Guia 1

Análise e projetos orientados a objetos I

Curso: Engenharia da Computação

Prof.: Maurício Acconcia Dias

Data: 15/04/2025

Árvore AVL

Caroline da Silva Grizante Ra:114105

Emilly Emanuelly R. dos Santos Ra:114095

Marcela Lovatto Ra:113926

Fundação Hermínio Ometto

Araras-SP

Sumário

1. Introdução
2. Estrutura do Projeto
3. Funcionamento do Menu
4. Exemplo de Entrada
5. Funcionalidades Implementadas
6. Observação
7. Resultados
8. Conclusão
9. Referenciais Utilizados

---

1. Introdução

Este relatório aborda a construção de uma importante estrutura de dados, que é a Árvore AVL. Esta estrutura é um tipo de árvore binária de busca, com o diferencial de manter o equilíbrio desta árvore após inserções e remoções de um elemento.

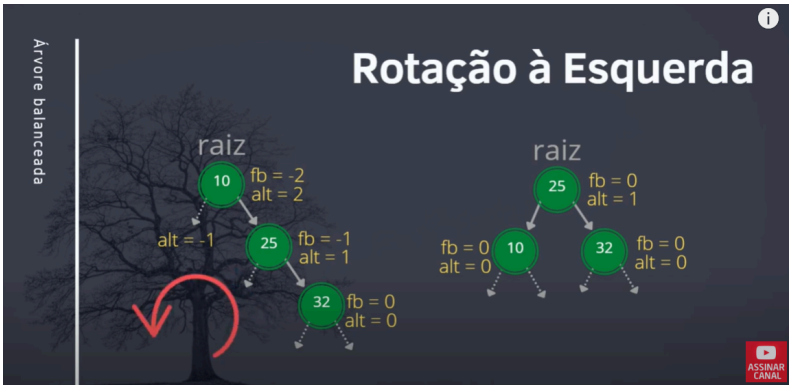
Esta estrutura segue o mesmo raciocínio lógico de uma árvore balanceada, ou seja, se o elemento adicionado for maior que a raiz vai ser inserido à direita da árvore. Se o elemento for menor, vai ser posicionado à esquerda da árvore. Contudo após tantos processos pode ser que a árvore fique degenerada, afetando o seu desempenho e a tornando lenta.

Devido a isso, em 1962 este formato de estrutura de dados foi proposto por dois matemáticos soviéticos Georgy Adelson-Velsky e Evgeny Landis, e por causa da iniciais de seus sobrenomes, a árvore foi nomeada como AVL. Tendo como objetivo otimizar e melhorar o desempenho na inserção, busca e remoção de elementos, tendo complexidade logarítmica de O(log n).

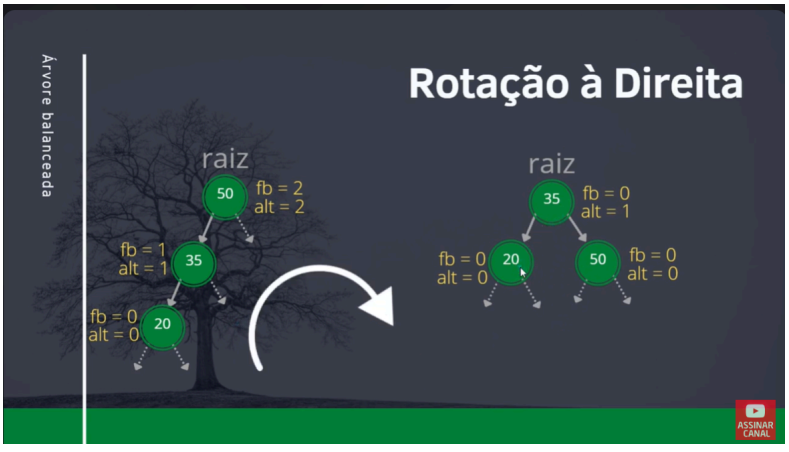
Para garantir este equilíbrio é calculado o Fator de Balanceamento de cada nó na árvore. Esta variável irá ser calculada a partir da subtração entre a altura da subárvore à esquerda, e a altura da subárvore à direita. Tendo como valores válidos: 0, 1 e -1. Se chegar em um resultado diferente, indica que precisamos balancear a nossa árvore.

