Universidade Presbiteriana Mackenzie

# INTRODUÇÃO A DATA SCIENCE USANDO ESTRUTURA DE DADOS

## Projeto II - Algoritmos

**DANIEL MANGABEIRA CORREIA**
**MESSIAS OLIVEIRA DA SILVA**

**SÃO PAULO**
**Outubro/2024**

# Sumário

## INTRODUÇÃO

Este trabalho tem como objetivo comparar o desempenho entre Árvores Binárias de Busca (BST) e Árvores AVL, utilizando uma base de dados de músicas com mais de 10 mil entradas, contendo informações como nome da faixa, artista, ano de lançamento, duração, popularidade e gênero. O foco da análise está na eficiência dessas estruturas de dados em operações de inserção e busca, além de avaliar suas alturas e tempos de execução.

Para isso, foram implementadas rotinas que leem o arquivo CSV, inserem as informações nas árvores e realizam buscas utilizando diferentes chaves. A comparação entre as árvores é feita por meio da criação de gráficos e tabelas, que apresentam os resultados de desempenho de cada uma em diferentes tamanhos de subconjuntos de dados.

Além disso, este trabalho discute a influência da escolha do atributo chave nas operações das árvores e como essa escolha afeta a construção e o desempenho das estruturas. O resultado final visa não apenas ilustrar as vantagens e desvantagens de cada estrutura em termos de balanceamento e eficiência, mas também oferecer insights práticos sobre a escolha apropriada de árvores para diferentes tipos de dados e cenários de uso.

## OBJETIVO GERAL

Comparar o desempenho entre Árvores Binárias de Busca (BST) e Árvores AVL na manipulação de uma base de dados de músicas, avaliando sua eficiência em termos de inserção, busca e balanceamento, e analisando como a escolha do atributo chave influencia na construção e no desempenho das árvores.

## OBJETIVOS ESPECÍFICOS

1. **Ler e processar** uma base de dados de músicas com mais de 10 mil registros, organizando os dados em estruturas adequadas para inserção nas árvores.
2. **Implementar** as operações de inserção e busca tanto na Árvore Binária de Busca (BST) quanto na Árvore AVL, garantindo a correta manipulação e balanceamento das estruturas.
3. **Avaliar o desempenho** de cada árvore em termos de tempo de inserção, altura e tempo de busca, utilizando subconjuntos de diferentes tamanhos para testar escalabilidade.
4. **Gerar gráficos e tabelas** que apresentem os resultados comparativos de desempenho entre as árvores, permitindo uma análise visual e quantitativa das diferenças entre elas.
5. **Discutir a influência** da escolha de diferentes atributos como chave para as operações nas árvores e como isso afeta a eficiência e o balanceamento das estruturas.
6. **Exportar os resultados** das operações de inserção e busca, gerando arquivos CSV com dados ordenados e relatórios detalhados com as análises de desempenho.

## MATERIAL E MÉTODOS

### EQUIPAMENTO UTILIZADO

Os experimentos foram conduzidos em um computador com as seguintes especificações:

- Processador: 13ª Geração Intel® Core™ i7-13650HX, 2.60 GHz
- Memória RAM: 16,0 GB (utilizável: 15,7 GB)
- Sistema Operacional: Sistema operacional de 64 bits, processador baseado em x64

Apesar do hardware de alto desempenho, o código Python foi executado dentro de um ambiente virtual (‘env’) configurado no Windows Subsystem for Linux (WSL), proporcionando um ambiente de execução que simula o comportamento de servidores Linux.

### MASSA DE DADOS

Foram utilizados dados de um arquivo CSV com 15.151 linhas, contendo informações de músicas com as seguintes colunas:

**`Track`, `Artist`, `Year`, `Duration`, `Time_Signature`, `Danceability`, `Energy`, `Key`, `Loudness`, `Mode`, `Speechiness`, `Acousticness`, `Instrumentalness`, `Liveness`, `Valence`, `Tempo`, `Popularity`, `Genre`**.

Esses dados foram organizados e processados para a construção de árvores binárias e AVL, permitindo a comparação de desempenho em diferentes tipos de operações de busca e inserção.

