UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
Curso Bacharelado em Sistemas de Informação
Disciplina: Estrutura de Dados II
Docente: Juliana Oliveira de Carvalho
Discente: Erlanny Rodrigues da Silva Rêgo

Relatório Trabalho I

Resumo

Este trabalho compara as estruturas de dados Árvore Binária de Busca (ABB) e Árvore AVL em um sistema de gerenciamento de biblioteca musical. A ABB organiza os dados pela regra da busca binária, com elementos menores à esquerda e maiores à direita. Embora a inserção seja rápida, a ausência de balanceamento pode tornar a árvore ineficiente em grandes volumes de dados. A Árvore AVL, por sua vez, realiza balanceamento automático, garantindo complexidade O(log n) nas operações. A aplicação gerencia artistas, álbuns e músicas, utilizando árvores para cadastro, busca e remoção. A ABB demonstrou melhor desempenho na inserção (0,715 s), enquanto a AVL se destacou nas buscas (0,433 ms), devido ao seu balanceamento. Foram implementadas funções específicas para inserção, busca e remoção, com destaque para as rotações na AVL. Os testes, realizados em Windows 11 com processador Intel Core i3, mostraram que a ABB é mais eficiente para inserções, e a AVL para buscas.

1 Introdução

Na computação, utilizamos dados, e a forma como esses dados são agregados e organizados depende diretamente de como serão utilizados e processados. Essa organização leva em conta fatores como a eficiência na busca, a quantidade de dados, sua complexidade e os relacionamentos entre eles. As diferentes maneiras de organizar esses dados são conhecidas como estruturas de dados. Elas podem ser estruturas lineares (como arrays, listas ligadas, pilhas e filas), estruturas não-lineares (como árvores e grafos), e estruturas como tabelas hash.

Uma estrutura de dados não linear bastante comum é a árvore, utilizada para armazenar informações de forma hierárquica, permitindo representar relações de dependência ou ancestralidade entre os elementos. A forma mais conhecida dessa estrutura é a árvore binária de busca (ABB), que consiste em um conjunto finito de nós, onde cada nó pode ter, no máximo, dois filhos: um à esquerda e outro à direita. Essa organização segue o princípio do algoritmo de busca binária, no qual os valores menores que o nó raiz são posicionados à esquerda e os valores maiores, à direita, facilitando buscas, inserções e remoções de forma eficiente.

As árvores de altura balanceada, ou árvores AVL, foram introduzidas em 1962 por Adelson-Velskii e Landis. Elas são chamadas assim porque mantêm a altura equilibrada entre

as subárvores, o que garante um bom desempenho nas operações de busca, inserção e remoção, mesmo com um grande número de elementos. Em uma árvore AVL com n elementos, essas operações têm complexidade logarítmica, ou seja, O(log n). A principal característica de uma árvore AVL é que, para qualquer nó, a diferença entre a altura das subárvores esquerda e direita (chamada de fator de balanceamento) deve ser -1, 0 ou 1. Isso significa que a árvore está sempre quase perfeitamente balanceada, o que evita degenerações em listas lineares e garante eficiência nas operações.

Este trabalho tem como objetivo apresentar e comparar as duas estruturas de dados descritas, destacando suas principais características e desempenho. A seguir, serão detalhadas as particularidades de cada estrutura, os problemas propostos e os aspectos funcionais considerados na implementação das soluções, com o intuito de resolver os desafios de forma eficiente. Por fim, serão discutidos os aspectos funcionais envolvidos na resolução dos problemas, incluindo decisões de implementação, técnicas de balanceamento, estruturação do código e possíveis melhorias futuras.

2 Seções Específicas

Nesta seção, serão apresentadas as partes específicas do trabalho. Primeiramente, é fundamental compreender um dos principais mecanismos da árvore binária AVL: as rotações, que são utilizadas para manter o balanceamento da estrutura. Abaixo, será feita uma breve explicação sobre esse conceito, seguida pela descrição dos problemas propostos e suas respectivas soluções, utilizando as estruturas de dados mencionadas anteriormente.

