FHO – Fundação Hermínio Ometto
Orientador: Prof. Maurício Acconcia Dias
Análise e projetos orientados a objetos I

ENGENHARIA DA COMPUTAÇÃO – 7° PERÍODO

Caroline da Silva Grizante – RA: 114105
Emilly Emanuelly R. dos Santos – RA: 114095
Marcela Lovatto – RA: 113926

IMPLEMENTAÇÃO DE UMA ÁRVORE AVL EM C# COM INTERFACE GRÁFICA

Araras - SP
2025

RESUMO

Este trabalho apresenta o desenvolvimento de uma estrutura de dados do tipo árvore AVL utilizando a linguagem C#, com foco na eficiência e balanceamento automático da árvore após inserções e remoções. O projeto é composto por três componentes principais: uma biblioteca de classes contendo a lógica da árvore AVL, uma aplicação console com menu interativo e uma aplicação WPF para visualização gráfica em tempo real. Além de garantir eficiência computacional com complexidade O(log n), a aplicação oferece uma abordagem didática para aprendizado de estruturas de dados. O relatório também aborda os conceitos teóricos por trás da árvore AVL, o processo de desenvolvimento, os testes realizados e a análise dos resultados obtidos.

Palavras-chave: Árvore AVL, Estrutura de Dados, C#, Balanceamento.

Sumário

1. INTRODUÇÃO

A eficiência no armazenamento e recuperação de dados é um dos pilares da Ciência e da Engenharia da Computação. Neste contexto, estruturas de dados como as árvores binárias de busca desempenham um papel fundamental. No entanto, quando essas estruturas se tornam desbalanceadas, o desempenho das operações básicas pode degradar significativamente.

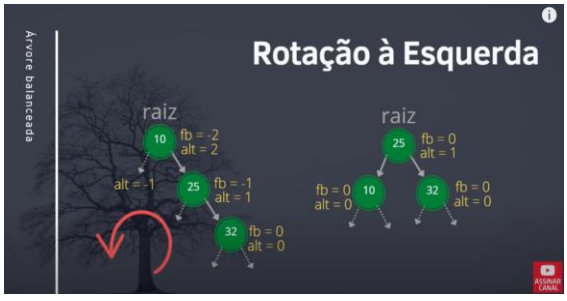
A estrutura de dados denominada Árvore AVL é uma variação da árvore binária de busca, com a característica essencial de manter-se balanceada após as operações de inserção e remoção de elementos. Esse balanceamento é fundamental para garantir a eficiência do tempo de execução das operações de busca, inserção e exclusão, mantendo complexidade logarítmica, isto é, O(log n).

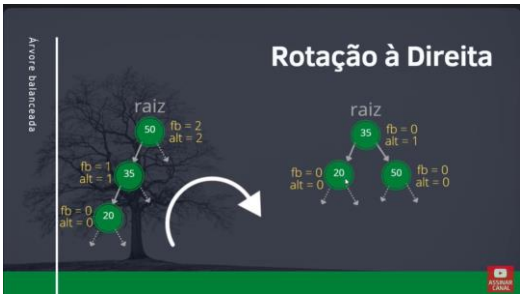
Em árvores binárias de busca comuns, a inserção sequencial de elementos pode levar a um formato degenerado, similar a uma lista encadeada, comprometendo o desempenho esperado. A proposta da Árvore AVL surgiu em 1962, idealizada por Georgy Adelson-Velsky e Evgeny Landis. Suas iniciais batizaram esta estrutura.

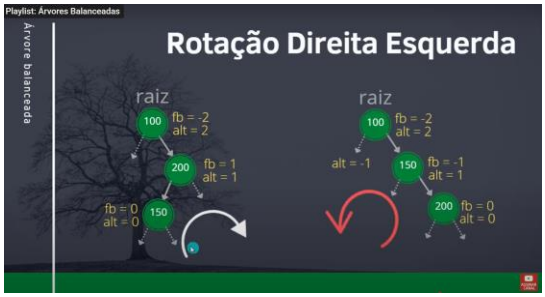
Este trabalho tem como objetivo implementar essa estrutura em C#, utilizando técnicas de orientação a objetos, além de prover uma visualização interativa através da plataforma WPF.