Se for valores maiores que 1, o desbalanceamento está presente do lado direito da árvore. E se for menor que -1, o desequilíbrio está do lado esquerdo dela.

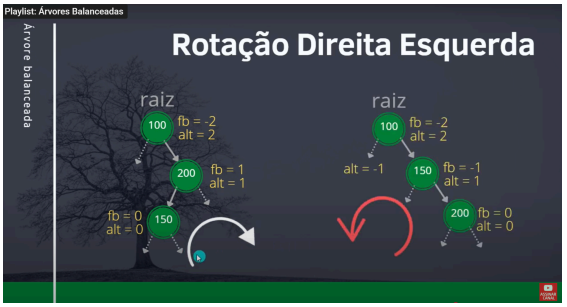
- **Rotações à Esquerda:** A rotação é realizada por meio da manipulação de ponteiros. O ponteiro à esquerda do nó anterior é direcionado para o primeiro nó, enquanto o nó anterior passa a ocupar a posição do primeiro nó. Essa operação é aplicada sempre que o fator de balanceamento for inferior a -1.

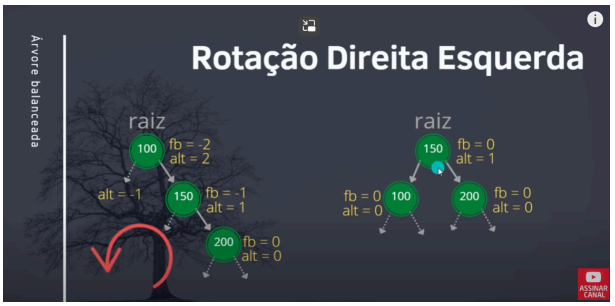


- **Rotações à Direita:** O fator de balanço (fb) do nó pai assume o valor 2, indicando que o nó pai deve ser reposicionado para o nó localizado à direita do seu antecessor. Assim, o nó antecessor passa a exercer a função de nó pai. Dessa maneira, o nó pai é deslocado para a posição à direita do nó antecessor.

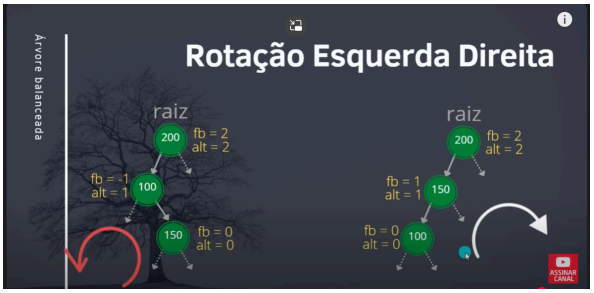


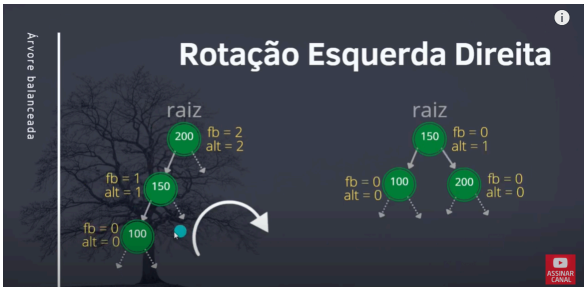
- **Rotação Direita Esquerda (Rotação Dupla Esquerda, ou Rotação Dupla Direita):** Este método é adotado quando a árvore apresenta desequilíbrio em ambos os lados, não apenas em um deles. Para resolver este problema, é preciso fazer uma rotação primeiro à direita, e depois uma rotação à esquerda.