## ALGORITMOS DE ÁRVORES

As seguintes estruturas de dados e algoritmos foram implementados e testados:

- **Árvore Binária de Busca (BST)**: Estrutura em que cada nó possui dois filhos, e a inserção segue a regra em que os nós à esquerda são menores que o nó atual e à direita são maiores.
- **Árvore AVL**: Variante da árvore binária de busca, onde é mantido um fator de balanceamento para garantir que a árvore permaneça balanceada após inserções e exclusões, evitando uma profundidade muito grande e, consequentemente, degradando o desempenho das operações.

## METODOLOGIA DE EXECUÇÃO

1. **Leitura dos Dados**: O arquivo CSV foi lido utilizando a biblioteca `csv` do Python, transformando os dados em listas e objetos antes da inserção nas árvores.
2. **Inserção e Busca**: Os registros de músicas foram inseridos nas Árvores Binária de Busca (BST) e AVL, seguidos de buscas baseadas em atributos como Artist, Year e Popularity para avaliar o desempenho.
3. **Medição de Tempo**: O tempo de execução das operações de inserção e busca foi medido com a função `time()` da biblioteca `time`, registrando os resultados em milissegundos.
4. **Análise dos Resultados**: Os tempos de execução foram comparados entre as estruturas, gerando gráficos de desempenho para as operações de inserção e busca em dados ordenados e desordenados.
5. **Relatório e Armazenamento de Dados**: Os resultados e informações do sistema foram exportados para arquivos CSV e visualizações gráficas, facilitando a análise de desempenho de cada árvore.

## FERRAMENTAS E BIBLIOTECAS UTILIZADAS

- **Linguagem de Programação**: Python (versão 3.10.12)
- **Bibliotecas**:
    - **csv**: Biblioteca padrão para leitura e escrita de arquivos CSV, facilitando a manipulação de dados tabulares.
    - **matplotlib.pyplot**: Biblioteca para visualização de dados, utilizada para criar gráficos de desempenho das árvores binárias.
    - **time**: Biblioteca para manipulação do tempo, usada para medir a duração de operações como inserção e busca.
    - **random**: Biblioteca para geração de números aleatórios, utilizada para criar dados de teste.
    - **pandas**: Biblioteca para manipulação e análise de dados, oferecendo estruturas como DataFrames para facilitar o trabalho com dados tabulares.

## PROCEDIMENTO PARA EXECUÇÃO

1. **Preparação do Ambiente:**
   O ambiente de desenvolvimento foi configurado em Python, com a utilização de bibliotecas como **`csv`**, **`matplotlib.pyplot`**, **`time`**, **`random`** **`e pandas.`** O código foi executado em um ambiente Linux simulado via WSL (Windows Subsystem for Linux), garantindo a compatibilidade do script com sistemas operacionais baseados em Unix.
2. **Execução do Script:**
   O script foi executado para ler os dados do CSV, inserir as informações nas Árvores Binárias de Busca (BST) e AVL, e realizar buscas em ambas as estruturas. Foram utilizados métodos para calcular o tempo de inserção e busca, além da altura das árvores. A execução foi realizada para diferentes subconjuntos de dados de tamanhos variados.
3. **Geração de Gráficos e Relatórios:**
   Os tempos de inserção e busca, assim como as alturas das árvores, foram utilizados para gerar gráficos comparativos entre as BST e AVL. Os gráficos mostram o desempenho de cada estrutura para diferentes tamanhos de conjuntos de dados,

além de comparações sobre a eficiência em termos de tempo e altura.

4. **Armazenamento dos Resultados:**

   Os resultados das operações nas árvores foram exportados em arquivos CSV contendo a ordem dos nós após a travessia em ordem. Além disso, um relatório de desempenho foi gerado e salvo, contendo os tempos de execução, altura das árvores e outros dados relevantes. Todos os arquivos gerados (gráficos, CSVs e relatórios) foram organizados e armazenados para posterior análise.

