# Prática 1 - AEDS II

Bruno Prado Dos Santos
bruno.santos@aluno.cefetmg.br

Centro Federal de Educação Tecnológica de Minas Gerais - Campus V
Engenharia da Computação

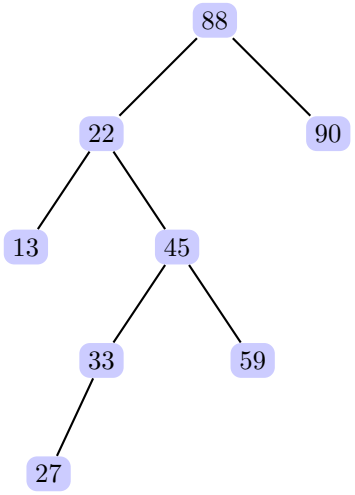
Professor: Michel Pires Dias

November 25, 2024

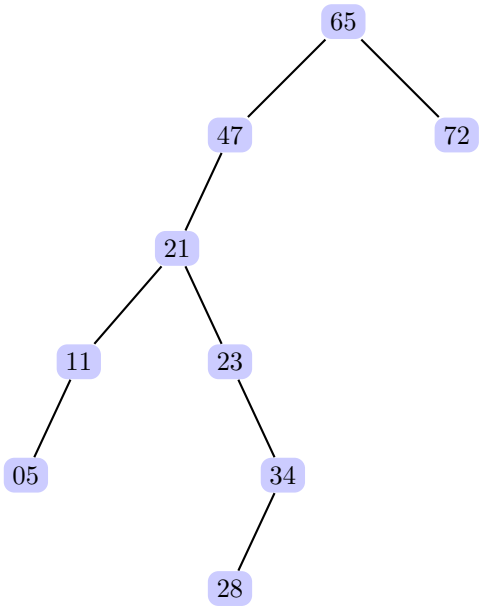
## 1 PROBLEMA 1

### 1.1 Construa as árvores binárias de busca a partir dos conjuntos abaixo, e desenhe a estrutura da árvore após cada inserção de k elementos.

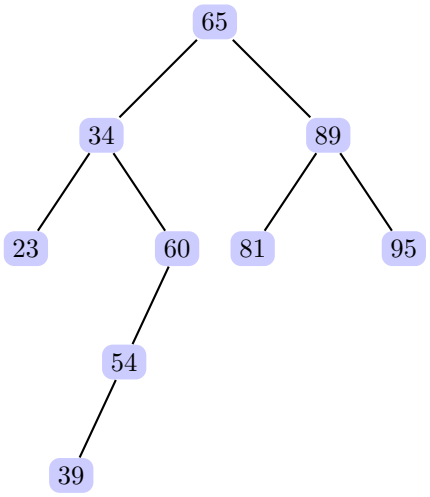
#### 1.1.1 Árvore 1: [88, 22, 45, 33, 22, 90, 27, 59, 13]



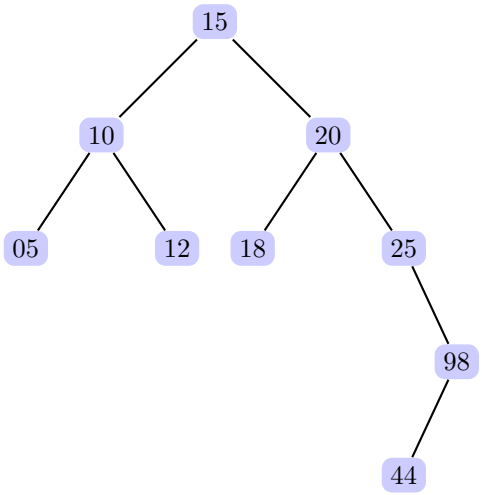
### 1.1.2 Árvore 2: [65, 47, 21, 11, 72, 23, 05, 34, 28]