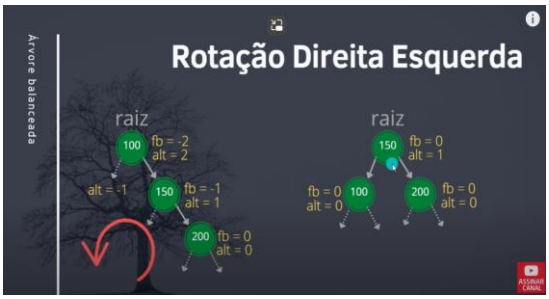
2. FUNDAMENTAÇÃO TEÓRICA

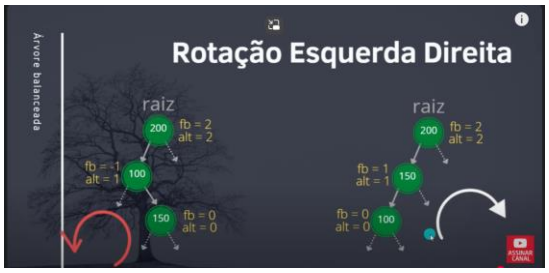
A árvore AVL é uma árvore binária de busca que mantém sua altura balanceada através do cálculo do Fator de Balanceamento (FB), definido como a diferença entre as alturas das subárvores esquerda e direita de um nó. Quando esse valor excede os limites de -1 a 1, a árvore necessita ser rebalanceada por meio de rotações específicas para restabelecer o equilíbrio. Essas rotações são divididas em quatro categorias principais: **rotação simples à esquerda**, **rotação simples à direita**, **rotação dupla à esquerda** (direita-esquerda) e **rotação dupla à direita** (esquerda-direita). A escolha da rotação adequada depende do tipo de desbalanceamento identificado durante a inserção ou remoção de nós. Com isso, a árvore mantém a complexidade O(log n) para inserção, remoção e busca.

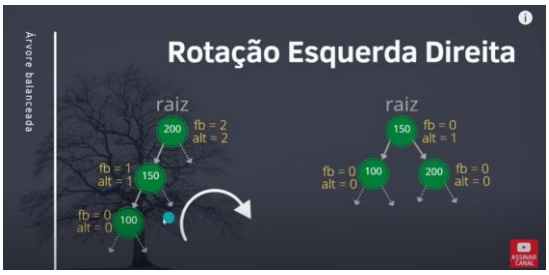












3. METODOLOGIA

O desenvolvimento do projeto foi dividido em três componentes principais:
- Biblioteca de Classes (Class Library): contém as definições das classes No e Arvore.
- Aplicativo Console: responsável por receber comandos do usuário e manipular a árvore AVL.
- Aplicação WPF: utilizada para exibir a estrutura da árvore graficamente.

Além disso, foi utilizado um arquivo de texto externo (dados.txt) para entrada de dados, permitindo testar a estrutura com diferentes conjuntos de valores.

4. DESENVOLVIMENTO E IMPLEMENTAÇÃO

O código foi dividido em três partes para facilitar manutenção e entendimento:

4.1. Classe `No.cs`

A classe `No` representa um elemento da árvore, contendo:

- Dois atributos do tipo inteiro, que vai ser o Valor e a Altura
- Duas variáveis do tipo No. Em um que vai conter as informações do nó à Direita, e outro para o nó à Esquerda, sendo que os dois podem ser vazios.
- E o método Construtor da classe.

A construção de um novo nó inicializa a altura como 1, e os ponteiros para os filhos como nulos.