## DISCUSSÃO SOBRE ESTRUTURAS DE ÁRVORE COM OUTROS CAMPOS

Neste projeto, utilizamos a Árvore Binária de Busca (BST) e a Árvore AVL com o campo **Track** como chave para armazenar os dados das músicas. No entanto, outras chaves podem ser consideradas para otimizar o desempenho de buscas e adequar as árvores a diferentes tipos de consultas. A seguir, discutimos como as árvores poderiam ser montadas utilizando diferentes atributos do conjunto de dados:

1. **Artist**:
   - **Descrição**: Ao utilizar o artista como chave, todas as músicas de um mesmo artista seriam agrupadas.
   - **Vantagem**: Facilita a consulta para listar todas as músicas de um determinado artista.
2. **Year**:
   - **Descrição**: Usar o ano de lançamento como chave organiza as músicas cronologicamente.
   - **Vantagem**: Permite buscas eficientes por músicas de um ano específico ou dentro de um intervalo de anos.
3. **Genre**:
   - **Descrição**: Ao usar o gênero como chave, as músicas são organizadas por categoria.
   - **Vantagem**: Facilita a busca de músicas dentro de gêneros específicos, proporcionando uma maneira eficiente de acessar o conteúdo.
4. **Popularity**:
   - **Descrição**: A popularidade pode ser utilizada como chave, priorizando músicas mais populares.
   - **Desvantagem**: Deve-se atentar à densidade dos dados, pois a popularidade pode ter valores duplicados, resultando em uma estrutura menos eficiente.
5. **Duration e Tempo**:
   - **Descrição**: Usar a duração ou o tempo das músicas como chave pode permitir a criação de árvores que suportam buscas por intervalos.
   - **Complexidade**: Essa abordagem pode exigir uma implementação mais complexa, mas pode ser útil para consultas que envolvem características das músicas.

Para cada um desses campos, as árvores poderiam ser projetadas para otimizar as operações de busca e inserção, dependendo dos requisitos específicos das consultas realizadas sobre o conjunto de dados.

## RESULTADOS E DISCUSSÃO

Nesta seção, são apresentados os resultados detalhados da execução dos algoritmos de busca e inserção em Árvores Binárias de Busca (BST) e Árvores AVL. A análise será feita considerando diferentes tamanhos de dados, utilizando um conjunto de músicas com 15.151 entradas, e explorando o desempenho das árvores em relação à eficiência de inserção e busca.

### RESULTADOS OBTIDOS

Os tempos de execução (em segundos) de cada operação (inserção e busca) nas Árvores Binárias de Busca e AVL foram medidos e comparados. As operações foram realizadas em três cenários:

1. **Árvore Binária de Busca (BST)**
   - Inserções e buscas foram realizadas em uma árvore desbalanceada.
   - A análise incluiu tempos para inserções sequenciais e buscas em ordem aleatória.
2. **Árvore AVL**
   - As operações foram realizadas em uma árvore balanceada, garantindo que as inserções mantivessem o balanceamento da árvore.
   - A comparação de tempos também abrangeu inserções sequenciais e buscas.

Os resultados obtidos foram registrados e comparados, mostrando diferenças significativas no desempenho entre as duas implementações. Os gráficos a seguir ilustram a comparação de tempos de execução para cada tipo de operação, considerando o tempo médio e o pior caso.