### 1.1.3 Árvore 3: [65, 34, 89, 23, 60, 54, 81, 95, 39]



#### 1.1.4 Árvore 4: [15, 10, 20, 05, 12, 18, 25, 98, 44]

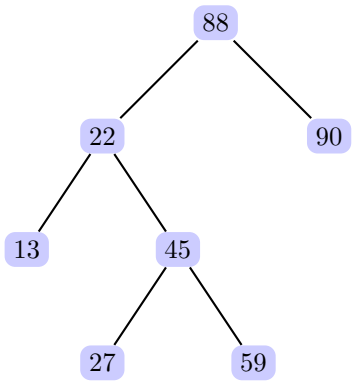



### 1.2 Realize a remoção dos elementos a seguir, redesenhando a árvore após cada remoção. Para cada remoção, discuta o impacto estrutural na árvore, abordando os diferentes casos de remoção (remoção de folha, nó com um filho e nó com dois filhos). Além disso, ao remover os nós com dois filhos, determine e justifique a escolha entre o sucessor in-ordem ou o predecessor in-ordem.

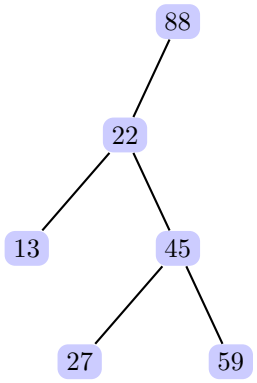
#### 1.2.1 Árvore 1: [33, 90, 33, 45]

Utilizei o método do predecessor:

**[33]:**



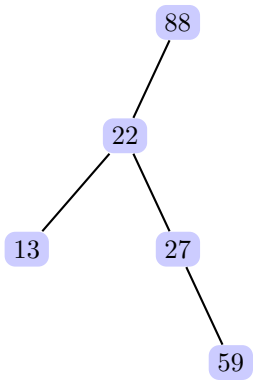

**[90]:**




**[33]:**

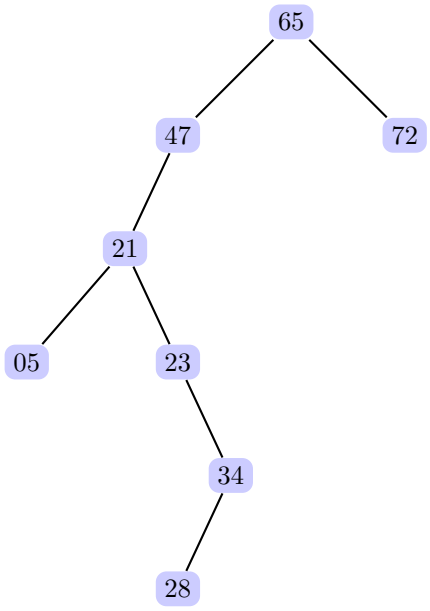
Não acontece nada pois não achará o número ao procurar.