Estrutura: A classe No vai representar uma “folha” da árvore, ela vai conter obrigatoriamente um valor. E pode ter outros nós conectados a ela, podendo ser do lado esquerdo, direito ou sem conexão alguma. Ela contém quatro atributos essenciais:

- Valor: Armazena o conteúdo do nó, sendo um valor do tipo inteiro.
- Altura: Esta variável vai armazenar o número de arestas do caminho mais longo da raiz até um nó.
- No Esquerda: Vai conter as informações de um nó que fica à esquerda do nó de origem, podendo ser nulo.
- No Direita: Vai armazenar as informações de um nó que está à direita do nó de origem, e pode ser nulo também.

  Além de conter um método construtor. Segue abaixo a implementação:

4.2. Classe `Arvore.cs`

Estrutura: Esta classe vai conter todos os métodos essenciais para construir a árvore, ou seja, vai conter os métodos para inserir, remover e buscar nós. Ademais vai ter outros métodos que irão realizar os cálculos, para garantir que está árvore esteja balanceada. E se caso for necessário, realizar uma rotação.

- **Inserir:** É o método público, ou seja, que pode ser chamado fora da classe e apenas passar o valor que deseja inserir. A inserção não acontece efetivamente neste método, ela vai delegar este trabalho ao método recursivo.
- **No Inserir:** Este vai ser o método recursivo que realmente vai inserir o novo nó na posição correta da árvore. Vai percorrer toda árvore, e analisar qual a posição correta para inserir ele. Sendo um método privado, podendo ser acessado apenas por aquela classe.
- **Remover:** Trata-se de um método público, que pode ser chamado fora da classe de origem. E ele vai apenas receber o valor que deseja que seja removido, e chamar o método recursivo.
- **No Remover:** Este método vai percorrer toda a árvore em busca do valor que foi passado como parâmetro, e ao encontrá-lo vai excluí-lo. Considerando o balanceamento, e atualizando o valor da altura.
- **EncontrarMenor:** O método em questão é do tipo No, que tem o intuito de encontrar o nó com o menor valor da árvore binária.
- **Buscar:** Refere-se a um método do tipo No, que vai buscar o nó a partir do valor que ele contém. Ele lida com a possibilidade de o nó não existir, ou estar a esquerda da árvore(menor), ou a direita da árvore (maior).
- **Existe:** Apenas vai chamar o método recursivo Buscar, ele é do tipo bool.
- **ImprimirInOrder:** Irá existir dois métodos com este mesmo nome, mas com funcionalidades diferentes. O primeiro método é público, devido a isto pode ser solicitado fora da classe, e será o responsável por chamar o método recursivo privado. E o outro método vai ser privado, ela será encarregada por imprimir a árvore em Ordem, que seria a exibição dos nós na seguinte ordem: Esquerda-> Raiz -> Direita.

- **ImprimirPreOrdem:** Será definido dois métodos com esse nome, cada um com um objetivo distinto. O primeiro será público, para que possa ser acessado em um ambiente externo a esta classe. Além de invocar o método recursivo privado. O segundo será privado, responsável por exibir a árvore em Pré-Ordem, ou seja, os nós irão ser apresentados na sequência: Raiz-> Esquerda -> Direita.

- **ImprimirPosOrdem:** O método ImprimirPosOrdem será implementado em duas versões diferentes. A primeira versão, será de acesso público, o que permite a sua chamada por outras classes/métodos. Tendo como principal objetivo chamar o método recursivo privado. Outrossim, a segunda versão é responsável por exibir os elementos da árvore na ordem Pós-Ordem, que seria na seguinte sequência: Esquerda-> Direita-> Raiz.

- **VerificarBalanceamento:** Também vai ser composto por dois métodos. Um método vai ser público, com o intuito de chamar a função recursiva. E o segundo, vai ser um método privado, em que vai calcular o fator de balanceamento de cada nó da árvore.
- **ImprimirArvoreVisual:** Irá utilizar a recursão para percorrer e exibir os nós que estão na árvore. Imprime a árvore com uma estrutura de diretórios.
- **AtualizarAltura:** Trata-se de um método privado. Em que após a cada operação realizada, como inserção, remoção ou rotação, precisamos chamar este método para atualizar a altura.
- **FatorBalanceamento:** Calcula o Fator de Balanceamento, realizando a subtração da altura da esquerda com a altura da direita.
- **Balancear:** Este método é do tipo No e privado, ele vai ser responsável por analisar o fator de balanceamento dos nós. E decidir se vai ser feito uma rotação simples, ou dupla.
- **RotacionarDireita:** É classificada como um método do tipo Nó, que tem como objetivo corrigir o desbalanceamento, que foi causado após a inserção de um novo nó à esquerda da árvore.
- **RotacionarEsquerda:** Consiste em um método do tipo Nó, que tem o intuito de reparar o desbalanceamento gerado pela inserção de um novo nó à direita da árvore.