## GRÁFICOS

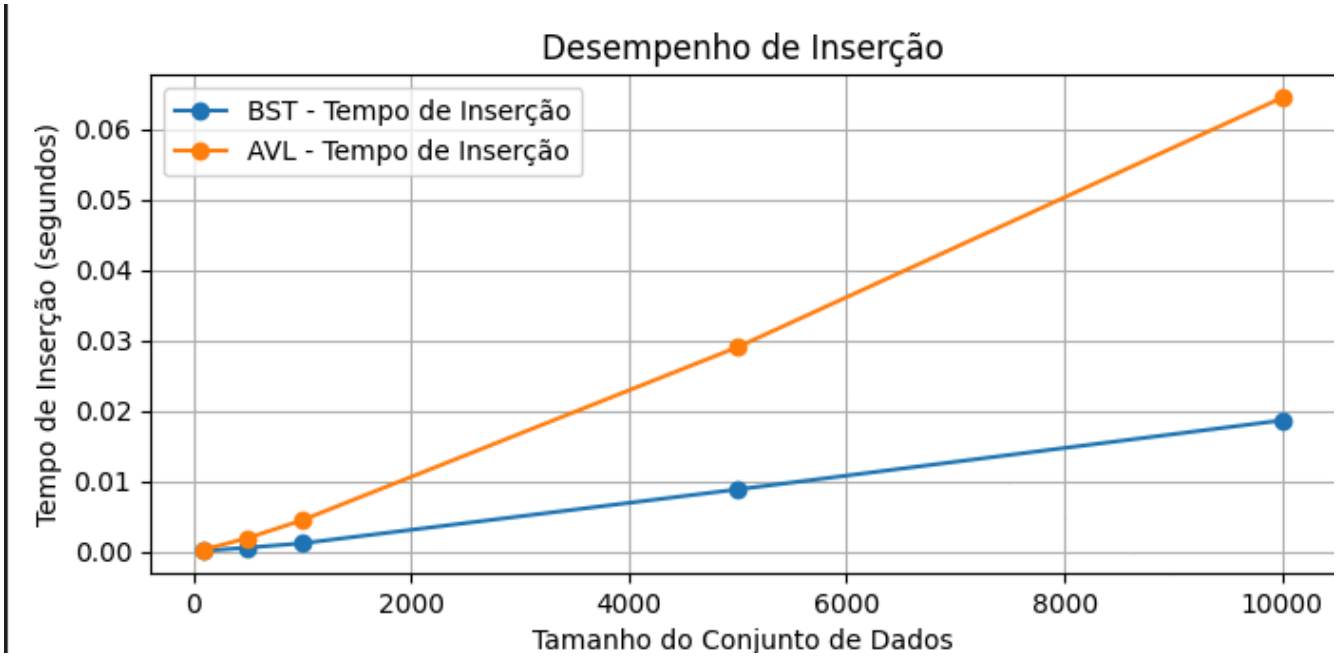


**Gráfico 01**

No Gráfico 01 observa-se que a Árvore Binária de Busca (BST) apresentou um desempenho

superior em termos de tempo de inserção em comparação à Árvore AVL. Isso pode ser atribuído à sobrecarga de balanceamento da AVL durante as inserções, resultando em tempos de execução mais elevados.