**[45]:**

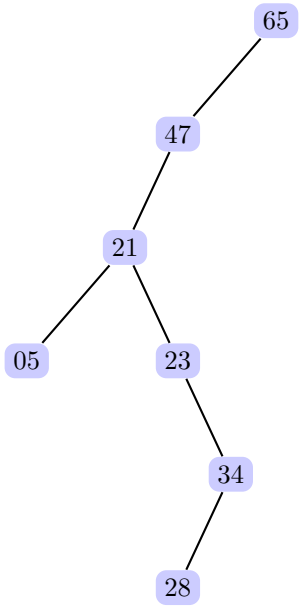



#### 1.2.2 Árvore 2: [11, 72, 65, 23]

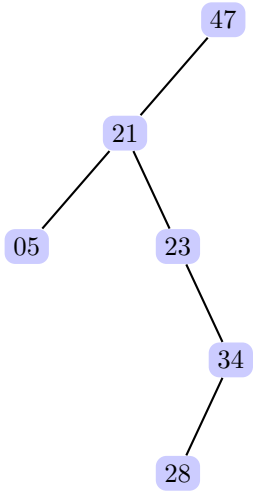
**[11]:**



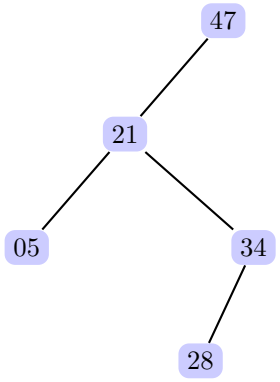
**[72]:**




**[65]:**



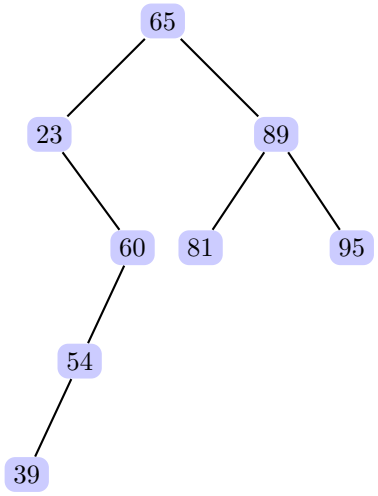
**[23]:**



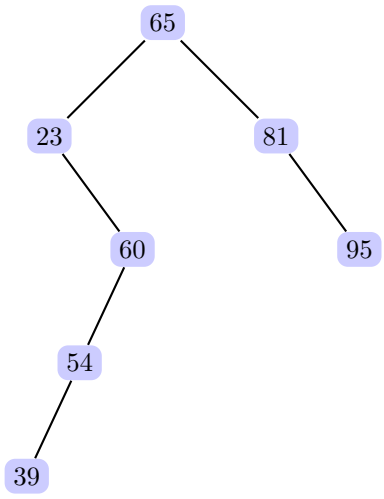

### 1.2.3 Árvore 3: [34, 89, 81, 95]

Utilizei o método do predecessor:

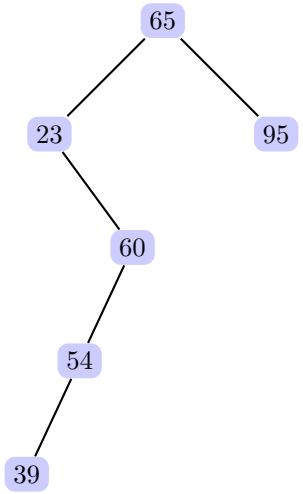
**[34]:**



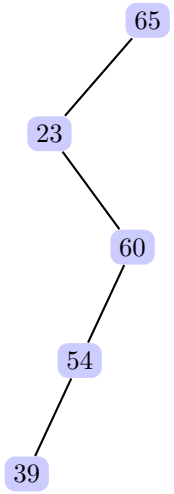

**[89]:**



**[81]:**



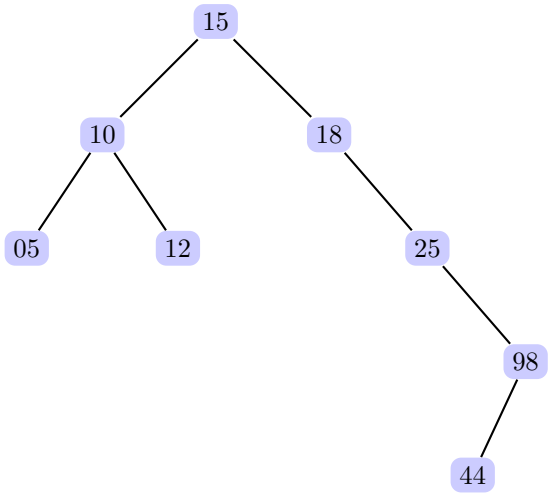

**[95]:**



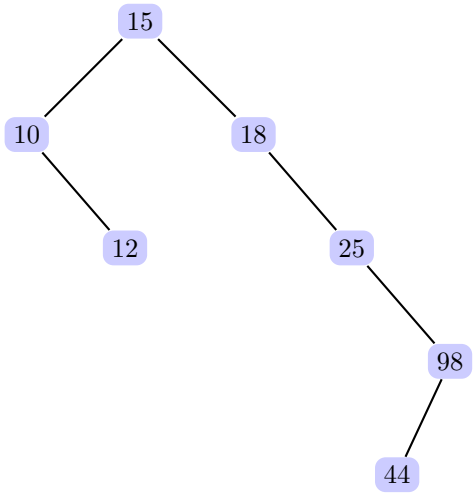
### 1.2.4 Árvore 4: [20, 05, 18, 44]

Utilizei o método do predecessor:

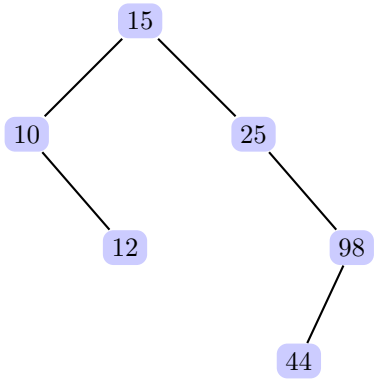
**[20]:**



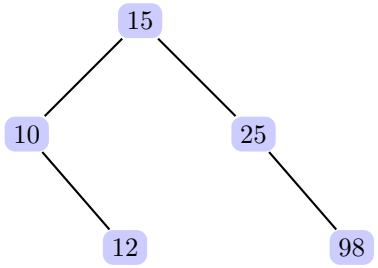

**[05]:**




**[18]:**



**[44]:**

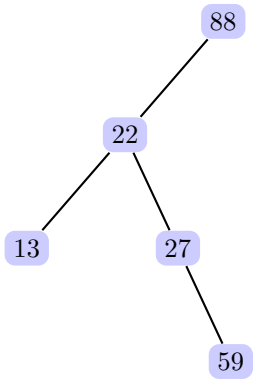



### 1.3 Após realizar todas as inserções e remoções, selecione um elemento específico em cada uma das árvores e utilize os quatro tipos de caminhamento apresentados em sala de aula como métodos de pesquisa para localizar o elemento escolhido. Para cada tipo de caminhamento, determine:

**• O número de interações necessárias com a estrutura da árvore para encontrar o elemento selecionado.**
**• A ordem de visitação dos nós até localizar o elemento, destacando o caminho percorrido.**
**• A eficiência de cada estratégia ao decorrer do processamento necessário para identificar o elemento desejado.**

#### 1.3.1 Árvore 1:

Procurar o elemento [59]:




1. **Pré-Ordem (Raiz → Esquerda → Direita)**
   - Número de interações: 7
   - Caminho percorrido: 88 → 22 → 27 → 59
   - Ordem de visitação (completa): 88 → 22 → 13 → 27 → 59
   - Eficiência: Melhor, pois o 59 foi encontrado após percorrer 7 nós

2. **In-ordem (Esquerda → Raiz → Direita)**
   - Número de interações: 8
   - Caminho percorrido: 13 → 22 → 27 → 59
   - Ordem de visitação (completa): 13 → 22 → 27 → 59 → 88
   - Eficiência: Média, pois o 59 foi encontrado após percorrer 8 nós

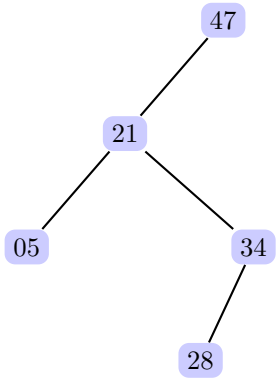
3. **Pós-ordem (Esquerda → Direita → Raiz)** - Número de interações: 9
   - Caminho percorrido: 13 → 59

- Ordem de visitação (completa): 13 → 59 → 27 → 22 → 88
- Eficiência: Média, pois o 59 foi encontrado após percorrer 9 nós

4. **Em nível (Nível por nível)** - Número de interações: 10
   - Caminho percorrido: 88 → 22 → 13 → 27 → 59
   - Ordem de visitação (completa): 88 → 22 → 13 → 27 → 59
   - Eficiência: Pior, pois o 59 foi encontrado após percorrer 10 nós

#### 1.3.2 Árvore 2:

Procurar o elemento [05]:




1. **Pré-Ordem (Raiz → Esquerda → Direita)**
   - Número de interações: 2
   - Caminho percorrido: 47 → 21 → 05
   - Ordem de visitação (completa): 47 → 21 → 05 → 34 → 28
   - Eficiência: Melhor, pois o 05 foi encontrado após percorrer 2 nós

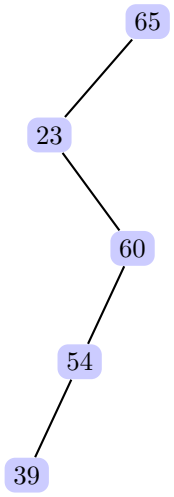
2. **In-ordem (Esquerda → Raiz → Direita)**
   - Número de interações: 3
   - Caminho percorrido: 05
   - Ordem de visitação (completa): 05 → 21 → 28 → 34 → 47
   - Eficiência: Média, pois o 05 foi encontrado após percorrer 3 nós