2.1 Rotações da Árvore Binária AVL

Para que a árvore AVL mantenha seu equilíbrio, ela utiliza um mecanismo chamado rotação. Sempre que uma inserção ou remoção de elemento causa um desbalanceamento, ou seja, quando a diferença entre as alturas das subárvores esquerda e direita de um nó passa de 1, a árvore se reorganiza automaticamente para corrigir isso. Essa reorganização acontece por meio de rotações, que ajustam a estrutura da árvore para que ela continue eficiente nas operações de busca, inserção e remoção.

As rotações podem ser simples ou duplas, dependendo de onde ocorre o desbalanceamento. Quando um novo valor é inserido no lado esquerdo do filho esquerdo de um nó, por exemplo, a árvore realiza uma rotação simples para a direita, promovendo o filho esquerdo para o lugar do nó desbalanceado. Da mesma forma, se o valor for inserido no lado direito do filho direito, a rotação é para a esquerda. Já nos casos em que o desequilíbrio acontece de forma mais interna, como no lado direito do filho esquerdo ou no lado esquerdo do filho direito, são necessárias rotações duplas: primeiro uma rotação simples no filho, seguida de uma segunda rotação no nó principal. Representadas na Figura 1.

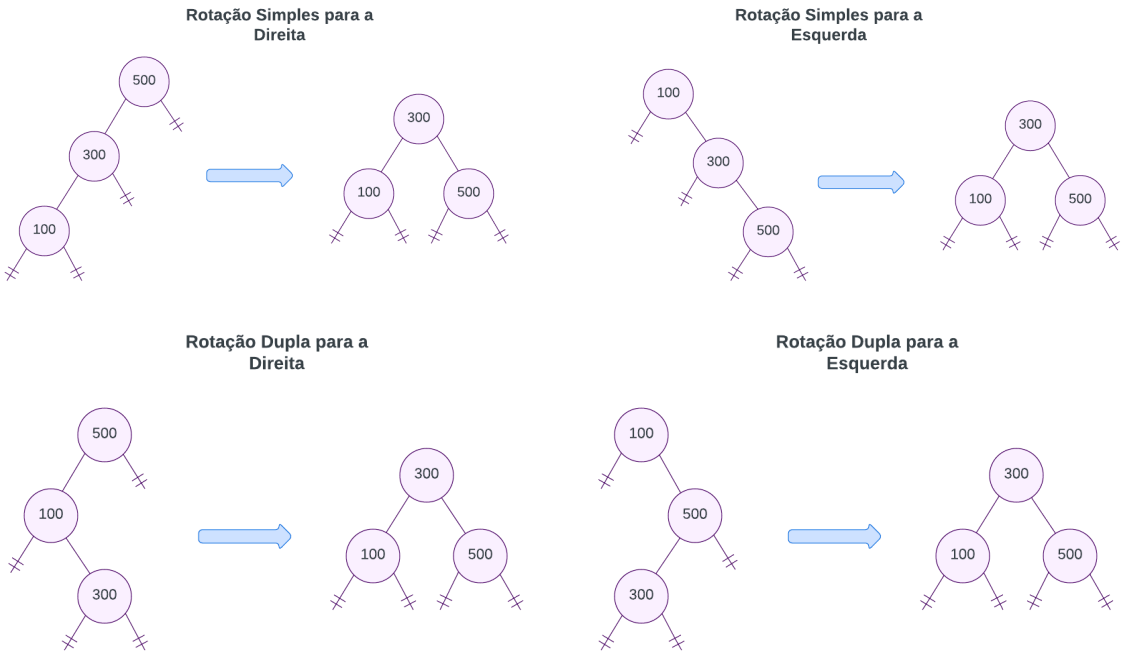

Figura 1 - Rotações da Árvore AVL

Essas rotações não apenas corrigem o desequilíbrio, mas também garantem que a estrutura da árvore permaneça organizada. Assim, mesmo com várias inserções ou remoções, a árvore continua funcionando de forma rápida e eficiente, mantendo o acesso aos dados sempre otimizado.