- **Rotação Esquerda Direita:** Esta rotação é aplicada quando a árvore também está desbalanceada em dois sentidos diferentes. Entretanto, para restabelecer o equilíbrio é necessário fazer uma rotação à esquerda, e depois outra rotação à direita.





---

2. Estrutura do Projeto

A estrutura possui duas classes principais, que são responsáveis por descrever os dois principais elementos da árvore AVL: o Nó e a Árvore. Além de possuir o arquivo Program.cs, que vai ser o ponto inicial para a construção desta estrutura, e conter um menu interativo com o usuário. Este projeto vai importar os dados de entrada a partir de um arquivo externo.

2.1 Classe No.cs

Neste arquivo vai ser definido uma classe chamada No, que irá ser composta por:

- 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

2.2 Classe Arvore.cs

- Inserção e remoção de Nós na árvore
- Faz o cálculo e a verificação do balanceamento
- Executa as rotações necessárias para o equilíbrio da árvore
- Tem o método para imprimir a árvore

2.1 Classe No.cs

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:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ArvoreAVL
{
    public class No
    {
        public int Valor;
        public int Altura;
        public No? Esquerda;
        public No? Direita;

         public No(int valor)
        {
            Valor = valor;
            Altura = 1;
            Esquerda = null;
            Direita = null;
        }
    }
}
```

2.2 Classe Arvore.cs

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. A classe Arvore vai ser composta pelos seguintes métodos:

- 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: Corresponde a um método do tipo bool, em que retorna 1 caso encontrou o nó que teve seu valor passado como parâmetro. Ou 0, se caso não existir.

- 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.

```
using System;

namespace ArvoreAVL
{
    public class Arvore
```

```
{
    private No? Raiz;

    public void Inserir(int valor)
    {
        Raiz = Inserir(Raiz, valor);
    }

    private No Inserir(No? no, int valor)
    {
        if (no == null)
            return new No(valor);
        if (valor < no.Valor)
            no.Esquerda = Inserir(no.Esquerda, valor);
        else if (valor > no.Valor)
            no.Direita = Inserir(no.Direita, valor);
        else
            return no;

        AtualizarAltura(no);
        return Balancear(no);
    }

    public void Remover(int valor)
    {
        Raiz = Remover(Raiz, valor);
    }

    private No? Remover(No? no, int valor)
    {
        if (no == null)
            return null;
        if (valor < no.Valor)
```

```
            no.Esquerda = Remover(no.Esquerda, valor);
        else if (valor > no.Valor)
            no.Direita = Remover(no.Direita, valor);
        else
        {
            if (no.Esquerda == null || no.Direita == null)
                return no.Esquerda ?? no.Direita;

            No sucessor = EncontrarMenor(no.Direita);
            no.Valor = sucessor.Valor;
            no.Direita = Remover(no.Direita, sucessor.Valor);
        }
        AtualizarAltura(no);
        return Balancear(no);
    }

    private No EncontrarMenor(No? no)
    {
        while (no!.Esquerda != null)
            no = no.Esquerda;
        return no;
    }

    public bool Buscar(int valor)
    {
        return Buscar(Raiz, valor) != null;
    }

    private No? Buscar(No? no, int valor)
    {
        if (no == null)
            return null;
        if (valor == no.Valor)
```

```
            return no;
        else if (valor < no.Valor)
            return Buscar(no.Esquerda, valor);
        else
            return Buscar(no.Direita, valor);
    }

    public bool Existe(int valor)
    {
        return Buscar(valor);
    }

    public void ImprimirInOrder()
    {
        ImprimirInOrder(Raiz);
        Console.WriteLine();
    }

    private void ImprimirInOrder(No? no)
    {
        if (no != null)
        {
            ImprimirInOrder(no.Esquerda);
            Console.Write($"{no.Valor} ");
            ImprimirInOrder(no.Direita);
        }
    }

    public void ImprimirPreOrdem()
    {
        ImprimirPreOrdem(Raiz);
        Console.WriteLine();
    }
```