3. **Pós-ordem (Esquerda → Direita → Raiz)** - Número de interações: 4
   - Caminho percorrido: 05
   - Ordem de visitação (completa): 05 → 28 → 34 → 21 → 47
   - Eficiência: Média, pois o 05 foi encontrado após percorrer 4 nós

4. **Em nível (Nível por nível)** - Número de interações: 6
   - Caminho percorrido: 47 → 21 → 05
   - Ordem de visitação (completa): 47 → 21 → 05 → 34 → 28
   - Eficiência: Pior, pois o 05 foi encontrado após percorrer 6 nós

#### 1.3.3 Árvore 3:

Procurar o elemento [60]:




1. **Pré-Ordem (Raiz → Esquerda → Direita)**
   - Número de interações: 3
   - Caminho percorrido: 65 → 23 → 60
   - Ordem de visitação (completa): 65 → 23 → 60 → 54 → 39
   - Eficiência: Melhor, pois o 60 foi encontrado após percorrer 3 nós

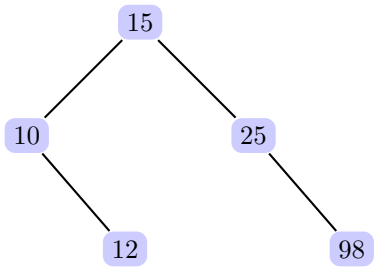
2. **In-ordem (Esquerda → Raiz → Direita)**
   - Número de interações: 8
   - Caminho percorrido: 23 → 39 → 54 → 60
   - Ordem de visitação (completa): 23 → 39 → 54 → 60 → 65
   - Eficiência: Média, pois o 60 foi encontrado após percorrer 8 nós

3. **Pós-ordem (Esquerda → Direita → Raiz)** - Número de interações: 9
   - Caminho percorrido: 39 → 54 → 60
   - Ordem de visitação (completa): 39 → 54 → 60 → 23 → 65
   - Eficiência: Média, pois o 60 foi encontrado após percorrer 9 nós

4. **Em nível (Nível por nível)** - Número de interações: 6
   - Caminho percorrido: 65 → 23 → 60
   - Ordem de visitação (completa): 65 → 23 → 60 → 54 → 39
   - Eficiência: Pior, pois o 60 foi encontrado após percorrer 6 nós

#### 1.3.4 Árvore 4:

Procurar o elemento [12]:



1. **Pré-Ordem (Raiz → Esquerda → Direita)**
   - Número de interações: 3
   - Caminho percorrido: 15 → 10 → 12
   - Ordem de visitação (completa): 15 → 10 → 12 → 25 → 98
   - Eficiência: Melhor, pois o 12 foi encontrado após percorrer 3 nós

2. **In-ordem (Esquerda → Raiz → Direita)**
   - Número de interações: 4
   - Caminho percorrido: 10 → 12
   - Ordem de visitação (completa): 10 → 12 → 15 → 25 → 98
   - Eficiência: Média, pois o 12 foi encontrado após percorrer 4 nós

3. **Pós-ordem (Esquerda → Direita → Raiz)** - Número de interações: 5
   - Caminho percorrido: 39 → 54 → 60
   - Ordem de visitação (completa): 12 → 10 → 98 → 25 → 15
   - Eficiência: Média, pois o 12 foi encontrado após percorrer 5 nós

4. **Em nível (Nível por nível)** - Número de interações: 8
   - Caminho percorrido: 15 → 10 → 25 → 12
   - Ordem de visitação (completa): 15 → 10 → 25 → 12 → 98
   - Eficiência: Pior, pois o 12 foi encontrado após percorrer 8 nós

# 2 PROBLEMA 2

Em árvores binárias, o nível máximo é frequentemente utilizado para compreender a profundidade da estrutura e o tempo necessário para percorrer a árvore em diferentes operações. O nível máximo, também chamado de altura da árvore, é definido como a distância (em termos de número de elementos ou nós) da raiz até a folha mais distante.