2.2 Estrutura do trabalho

A estrutura do trabalho está organizada em uma pasta principal chamada “trabalho-I”, que contém duas subpastas: uma referente à árvore binária de busca (ABB) e outra à árvore AVL. Cada uma dessas subpastas possui uma pasta chamada “includes”, onde estão localizados os arquivos de cabeçalho com as declarações das funções e estruturas de dados utilizadas, e uma pasta “src”, que contém as implementações dessas funções. Como mostra a Figura 2.

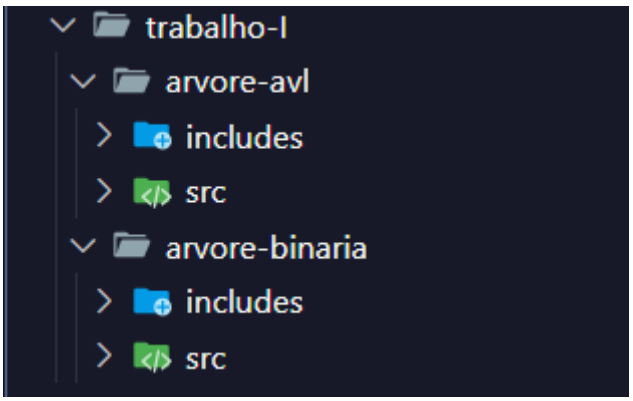

Figura 2 - Estrutura do trabalho

Para executar o projeto, basta compilar e rodar o arquivo executável “main.exe” diretamente no terminal, dentro do diretório correspondente à árvore desejada, como por exemplo em “../arvore-binaria/src/” para a ABB e “../arvore-avl/src/” para a AVL.

2.3 Problema

O problema propõe um programa para gerenciar uma biblioteca de música usando árvores binárias. Os dados são organizados hierarquicamente: artistas possuem uma árvore de álbuns, e cada álbum tem uma árvore de músicas. O sistema deve permitir cadastro e listagem de artistas, álbuns e músicas, com validações para evitar duplicatas. Também deve permitir criar playlists em árvore, compostas por músicas já cadastradas, e oferecer funções para consultar, remover músicas de playlists e excluir playlists. Músicas só podem ser removidas dos álbuns se não estiverem em nenhuma playlist.

2.3.1 Árvore Binária de Busca

Na implementação da Árvore Binária de Busca (ABB), foram desenvolvidas diversas funções. Devido à grande quantidade de funcionalidades presentes no código, torna-se inviável explicar cada uma individualmente. Portanto, optei por agrupá-las em três categorias principais: inserção, busca e remoção.

A seguir, explicamos as funções utilizando como exemplo a estrutura de artista, pois as demais seguem a mesma lógica tanto para inserção quanto para busca.

Inserção

- cadastrarArtista(): Verifica se já existe um artista com o nome informado na árvore. Caso não exista, cria um novo nó com os dados fornecidos e tenta inseri-lo na estrutura.
- criarArtista(): Responsável por criar o nó do artista. Essa função aloca memória, copia os dados fornecidos e inicializa os ponteiros e contadores.
- insereArtista(): Realiza a inserção ordenada do novo artista na árvore com base na ordem alfabética do nome, comparando-o com os nós existentes e posicionando-o à esquerda ou à direita, conforme necessário.

Busca

- buscaArtista(): Realiza a busca por um artista na árvore binária, utilizando o nome como chave para navegação na estrutura.