Resumidamente, classe `Arvore` concentra a lógica da AVL. Dentre os métodos implementados, destacam-se:

- **Inserção e Remoção**: realizadas por métodos recursivos, com atualização da altura e verificação de balanceamento;
- **Rotação e Rebalanceamento**: a estrutura executa automaticamente as rotações quando o fator de balanceamento ultrapassa os limites permitidos;
- **Busca e Existência**: verifica a presença de valores na árvore;
- **Impressões**: exibe a árvore nas ordens InOrder, Pré-Ordem e Pós-Ordem;
- **Visualização**: representação hierárquica visual da estrutura;
- **Verificação de Balanceamento**: exibe o fator de balanceamento de cada nó.

4.3. Classe `Program.cs`

Estrutura: Este é o ponto de partida da aplicação. Através dele:

- A árvore é inicializada;
- Os dados do arquivo `dados.txt` são carregados;
- Um menu interativo é exibido ao usuário, permitindo executar todas as operações citadas.

O menu utiliza a estrutura `do-while`, e cada opção invoca métodos específicos da classe `Arvore`, garantindo a atualização e o equilíbrio da estrutura após cada operação. Todas as ações irão passar por checagens, para desta forma garantir a integridade dos dados. Como por exemplo, nas operações Inserir

e Remover, após fazer as alterações necessárias, irá passar por uma verificação no balanceamento. Em que se caso esteja com um valor diferente de -1, 0, 1 ele irá analisar qual rotação será realizada.

Além disso, possibilita uma visualização mais clara da sequência dos elementos e das diferentes ordens de exibição da árvore.

4.4. Leitura de Arquivo

A leitura dos dados de entrada ocorre logo na inicialização do sistema. Caso o arquivo não esteja presente, a árvore será iniciada vazia.

4.5. Menu Interativo

A interface em console fornece todas as ferramentas necessárias para manipular a árvore AVL, com foco em didática e praticidade.

Em suma, classe No contém atributos que armazenam o valor do nó, sua altura, e referências para os filhos esquerdo e direito. Já a classe Arvore implementa todos os métodos de manipulação da estrutura AVL. O Program.cs é o ponto de entrada da aplicação, onde é exibido um menu de interação com o usuário para realizar inserções, buscas, remoções e impressões da árvore.

5. RESULTADOS E TESTES

O funcionamento do menu interativo permite ao usuário executar dinamicamente as seguintes operações:

1. **Inserir valor**
2. **Remover valor**
3. **Buscar valor**
4. **Imprimir em ordem (InOrder)**
5. **Imprimir em pós-ordem (Pós-Ordem)**
6. **Imprimir em pré-ordem (Pré-Ordem)**
7. **Verificar balanceamento**
8. **Visualizar a árvore**
9. **Sair**

Cada uma dessas opções é validada para evitar erros de entrada. Após qualquer modificação na árvore, o sistema realiza a verificação do fator de balanceamento e executa as rotações adequadas. Então, a

estrutura é inicializada com os valores contidos no arquivo externo `dados.txt`, onde cada linha representa um valor a ser inserido. Com base em testes utilizando o arquivo de entrada abaixo, foi possível verificar o balanceamento automático da árvore AVL:
Conteúdo do arquivo dados.txt:

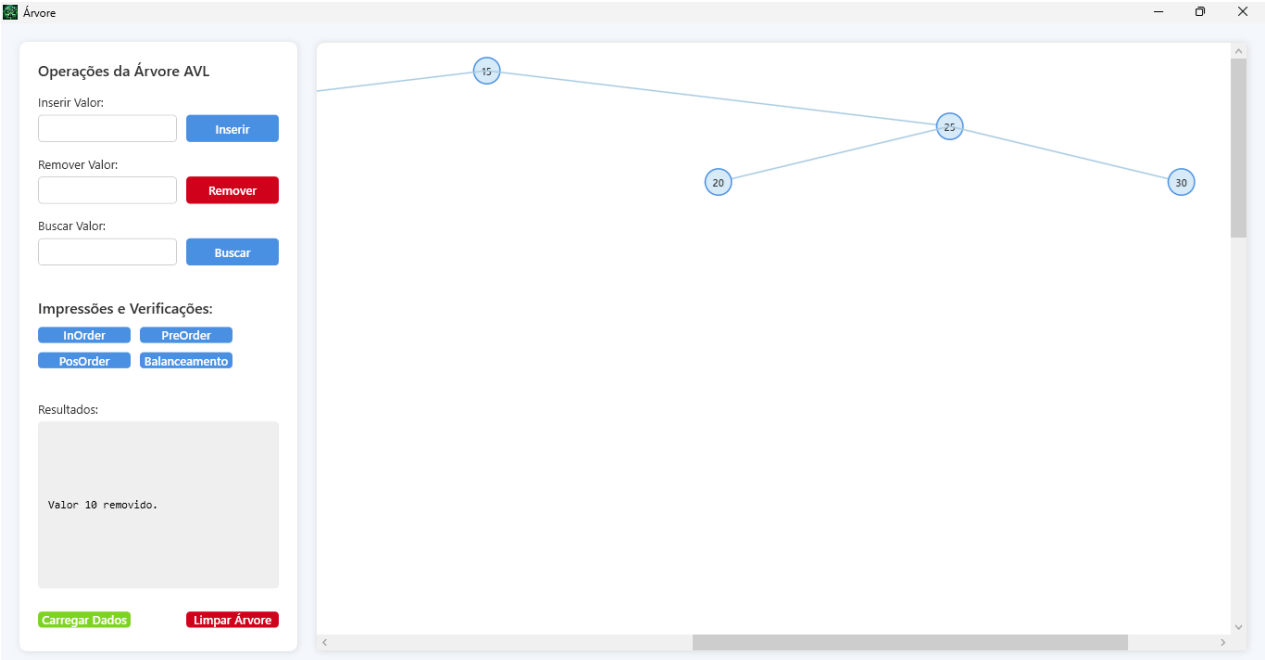
```
10
20
5
6
15
30
25
```

A árvore resultante foi:

```
      10
    /    \
   5      25
    \     / \
     6   20  30
         /
       15
```

Este formato demonstra que a árvore se manteve balanceada após cada inserção, preservando a complexidade logarítmica e a integridade da estrutura.

As imagens a seguir ilustram o funcionamento do sistema:



Impressões e Verificações:

InOrder PreOrder

PosOrder Balanceamento

Resultados:

```
InOrder: 5, 6, 15, 20, 25, 30
```

Impressões e Verificações:

InOrder PreOrder

PosOrder Balanceamento

Resultados:

```
PreOrder: 15, 5, 6, 25, 20, 30
```

Impressões e Verificações:

InOrder PreOrder

PosOrder Balanceamento

Resultados:

```
PosOrder: 6, 5, 20, 30, 25, 15
```

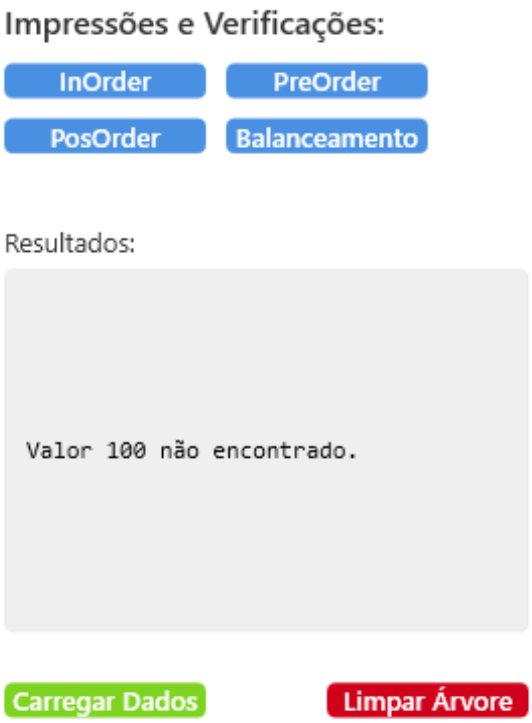
Impressões e Verificações:

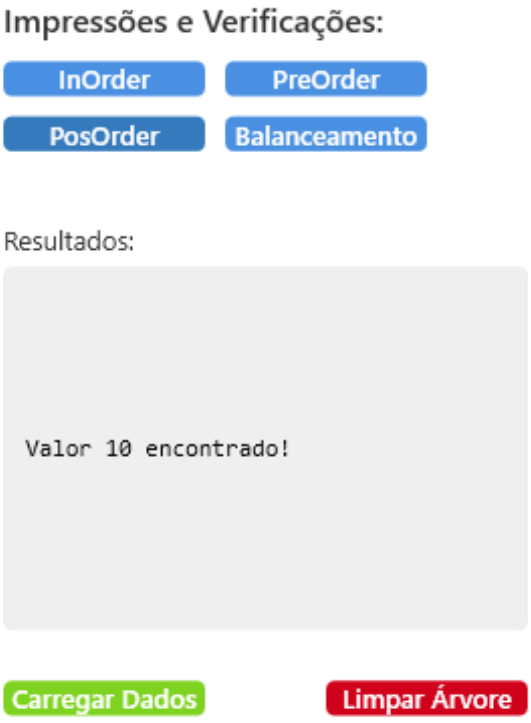
InOrder PreOrder

PosOrder Balanceamento

Resultados:

```
Fatores de Balanceamento:
Nó 5 - FB = -1
Nó 6 - FB = 0
Nó 15 - FB = 0
Nó 20 - FB = 0
Nó 25 - FB = 0
Nó 30 - FB = 0
```





Após o desenvolvimento e execução do projeto, foi possível verificar os seguintes resultados:

- Os dados foram corretamente carregados a partir de um arquivo externo;
- A árvore permaneceu balanceada após inserções e remoções;
- O programa permitiu a visualização da árvore em diferentes ordens;
- O balanceamento foi validado através dos fatores exibidos;
- A representação visual forneceu uma visão clara da estrutura da árvore.

6. CONCLUSÃO

A implementação de uma árvore AVL em C# com visualização gráfica permite compreender, de forma prática, o funcionamento de estruturas de dados balanceadas. O projeto oferece uma base sólida para estudos futuros em algoritmos, interfaces gráficas e desempenho computacional.

A implementação ilustra de forma prática como o balanceamento em estruturas de dados pode impactar significativamente o desempenho das operações. A utilização do fator de balanceamento, aliada às rotações apropriadas, garante que a estrutura mantenha sua eficiência, mesmo após grandes quantidades de inserções ou remoções.

A interface em console proporcionou uma experiência interativa ao usuário, permitindo a exploração prática do comportamento da árvore em diferentes cenários. Este projeto reforça a importância do equilíbrio em árvores binárias. A aplicabilidade de conceitos de orientação a objetos reforçam sua utilidade no contexto acadêmico e oferece uma base sólida para estudos futuros em estruturas de dados mais complexas.

7. REFERÊNCIAS

- GITHUB - TreeAVL-csharp. Disponível em: https://github.com/CarolineGrizante/TreeAVL-csharp
- CORMEN, T. H. et al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.
- Documentação oficial do C#: https://learn.microsoft.com/en-us/dotnet/csharp/
- Notas de aula e conteúdos fornecidos na disciplina Análise e Projeto Orientado a Objetos I.