Neste exercício, você deverá elaborar uma função que não apenas calcule o nível máximo da árvore, mas que também apresente os seguintes desafios:

1. **Cálculo do Nível Máximo:** Implemente uma função que calcule o nível máximo de uma árvore binária sem balanceamento. A função deve percorrer toda a estrutura e identificar o nível da folha mais distante da raiz, retornando esse valor ao usuário.

2. **Visualização Interativa:** A cada nova inserção ou remoção de um nó, atualize e exiba o nível máximo da árvore, permitindo que o usuário visualize como a profundidade da árvore é impactada pela desbalanceamento natural da estrutura.

3. **Análise de Crescimento:** Considere dois conjuntos de inserções, um que gere uma árvore "torta" (mais desbalanceada) e outro que resulte em uma árvore mais equilibrada (embora sem ser balanceada automaticamente). Calcule e compare os níveis máximos dessas duas árvores ao longo de cada inserção, explicando por que certas inserções resultam em maiores níveis do que outras. Além disso, tente observar, se possível, se a prerrogativa de custo de 39% de depreciação de fato ocorre em uma árvore não balanceada em comparação com aquela que se mostra mais organizada.

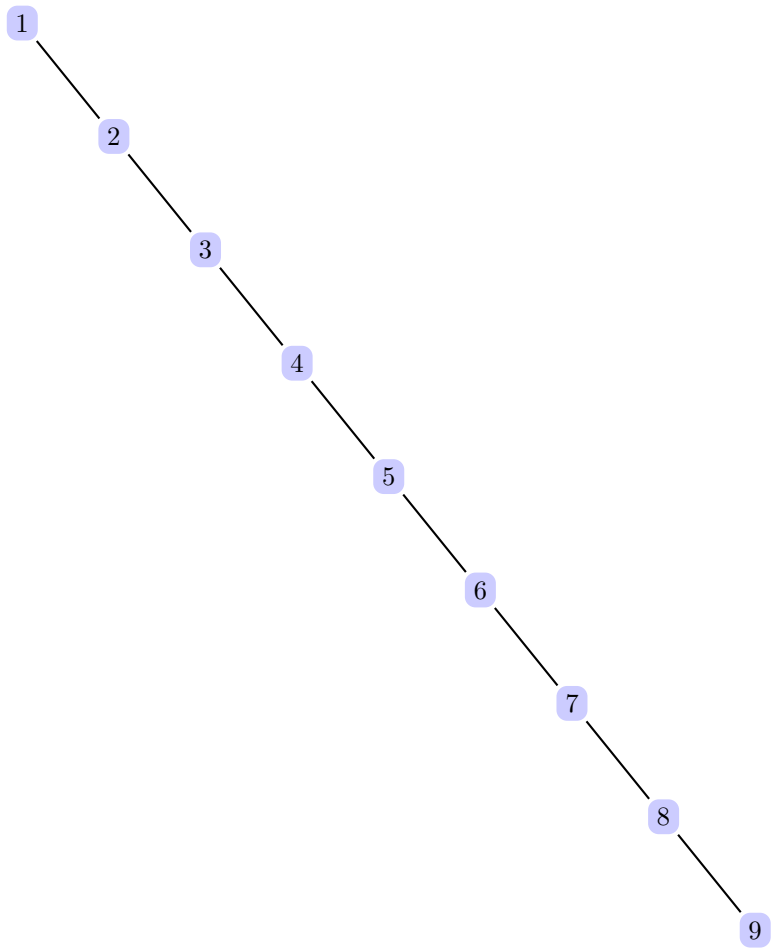
4. **Caminho mais longo:** Após calcular o nível máximo da árvore, identifique e mostre ao usuário o caminho completo da raiz até a folha que define esse nível. Discuta como o desbalanceamento da árvore afeta o comprimento desse caminho em comparação com uma árvore idealmente balanceada.