Remoção

A remoção na ABB é utilizada em três situações distintas no projeto. Como duas delas compartilham lógica semelhante, detalharemos o caso mais completo, que envolve múltiplas verificações:

- removerMusica(): Remove uma música da árvore binária de músicas, seguindo as regras de remoção de nós folha, com um filho ou com dois filhos.
- musicaEmPlaylists(): Percorre todas as playlists verificando se a música está presente em alguma delas.
- removerMusicaDeAlbum(): Localiza o artista e o álbum correspondentes e remove a música apenas se ela não estiver presente em nenhuma playlist, exibindo mensagens adequadas ao usuário.

2.3.2 Árvore Binária AVL

A implementação das funções da AVL segue, em sua maior parte, a mesma lógica utilizada na ABB. A principal diferença está na adição do balanceamento automático após as operações de inserção, garantindo que a altura da árvore permaneça ideal para manter a eficiência das operações.

Para realizar esse balanceamento, foram adicionadas as seguintes funções auxiliares:

- maiorA(): Recalcula a altura de um nó com base nas alturas de seus filhos esquerdo e direito.
- alturaNoA(): Recebe um ponteiro para um nó do tipo Artista e retorna sua altura.
- rotacaoEsqA(): Realiza uma rotação simples à esquerda. Caso o filho direito do nó raiz não seja nulo, ele é promovido à nova raiz, e os ponteiros são reorganizados de forma que o antigo nó raiz se torne o filho esquerdo da nova raiz. Ao final, as alturas dos nós afetados são atualizadas.
- rotacaoDirA(): Executa uma rotação simples à direita, com lógica análoga à rotação à esquerda. Se o filho esquerdo do nó raiz for não nulo, ele se torna a nova raiz, e os ponteiros são reorganizados. As alturas dos nós também são atualizadas.
- fatorBalanceamentoA(): Calcula o fator de balanceamento de um nó, que é a diferença entre a altura da subárvore esquerda e da subárvore direita.
- balanceamentoA(): Utiliza o fator de balanceamento para determinar se a árvore está desbalanceada e, se necessário, realiza as rotações apropriadas (simples ou duplas) para restaurar o equilíbrio.

2.4 Ambiente de Testes

O ambiente de testes foi feito em um notebook com as seguintes especificações de hardware e software:

- Sistema operacional: Windows 11 Home Single Language 64 bits (10.0, Compilação 26100)
- Fabricante do sistema: Dell Inc.
- Modelo do sistema: Inspiron 15 3511
- BIOS: 1.35.0
- Processador: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz (4 CPUs), ~3.0GHz
- Memória: 8192MB RAM
- Arquivo de paginação: 16116MB usados, 1813MB disponíveis
- Versão do DirectX: DirectX 12

2 Resultados da Execução do Programa

Os resultados apresentados na Tabela 1 mostram a média do tempo de execução das inserções em ambas as estruturas, em segundos, e o tempo médio de busca de um artista, em milissegundos. Para isso, foram realizadas 30 execuções com a inserção de 100 mil artistas, cada um contendo um álbum com cinco músicas.

Na etapa de busca, foi utilizado o "Artista_49985", escolhido por ser um nó localizado em uma região mais profunda da árvore, o que permite melhor avaliação do desempenho. Foram realizadas 30 buscas desse mesmo nó em ambas as árvores, garantindo assim a consistência dos resultados, já que o uso de nós diferentes poderia afetar a comparação.

Árvore	Tempo Médio de Isenção (em segundos)	Tempo Médio de Busca e Impressão (em milissegundos)
Árvore Binária de Busca	0.715	1.067
Árvore Binária AVL	0.840	0.433

Tabela 1 - Resultados da Execução

Primeiramente ao analisar o tempo de inserção, observa-se que a árvore AVL apresenta um tempo maior ao inserir 100 mil artistas. Isso ocorre devido aos processos de rotação dos nós realizados durante o balanceamento da árvore, necessários para manter sua estrutura equilibrada, especialmente diante da grande quantidade de elementos. Por outro lado, a árvore ABB possui um tempo de inserção menor, pois os nós são adicionados sem a necessidade de balanceamento. Esses resultados são representados visualmente no Gráfico 1, que também exibe os dados obtidos em cada uma das 30 execuções realizadas.

