fundamental para o cálculo do fator de balanceamento.

Responsabilidades:

- Armazenar os dados de cada elemento da árvore.
- Manter as referências aos filhos, permitindo a construção da estrutura binária.
- Fornecer informações necessárias para manter a árvore balanceada.

3.3 CLASSE ARVOREAVL

A classe ArvoreAVL é o núcleo da aplicação. Ela contém toda a lógica de funcionamento da árvore AVL, incluindo os métodos para inserir, remover e buscar elementos, além dos métodos responsáveis por garantir o balanceamento da árvore.

Atributo:

- **Raiz (No):** Nó raiz da árvore, ponto inicial para todas as operações.

Métodos Privados:

- **Altura(No):** Retorna a altura de um nó específico.
- **FatorBalanceamento(No):** Calcula o fator de balanceamento, que é a diferença entre as alturas dos filhos esquerdo e direito.

- **AtualizarAltura(No):** Atualiza a altura de um nó após alterações na árvore.
- **RotacaoDireita(No):** Realiza uma rotação simples à direita.
- **RotacaoEsquerda(No):** Realiza uma rotação simples à esquerda.
- **Balancear(No):** Avalia o fator de balanceamento e executa as rotações necessárias para manter a árvore equilibrada.

Métodos Públicos:

- **Inserir(int valor):** Insere um novo elemento na árvore, mantendo-a balanceada.
- **Remover(int valor):** Remove um elemento, reequilibrando a árvore se necessário.
- **Buscar(int valor):** Verifica se um valor existe na árvore.
- **ImprimirPreOrdem():** Mostra a árvore em percurso pré-ordem.
- **ImprimirFatores():** Mostra o fator de balanceamento de cada nó.
- **ImprimirFB(No):** Mostra o fator de um nó específico.
- **ImprimirAltura():** Mostra a altura total da árvore.

3.4 CLASSE ARQUIVOAVL

A classe ArquivoAVL permite que os comandos sejam executados a partir de um arquivo texto. Essa abordagem automatiza processos, facilitando testes e a manipulação de grandes volumes de dados.

Atributo:

- **arvore (ArvoreAVL):** Instância da árvore AVL que será manipulada.

Método Principal:

- **ExecutarComandos(string caminhoArquivo):** Lê um arquivo linha por linha e executa os comandos que estão nele, como **INSERIR**, **REMOVER**, **IMPRIMIR**, etc.

Vantagens:

- Possibilita testes rápidos e automáticos.
- Permite executar grandes volumes de operações sem interação manual constante.
- Facilita a demonstração do funcionamento da árvore para diferentes conjuntos de dados.

4. FUNCIONAMENTO GERAL DO SISTEMA

O programa foi projetado para permitir ao usuário a criação, manipulação e visualização de uma árvore AVL. A cada operação de inserção ou remoção, o sistema verifica automaticamente o fator de balanceamento dos nós e realiza as rotações necessárias (simples ou duplas) para garantir que a árvore permaneça balanceada.

Essa automação assegura que as operações de busca, inserção e remoção tenham uma complexidade de $O(\log n)$, mesmo após inúmeras operações consecutivas. O sistema é robusto, modularizado e segue rigorosamente os princípios da programação orientada a objetos, garantindo clareza, manutenção facilitada e escalabilidade.

5. COMO UTILIZAR O PROGRAMA

O programa desenvolvido em C# para a manipulação da árvore AVL pode ser executado de duas formas diferentes: **modo padrão com arquivo fixo** ou **modo com arquivo personalizado**, ambos utilizando o terminal através do comando `dotnet run`. Em ambas as formas, é essencial seguir rigorosamente a estrutura correta dos comandos no arquivo `.txt`, pois qualquer erro de formatação pode causar falhas na execução.

5.1. MODO COM ARQUIVO PADRÃO

Passos para executar:

1. Garanta que o arquivo **comandos.txt** esteja salvo na raiz do projeto.
2. No terminal, dentro da pasta do projeto, execute:

```
dotnet run
```

O programa irá automaticamente buscar pelo arquivo padrão, ler os comandos e executar todos na sequência.

5.2. MODO COM ARQUIVO PERSONALIZADO (CAMINHO ESPECÍFICO)

Este modo permite ao usuário escolher qualquer arquivo de texto, desde que esteja corretamente estruturado, e fornecer seu caminho diretamente no comando de execução.

Passos para executar:

1. Crie um arquivo `.txt` contendo os comandos no formato correto.
2. No terminal, execute o programa indicando o caminho do arquivo. Por exemplo:

```
dotnet run "Caminho/Para/Seu/Arquivo.txt"
```

5.3. ESTRUTURA DO ARQUIVO DE COMANDOS

O arquivo de texto deve conter um comando por linha, utilizando letras para as operações e, quando necessário, um valor numérico.

Comandos válidos:

- **I X** → Inserir o valor **X** na árvore.
- **R X** → Remover o valor **X** da árvore.
- **B X** → Buscar o valor **X** na árvore (retorna se encontrado ou não).
- **P** → Imprimir a árvore no percurso pré-ordem.

- **F** → Imprimir os fatores de balanceamento de todos os nós.
- **H** → Imprimir a altura total da árvore.

```
I 30
I 20
I 40
I 25
B 20
R 40
P
F
H
```

Se o arquivo não seguir esse formato — como faltar o espaço entre o comando e o valor ou usar comandos inexistentes — o programa emitirá um aviso de “**Comando desconhecido**” e pulará aquele comando.

5.4. EXIBIÇÃO NO PROMPT

Ao executar o programa, os resultados das operações realizadas na árvore AVL serão exibidos diretamente no prompt (terminal). Cada comando gera uma saída específica, conforme sua função.

Abaixo, será possível visualizar exemplos dessas saídas através de capturas de tela (prints).

Comandos.txt, Exibição Prompt:

```
comandos.txt
1 I 10
2 I 20
3 I 30
4 P
5 I 15
6 P
7 R 20
8 P
9 B 15
10 B 25
11 F
12 H
```

```
*****
Arquivo não selecionado, usando o arquivo padrão 'comandos.txt'.
*****
Árvore em pré-ordem: 20 10 30
Árvore em pré-ordem: 20 10 15 30
Árvore em pré-ordem: 15 10 30
Valor encontrado
Valor não encontrado
Fatores de balanceamento:
Nó 15: Fator de balanceamento 0
Nó 10: Fator de balanceamento 0
Nó 30: Fator de balanceamento 0
Altura da árvore: 2
*****
```

6. CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto foi extremamente enriquecedor, permitindo uma aplicação prática dos conceitos de **orientação a objetos** e das técnicas de implementação de **estruturas de dados balanceadas**.

A árvore AVL é uma estrutura eficiente que garante o balanceamento automático, proporcionando excelente desempenho em sistemas que exigem alta performance na busca e manipulação de dados.

Além dos conhecimentos técnicos, o projeto também fortaleceu habilidades de organização de código, modularização, abstração e reaproveitamento de classes. O uso da linguagem **C#** foi fundamental para aplicar de forma clara os conceitos da POO.

7. REFERÊNCIAS

- Repositório GitHub - Árvore AVL em C#: <https://github.com/YanSilva22/Arvore-AVL-/tree/master>
- Documentação oficial do C#: <https://learn.microsoft.com/pt-br/dotnet/csharp/>