**RESPOSTA**

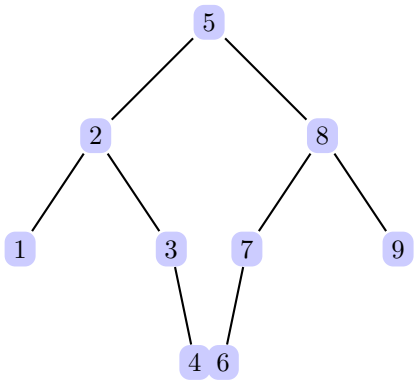
O código para o Problema 2 está no GitHub. https://github.com/ByBrun0/Pratica1-AEDS-II.git

## 2.1 Análise de Crescimento:

- Árvore Desbalanceada [1, 2, 3, 4, 5, 6, 7, 8, 9]:




- Árvore Balanceada [5, 2, 8, 1, 9, 3, 7, 4, 6]:



| **Etapa de Inserção** | **Altura da Árvore Desbalanceada** | **Altura da Árvore Balanceada** |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1 | 1 |
| 3 | 2 | 1 |
| 4 | 3 | 2 |
| 5 | 4 | 2 |
| 6 | 5 | 2 |
| 7 | 6 | 2 |
| 8 | 7 | 3 |
| 9 | 8 | 3 |

Table 1: Comparação da altura entre árvores desbalanceadas e balanceadas em diferentes etapas de inserção.

**Conclusão:** Percebe-se que ao inserir de maneira mais equilibrada a altura tende a não se alterar tanto, ela demora mais tempo para aumentar, o que é o contrario do resultado apresentado por uma arvore desbalanceada.

# 3 PROBLEMA 3

Imagine que você está desenvolvendo um dicionário eletrônico que permite aos usuários pesquisar rapidamente definições de palavras em um idioma específico. O desafio é projetar uma estrutura de dados eficiente, que permita buscas rápidas e economize espaço de armazenamento. Nesse contexto, elabore uma solução baseada em uma árvore binária de busca, com as seguintes características e requisitos adicionais:

- **Autocompletar e Sugestões Inteligentes:** Implemente recursos de autocompletar que, conforme o usuário digita as primeiras letras de uma palavra, sugira automaticamente termos correspondentes. A estrutura da árvore deve ser otimizada para permitir buscas rápidas e dinâmicas, retornando sugestões em tempo real. Discuta a eficiência do autocompletar utilizando a árvore binária e apresente uma análise comparativa em termos de tempo de busca para diferentes tamanhos de dicionário.
- **Desempenho e Otimizações:** Embora a árvore binária de busca ofereça vantagens de eficiência, ela pode se tornar desbalanceada à medida que mais palavras são inseridas. Discuta técnicas de otimização, como a utilização de árvores balanceadas (ex.: AVL, Red-Black) para garantir que a estrutura mantenha sua eficiência, mesmo com grandes volumes de dados. Proponha métodos de compactação ou armazenamento que minimizem o uso de memória sem comprometer o desempenho.

Além disso, elabore um conjunto de testes que simulem o uso do dicionário, com inserções e buscas de palavras, e avalie o tempo de resposta para diferentes volumes de dados. Utilize essas métricas para justificar as escolhas feitas na estrutura e nas otimizações aplicadas.

**RESPOSTA**

Resolvido no GitHub

# References


[1] SILVA, Michel Pires da. *Aula 1: Introdução a algoritmos de pesquisa e árvores binárias*. CEFET-MG, Campus Divinópolis, 2024. Disponível no Moodle: https://ava.cefetmg.br. Acesso em: 25 out. 2024.

[2] SILVA, Michel Pires da. *Aula 2: Estruturas AVL e balanceamento*. CEFET-MG, Campus Divinópolis, 2024. Disponível no Moodle: https://ava.cefetmg.br. Acesso em: 28 out. 2024.

[3] SILVA, Michel Pires da. *Aula 3: Árvores vermelho e preta (Red-Black)*. CEFET-MG, Campus Divinópolis, 2024. Disponível no Moodle: https://ava.cefetmg.br. Acesso em: 12 nov. 2024.

[4] SILVA, Michel Pires da. *Aula 4: Estruturas B-Tree e B+-Tree*. CEFET-MG, Campus Divinópolis, 2024. Disponível no Moodle: https://ava.cefetmg.br. Acesso em: 19 nov. 2024.

[5] SANTOS, Bruno Prado dos. *Repositório GitHub para a implementação dos problemas 2 e 3*. Disponível em: https://github.com/ByBrun0/Pratica1-AEDS-II.git. Acesso em: 24 nov. 2024.