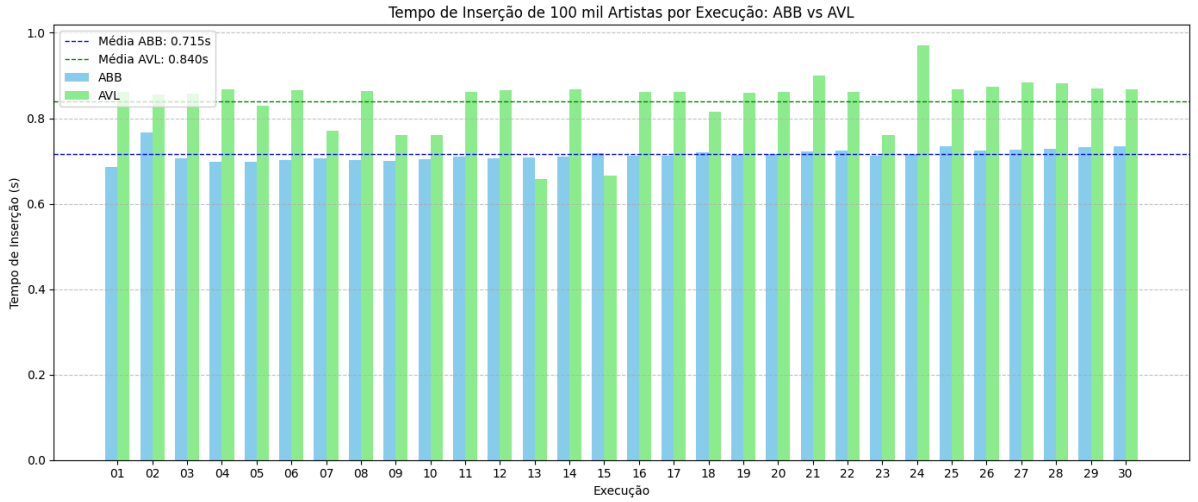

Gráfico 1 - Tempo Médio de Isenção

Na operação de busca, observa-se que, apesar da árvore AVL apresentar um tempo de inserção maior, ela realiza buscas mais rapidamente. Isso se deve ao seu balanceamento automático, que garante uma altura menor e mais uniforme, facilitando o acesso aos dados. Em contrapartida, a árvore ABB, por não realizar balanceamento, pode se tornar desbalanceada e, consequentemente, mais lenta nas operações de busca. Isso mostra visualmente no Gráfico 2.