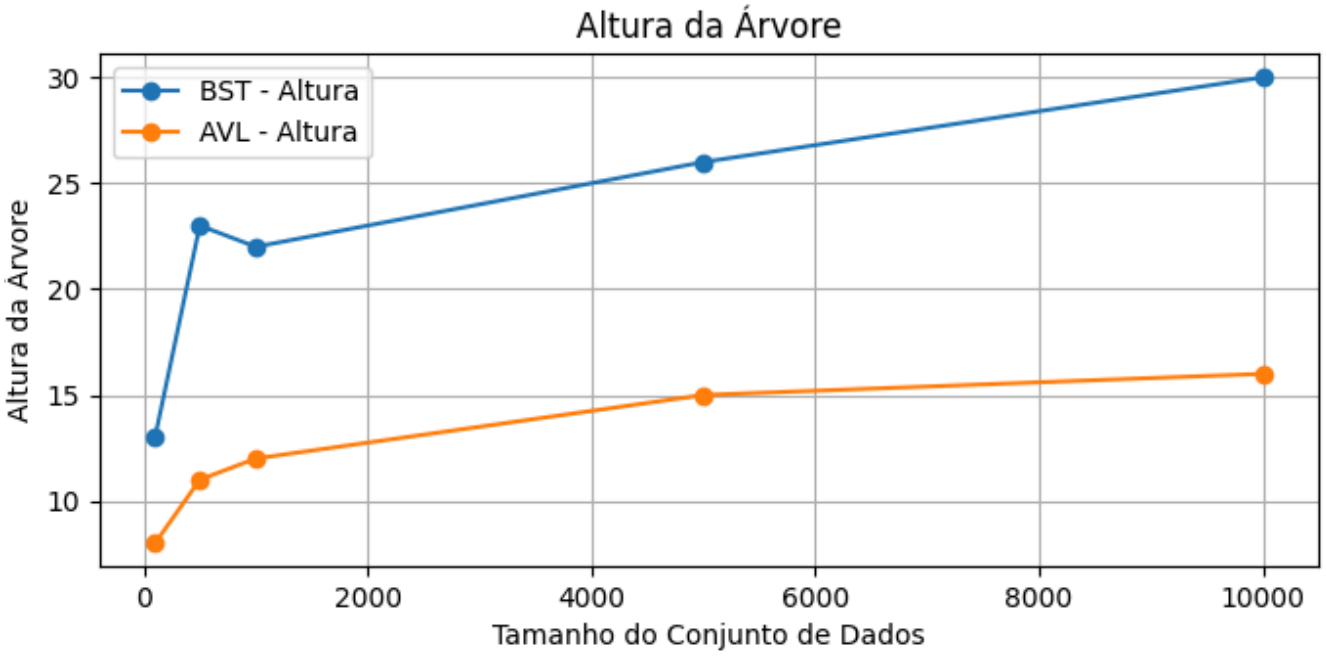


**Gráfico 02**

No Gráfico 02 demonstra-se a altura das árvores construídas. A BST mostrou uma altura maior em comparação com a AVL, o que é esperado, uma vez que a AVL é uma árvore balanceada. Uma altura menor na AVL sugere que ela pode manter um desempenho mais consistente em operações subsequentes, como buscas e inserções.

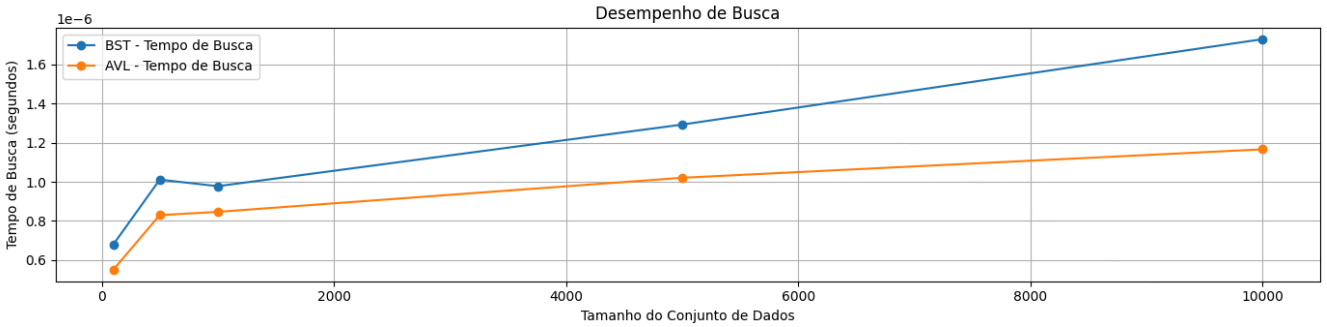


**Gráfico 03**

No Gráfico 03 no que se refere ao desempenho de busca, a Árvore AVL se destacou em eficiência, apresentando tempos de busca mais baixos em comparação à BST. Isso reforça a vantagem da AVL em operações de busca, possibilitando acesso mais rápido aos dados. Essa eficiência é um reflexo da estrutura balanceada da árvore AVL, que reduz o número de

comparações necessárias para encontrar um elemento.