```
private void ImprimirPreOrdem(No? no)
{
    if (no == null) return;

    Console.Write(no.Valor + " ");
    ImprimirPreOrdem(no.Esquerda);
    ImprimirPreOrdem(no.Direita);
}

public void ImprimirPosOrdem()
{
    ImprimirPosOrdem(Raiz);
    Console.WriteLine();
}

private void ImprimirPosOrdem(No? no)
{
    if (no == null) return;

    ImprimirPosOrdem(no.Esquerda);
    ImprimirPosOrdem(no.Direita);
    Console.Write(no.Valor + " ");
}

public void VerificarBalanceamento()
{
    VerificarBalanceamento(Raiz);
}

private void VerificarBalanceamento(No? no)
{
    if (no != null)
```

```
            {
                VerificarBalanceamento(no.Esquerda);
                int fb = FatorBalanceamento(no);
                Console.WriteLine($"Nó {no.Valor} - FB = {fb}");
                VerificarBalanceamento(no.Direita);
            }
        }

        public void ImprimirArvoreVisual()
        {
            ImprimirArvoreVisual(Raiz, " ", true);
        }

        private void ImprimirArvoreVisual(No? no, string prefixo,
bool ehFilhoDireito)
        {
            // Prefixo controla os espaços/linhas verticais
            // ehFilhoDireito define se o nó atual está à direita
(controla └── ou ├──)
            if (no != null)
            {
                Console.WriteLine(prefixo + (ehFilhoDireito ? "└──
" : "├── ") + no.Valor);
                ImprimirArvoreVisual(no.Direita, prefixo +
(ehFilhoDireito ? "    " : "│   "), false);
                ImprimirArvoreVisual(no.Esquerda, prefixo +
(ehFilhoDireito ? "    " : "│   "), true);
            }
        }
        private int Altura(No? no) => no?.Altura ?? 0;

        private void AtualizarAltura(No no)
        {
```

```
            no.Altura = 1 + Math.Max(Altura(no.Esquerda),
Altura(no.Direita));
        }

        private int FatorBalanceamento(No no)
        {
            return Altura(no.Esquerda) - Altura(no.Direita);
        }

        private No Balancear(No no)
        {
            int fb = FatorBalanceamento(no);

            if (fb > 1)
            {
                if (FatorBalanceamento(no.Esquerda!) < 0)
                    no.Esquerda = RotacionarEsquerda(no.Esquerda!);
                return RotacionarDireita(no);
            }

            if (fb < -1)
            {
                if (FatorBalanceamento(no.Direita!) > 0)
                    no.Direita = RotacionarDireita(no.Direita!);
                return RotacionarEsquerda(no);
            }

            return no;
        }
        private No RotacionarDireita(No y)
        {
            No x = y.Esquerda!;
            No T2 = x.Direita!;
```

```
            x.Direita = y;
            y.Esquerda = T2;

            AtualizarAltura(y);
            AtualizarAltura(x);

            return x;
        }

        private No RotacionarEsquerda(No x)
        {
            No y = x.Direita!;
            No T2 = y.Esquerda!;

            y.Esquerda = x;
            x.Direita = T2;

            AtualizarAltura(x);
            AtualizarAltura(y);

            return y;
        }
    }
}
```

2.3 Program.cs

É neste arquivo que toda a estrutura da árvore AVL será construída, porque é nele que os métodos irão ser chamados. Ela vai conter a lógica principal de interação com o usuário, e é a partir dela que será possível manipular e realizar diferentes operações na árvore.

- Contém um método principal que vai ser responsável por inicializar a árvore. Além de carregar o arquivo dados.txt, que contém os valores que deseja que sejam inseridos na árvore.
- É uma lógica Do/ While, em que vai ser apresentado um menu interativo para que o usuário possa escolher a opção que deseja para manipular a árvore. E se caso ele digitar 0, o programa será encerrado.
- As opções apresentadas no menu:
    1. Inserir Valor
    2. Remover Valor
    3. Buscar Valor
    4. Imprimir InOrdem
    5. Imprimir PosOrdem
    6. Imprimir PreOrdem
    7. Verificar Balanceamento
    8. Ver árvore visualmente
    9. Sair

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.