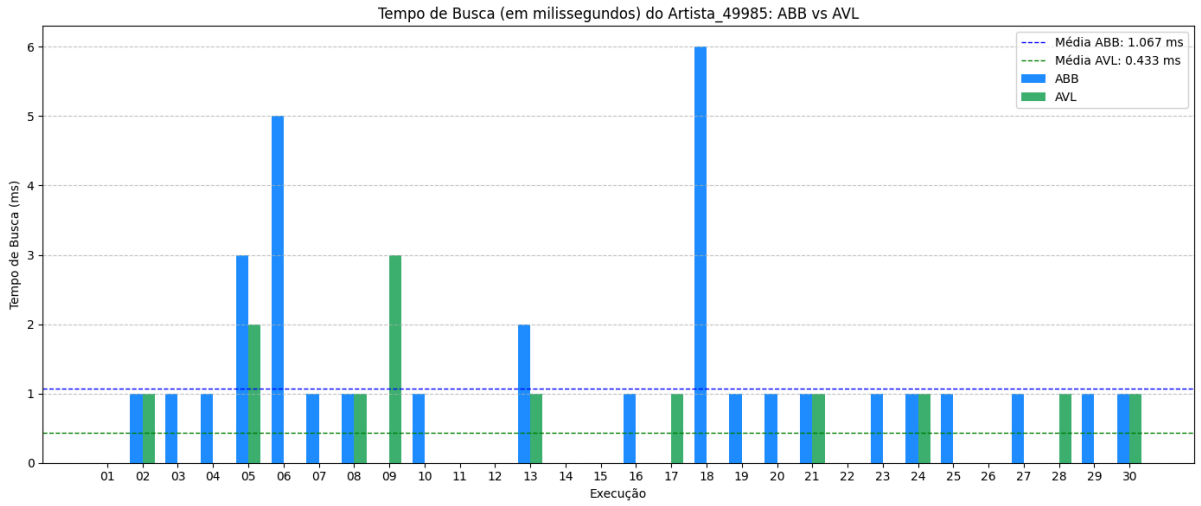

Gráfico 2 - Tempo de Busca

Dessa forma, os resultados obtidos evidenciam as principais diferenças de desempenho entre as árvores AVL e ABB. Enquanto a ABB se mostra mais eficiente na etapa de inserção, a AVL compensa esse custo com buscas significativamente mais rápidas, graças à sua estrutura balanceada. A escolha entre uma ou outra deve, portanto, considerar o tipo de operação que será mais frequente no sistema em questão. Em aplicações com alta demanda por buscas rápidas e frequentes, a AVL é mais vantajosa. Já em cenários onde a inserção é predominante e o tempo de resposta da busca não é crítico, a ABB pode ser uma alternativa mais simples e eficiente.

3 Conclusão

Ao longo deste trabalho, foi possível compreender e comparar duas estruturas fundamentais no campo das estruturas de dados: a Árvore Binária de Busca (ABB) e a Árvore AVL. Ambas se mostraram eficazes no gerenciamento hierárquico de dados, mas com características e comportamentos diferentes conforme a operação realizada. A ABB demonstrou melhor desempenho na inserção de elementos, principalmente por sua simplicidade estrutural e ausência de processos adicionais de balanceamento. No entanto, essa mesma característica pode se tornar uma desvantagem em operações de busca, principalmente à medida que a árvore cresce e se torna desbalanceada.

A árvore AVL, embora apresente maior complexidade e tempo de inserção devido às rotações necessárias para manter seu equilíbrio, compensa essa desvantagem com tempos de busca muito mais eficientes. Seu balanceamento constante garante que a profundidade da árvore permaneça ideal, resultando em acesso rápido aos dados mesmo em grandes volumes.

A análise dos resultados reforça a importância de escolher a estrutura de dados mais adequada ao tipo de operação que será mais frequente em um sistema. Em aplicações com alta taxa de leitura e necessidade de rapidez na busca, a AVL se mostra mais eficiente. Já em sistemas onde a inserção de dados é o foco principal, e o desempenho da busca é menos crítico, a ABB pode ser uma solução mais simples e eficiente. Por fim, este trabalho também evidenciou a importância de uma boa organização de código, da utilização de testes em ambiente controlado e da análise de desempenho como ferramentas essenciais no desenvolvimento de soluções computacionais eficientes.

Referências

ALURA. Estruturas de dados: por onde começar? *Alura*, 2022. Disponível em: https://www.alura.com.br/artigos/estruturas-de-dados-introducao. Acesso em: 25 abr. 2025.

BARANAUSKAS, José Augusto. Árvores AVL. Disponível em: https://dcm.ffclrp.usp.br/~augusto/teaching/aedi/AED-I-Arvores-AVL.pdf. Acesso em: 25 abr. 2025.

TENENBAUM, Aaron M; Langsan Yedidyah; Augenstein Moshe J. Estruturas de Dados usando C. São Paulo: Pearson Makron Books, 1995. 884p