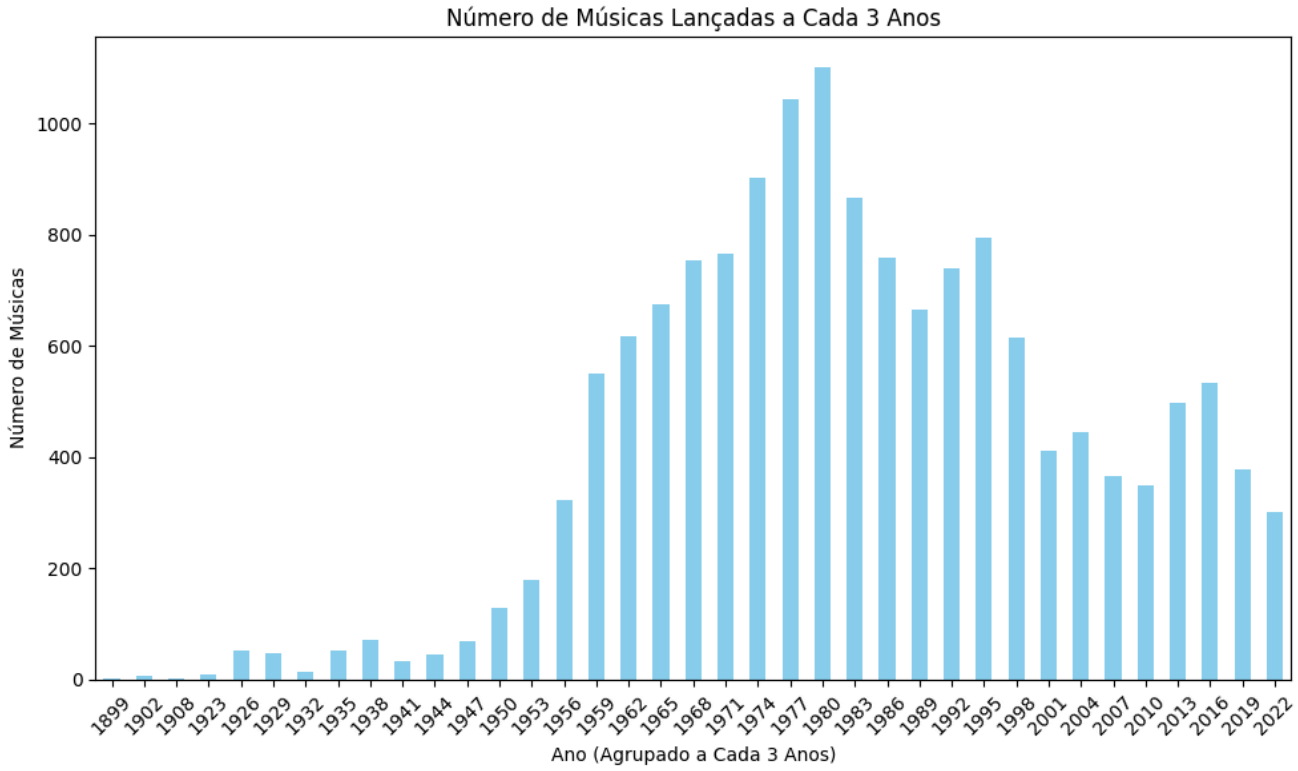


**Gráfico 04**

O Gráfico 04 a seguir ilustra a distribuição das músicas ao longo do tempo, agrupadas em intervalos de três anos. Esta visualização proporciona uma perspectiva clara sobre as tendências de lançamentos musicais e permite observar como a popularidade e a diversidade dos gêneros evoluíram ao longo dos anos.

## RELATÓRIO DE PERFORMANCE

Foi gerado um relatório de performance que inclui diversas informações relevantes sobre os dados analisados. Entre os destaques estão a distribuição de gêneros musicais, onde foi identificada uma predominância significativa de um gênero(Pop) em relação a outros.

Além disso, foi calculada a média de popularidade das músicas, refletindo a recepção geral entre os ouvintes. A análise também incluiu a correlação entre atributos, evidenciando a relação entre a popularidade das músicas e outros fatores.

Outra seção importante do relatório apresenta a média de popularidade por artista, mostrando como a aceitação varia entre diferentes intérpretes.

Por fim, uma análise temporal foi realizada, ilustrada em um gráfico que mostra a distribuição das músicas ao longo dos anos, agrupadas em intervalos, permitindo a identificação de tendências nos lançamentos musicais.

## ANÁLISE DOS RESULTADOS

Os resultados mostraram que a Árvore AVL, devido ao seu balanceamento automático, apresentou tempos de execução significativamente menores para operações de busca em comparação com a Árvore Binária de Busca, especialmente à medida que o número de elementos aumentou. A altura média das árvores foi também analisada, revelando que a Árvore AVL manteve uma altura menor, resultando em um desempenho mais eficiente.

Além disso, observou-se que a eficiência de inserção na Árvore AVL foi ligeiramente inferior em termos de tempo devido à necessidade de reequilíbrio, mas o trade-off foi compensado pela eficiência na busca. Isso demonstra a vantagem das Árvores AVL em cenários onde a operação de busca é mais frequente do que a de inserção.