```
using System;
namespace ArvoreAVL
{
    class Program
    {
        static void Main(string[] args)
        {
            Arvore arvore = new Arvore();

            if (File.Exists("dados.txt"))
            {
                string[] linhas = File.ReadAllLines("dados.txt");
                foreach (var linha in linhas)
                {
                    if (int.TryParse(linha, out int valor))
                    {
                        arvore.Inserir(valor);
                    }
                    else
                    {
                        Console.WriteLine("Arquivo 'dados.txt' não
encontrado. Iniciando árvore vazia.");
                    }
                }
            }

            int opcao = -1;
            while (opcao != 0)
            {
                Console.WriteLine("\n--- MENU ---");
                Console.WriteLine("1 - Inserir valor");
                Console.WriteLine("2 - Remover valor");
                Console.WriteLine("3 - Buscar valor");
```

```
                Console.WriteLine("4 - Imprimir InOrder");
                Console.WriteLine("5 - Imprimir PosOrdem");
                Console.WriteLine("6 - Imprimir PreOrdem");
                Console.WriteLine("7 - Verificar balanceamento");
                Console.WriteLine("8 - Ver árvore visualmente");
                Console.WriteLine("0 - Sair");
                Console.Write("Escolha uma opção: ");

                string? entrada = Console.ReadLine();
                if (!int.TryParse(entrada, out opcao))
                {
                    Console.WriteLine("Entrada inválida.");
                    continue;
                }

                switch (opcao)
                {
                    case 1:
                        Console.Write("Digite o valor a inserir: ");
                        if (int.TryParse(Console.ReadLine(), out int
valorInserir))
                        {
                            if (arvore.Existe(valorInserir))
                            {
                                Console.WriteLine("Valor já existe
na árvore!");
                            }
                            else
                            {
                                arvore.Inserir(valorInserir);
                                Console.WriteLine("Valor
inserido.");
                            }
```

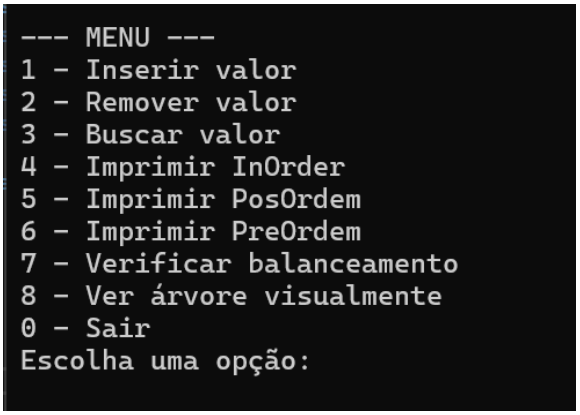
```
                        }
                        else
                        {
                            Console.WriteLine("Valor inválido.");
                        }
                        break;

                    case 2:
                        Console.Write("Digite o valor a remover: ");
                        if (int.TryParse(Console.ReadLine(), out int
valorRemover))
                        {
                            arvore.Remover(valorRemover);
                            Console.WriteLine("Valor removido (se
existir).");
                        }
                        else
                        {
                            Console.WriteLine("Valor inválido.");
                        }
                        break;

                    case 3:
                        Console.Write("Digite o valor a buscar: ");
                        if (int.TryParse(Console.ReadLine(), out int
valorBuscar))
                        {
                            bool encontrado =
arvore.Buscar(valorBuscar);
                            Console.WriteLine(encontrado ? "Valor
encontrado!" : "Valor não encontrado.");
                        }
                        else
```