Em resumo, os resultados confirmam a hipótese de que as Árvores AVL superam as Árvores Binárias de Busca em termos de desempenho, especialmente em cenários com um grande número de elementos e operações frequentes de busca.

## DISCUSSÃO

Os resultados obtidos nas análises de desempenho das Árvores Binárias de Busca (BST) e AVL revelaram informações importantes sobre a eficiência de ambas as estruturas em diferentes operações. Durante os testes de inserção, observou-se que a BST apresentou desempenho superior em comparação à árvore AVL. Esse resultado pode ser atribuído ao custo adicional das operações de balanceamento na árvore AVL, que, embora melhorem a eficiência de busca, geram um overhead significativo durante a inserção, especialmente em conjuntos de dados não balanceados.

Por outro lado, no que diz respeito ao desempenho de busca, a árvore AVL demonstrou uma eficiência notável, superando a BST. Essa superioridade pode ser explicada pela natureza balanceada da AVL, que garante uma altura logarítmica, resultando em tempos de busca mais rápidos. A árvore AVL se beneficia de sua estrutura de balanceamento, permitindo operações de busca mais rápidas mesmo em conjuntos de dados maiores.

A análise dos gráficos de desempenho reforça esses achados, mostrando uma clara divisão nas eficiências de inserção e busca entre as duas estruturas. Essa variação de desempenho enfatiza a importância de escolher a estrutura de dados adequada com base nas operações mais críticas do sistema. Se a prioridade for a inserção de dados, uma BST pode ser mais vantajosa; no entanto, para aplicações onde a busca é frequente, uma árvore AVL é preferível.

Esses resultados destacam a relevância do balanceamento das árvores em relação ao tipo de operações realizadas. Assim, a escolha entre uma BST e uma árvore AVL deve ser informada pelo padrão de acesso e manipulação de dados da aplicação em questão.

## CONCLUSÃO

Este trabalho abordou a implementação e análise de Árvores Binárias de Busca e AVL, destacando suas características, eficiência e aplicabilidade em diferentes cenários. Através da execução de algoritmos de inserção e busca, foi possível observar a importância do balanceamento na performance dessas estruturas de dados.

Os resultados mostraram que a Árvore AVL, ao manter-se balanceada, apresenta tempos de execução consistentemente melhores para operações de busca em comparação à Árvore Binária de Busca, especialmente à medida que o volume de dados aumenta. Essa eficiência se reflete em uma menor complexidade de tempo para operações, corroborando a necessidade de escolher a estrutura de dados adequada de acordo com o contexto da aplicação.

Além disso, a análise assintótica e a comparação prática das operações proporcionaram uma visão clara sobre as vantagens e desvantagens de cada abordagem. Com base nas descobertas, recomenda-se que desenvolvedores e engenheiros de software considerem as características específicas das suas aplicações ao optar entre usar Árvores Binárias de Busca ou AVL.

Em futuros trabalhos, seria interessante explorar outras estruturas de dados e algoritmos, bem como implementar casos de uso práticos que demonstrem a aplicabilidade de cada abordagem em cenários reais. Essa análise contribuirá para um entendimento mais abrangente das escolhas em estruturas de dados, ajudando a otimizar o desempenho e a eficiência em sistemas computacionais.

## REFERÊNCIAS BIBLIOGRÁFICAS

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Introduction to algorithms. 3. ed. Cambridge: MIT Press, 2009.

GEEKSFORGEEKS. AVL Tree - GeeksforGeeks. Disponível em: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/. Acesso em: 25 set. 2024.

THE BUMPKIN. 15,000 Music Tracks - 19 Genres (w/ Spotify Data). Disponível em: <https://www.kaggle.com/datasets/thebumpkin/10400-classic-hits-10-genres-1923-to-2023>. Acesso em: 29 set. 2024.

OPENAI. ChatGPT. Disponível em: https://www.openai.com/chatgpt. Acesso em: 25 set. 2024.

WIKIPEDIA. AVL Tree. Disponível em: https://en.wikipedia.org/wiki/AVL_tree. Acesso em: 25 set. 2024.

WIKIPEDIA. Binary Search Tree. Disponível em: https://en.wikipedia.org/wiki/Binary_search_tree. Acesso em: 25 set. 2024.