```
                    {
                        Console.WriteLine("Valor inválido.");
                    }
                    break;

                case 4:
                    Console.WriteLine("Impressão InOrder:");
                    arvore.ImprimirInOrder();
                    break;

                case 5:
                    Console.WriteLine("Impressão Pós-Ordem:");
                    arvore.ImprimirPosOrdem();
                    break;

                case 6:
                    Console.WriteLine("Impressão Pré-Ordem:");
                    arvore.ImprimirPreOrdem();
                    break;

                case 7:
                    Console.WriteLine("Verificando
balanceamento...");
                    arvore.VerificarBalanceamento();
                    break;

                case 8:
                    Console.WriteLine("Visual da árvore:");
                    arvore.ImprimirArvoreVisual();
                    break;

                case 0:
                    Console.WriteLine("Encerrando o programa.");
```

```
                    break;

                default:
                    Console.WriteLine("Opção inválida.");
                    break;
            }
        }
    }
}
```

---

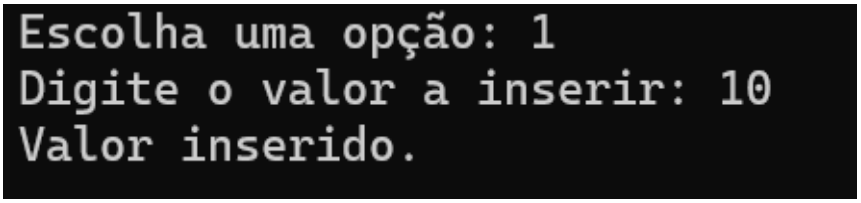
3. Funcionamento do Menu



O programa principal fornece um menu de opções, para que desta forma o usuário seja capaz de realizar diversas operações na árvore. Essa interface facilita a manipulação e a visualização dessa estrutura de dados.

Opções do Menu:

1. **Inserir Valor:** Esta opção permite a inserção de um novo valor na Árvore AVL, e se caso necessário, o programa já faz a rotação, se caso estiver desbalanceada.



2. **Remover Valor:** um valor específico da Árvore AVL será removido, e ela pode passar por um processo de balanceamento se precisar.

```
Escolha uma opção: 2
Digite o valor a remover: 10
Valor removido (se existir).
```

3. **Buscar Valor:** Vai buscar o valor informado pelo usuário na árvore, e imprimir se ele foi encontrado ou se não existe.

```
Escolha uma opção: 3
Digite o valor a buscar: 40
Valor encontrado!
```

4. **Imprimir InOrder:** Vai imprimir os valores da árvore seguindo a ordem: esquerda → raiz → direita.

```
Escolha uma opção: 4
Impressão InOrder:
23 40
```

5. **Imprimir PosOrdem:** Os valores contidos na árvore serão exibidos na ordem: esquerda → direita → raiz.

```
Escolha uma opção: 5
Impressão Pós-Ordem:
40 23
```

6. **Imprimir PreOrdem:** Nesta opção os valores irão ser impressos na seguinte sequência: raiz → esquerda → direita.

```
Escolha uma opção: 6
Impressão Pré-Ordem:
23 40
```

7. **Verificar Balanceamento:** Exibe o fator de balanceamento de cada nó da árvore, para que o usuário consiga verificar se ela está corretamente balanceada.

```
Escolha uma opção: 7
Verificando balanceamento...
Nó 23 - FB = -1
Nó 40 - FB = 0
```

8. **Ver árvore visualmente:** Exibe em um formato mais visual de como é a árvore, ela já vai estar balanceada nesta visualização.

```
Escolha uma opção: 8
Visual da árvore:
 └── 23
     ├── 40
```

9. **Sair:** Encerra a execução do programa

```
Escolha uma opção: 0
Encerrando o programa.
```

---

4. Exemplo de Entrada- dados.txt

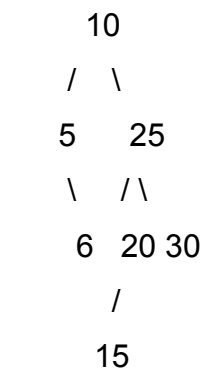
Os dados utilizados nos testes de verificação do funcionamento da árvore foram carregados por um arquivo txt. Cada valor refere-se a um nó da árvore, que é inserido sequencialmente na estrutura. Durante a inserção, a árvore AVL realiza automaticamente as rotações necessárias, simples ou duplas, para desta forma garantir que a propriedade de balanceamento seja mantida. Foi carregado o seguinte arquivo:

dados.txt

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

25

Levando em consideração os dados fornecidos, a Árvore AVL teria a seguinte configuração:



Esta estrutura representa uma árvore AVL balanceada, onde os valores estão posicionados corretamente na árvore. E por mais que ocorra posteriormente uma inserção ou remoção de valor, o programa irá estruturar ela novamente realizando a rotação necessária.

Ao iniciar o programa, este arquivo vai ser carregado automaticamente, e permite que o usuário manipule os valores da árvore através do menu.

---

5. Funcionalidades Implementadas

5.1. Leitura dos Dados

Quando o programa é iniciado ele vai realizar a leitura dos dados dentro de um arquivo txt, que chama dados.txt. Se caso ele não existir, a árvore vai ser iniciada mas com valores vazios. Este arquivo contém vários números inteiros, que nada mais são que os valores dos nós que irão compor a árvore.

5.2. Menu Interativo no Console

O programa irá exibir um menu em que permite que o usuário manipule a árvore AVL. Oferece as seguintes opções:

- Inserir valor
- Remover valor
- Buscar valor
- Imprimir InOrder
- Imprimir Pós-Ordem
- Imprimir Pré-Ordem

- Verificar balanceamento
- Ver árvore visualmente
- Sair

5.3. Estrutura da Árvore AVL

As funcionalidades foram implementadas através de três arquivos, em que cada um é responsável por vários métodos que precisam ser executados ao longo do programa. Abaixo segue cada arquivo, com a informação de quais métodos que ele implementa.

I) No.cs

- Armazena um valor inteiro
- Altura do Nó
- Tem os nós filhos, que estão à esquerda e a direita.

II) Arvore.cs

- Tem um método para inserir um nó (valor)
- Tem um método responsável por remover um nó (valor)
- Tem um método que irá buscar um nó (valor)
- Imprime a árvore de três formas diferentes: InOrder, PreOrdem e PosOrdem
- Verificar o balanceamento
- Imprime a árvore de uma maneira mais visual
- Realiza o balanceamento

III) Program.cs

- Carrega as informações contidas no arquivo dados.txt
- Inicializa a árvore
- Exibe o menu para o usuário
- Lê os dados informados pelo usuário, e chama os métodos correspondentes

---

6. Observação

Precisa melhorar a escrita depois: A Carol me falou que o nosso programa não lê as letras, apenas os números. Isto acontece porque se ele lesse as letras, não íamos conseguir acrescentar valores na árvore.

7. Resultados

Depois da execução do projeto, foi obtido os seguintes resultados:

- É possível visualizar todos os nós e valores carregados pelo arquivo txt.
- O usuário pode inserir ou remover valores da árvore sem que ela fique desbalanceada, já que o programa balanceia ela automaticamente.
- Permite a visualização da árvore em várias ordens diferentes, além de uma maneira mais visual.
- Pode-se analisar o balanceamento de cada nó da árvore.

---

8. Conclusão

Este projeto ilustrou a importância de uma estrutura de dados balanceada, pois ela é capaz de garantir uma boa eficiência em operações de inserção, remoção e busca de valores em uma árvore.

A árvore consegue manter um bom desempenho mesmo trabalhando com um grande volume de dados, porque ela calcula o fator de balanceamento de cada nó. E se caso for necessário, realiza as devidas rotações para manter o equilíbrio dela.

O menu desenvolvido permite que o usuário tenha a experiência de manipular a árvore, e ver ela organizada de várias formas diferentes. Isto ilustra a integração do desenvolvimento técnico e o prático**.**

---

9. Referências