Relatório Técnico

Gabriel Lima Pereira

Resumo

Diante da apresentação do conceito da árvore binária em aula e do desenvolvimento de um programa que utilizasse os recursos dessa estrutura de dados, o trabalho que será nesse documento apresentado tem como objetivo justamente desenvolver funcionalidades que explorem tanto a Árvore Binária de Busca quanto a Árvore AVL. Nisso há também um estudo comparativo entre ambas no que diz respeito aos tempos de inserção, exibindo resultados de testes relacionados.

1 Introdução

Diante da necessidade de armazenamento de um grande volume de dados na memória de um computador se faz fundamental a otimização dos métodos de organização desses dados. Isso não só otimiza a devida inserção de informações, mas também a consulta e modificação delas em memória. Através de estruturas de dados lineares e não lineares (como listas ou árvores) essa carência pode ser suprida. Todavia, nem toda estrutura é ideal para todo caso.

Listas encadeadas são lineares, portanto, seguem um método de busca sequencial que ao lidar com uma quantidade considerável de dados pode tornar esse e mais processos lentos. Mesmo no caso de árvores é relevante preocupar-se com essa mesma problemática, evitando o uso desnecessário de recursos computacionais e o aprimoramento do gerenciamento do armazenamento.

O projeto desenvolvido tem como escopo um sistema que gerencie árvores binárias através da temática de música, aninhando essas estruturas em structs relacionadas a músicas, álbuns, artistas e playlists. As funcionalidades seguem as competências básicas dessa estrutura: busca, inserção, remoção e exibição. Foram feitos também testes comparativos entre os tempos das inserções da Árvore Binária de Busca (BB) e a Árvore AVL, respectivas árvores trabalhadas.

2 Seções específicas

2.1 Árvores Binárias

Árvores binárias pertencem ao grupo das estruturas de dados não lineares, ou seja, os elementos não estão organizados de forma sequencial. Nessa estrutura, cada elemento é chamado de nó e pode ter até dois nós filhos, além de um único nó pai — com exceção

da raiz, que não possui pai. A inserção dos dados pode seguir determinadas regras, o que facilita a busca por informações, já que nem todos os nós precisam ser examinados.

2.1.1 Estrutura básica de um nó

- Valor armazenado.
- Ponteiro para o filho à esquerda.
- Ponteiro para o filho à direita.

2.2 Árvore Binária de Busca

A Árvore Binária de Busca organiza os nós de forma que, para qualquer nó, os valores menores ficam à sua esquerda e os maiores, à direita. Essa disposição permite localizar elementos com menos comparações, seguindo apenas um dos caminhos da árvore. Se um caminho for seguido até o fim sem encontrar o valor, significa que ele não existe na estrutura.

Um ponto a considerar sobre a BB é que sua eficiência pode ser comprometida pela ordem em que os valores são inseridos. Se os dados forem adicionados já em ordem crescente ou decrescente, a árvore pode perder o formato de ramificação ideal e se comportar quase como uma lista, o que prejudica o desempenho nas buscas. Por isso, essa estrutura funciona melhor quando os dados são inseridos de forma aleatória.

2.3 Árvore AVL

A árvore AVL é uma variação da Árvore Binária de Busca que inclui um mecanismo para manter seus ramos mais equilibrados. Sempre que um elemento é inserido ou removido, a árvore verifica um valor chamado fator de balanceamento. Esse valor indica se a árvore está simétrica ou mais carregada para um lado. Se houver desequilíbrio, são feitas rotações para reorganizar os nós.

Essas rotações tornam a árvore AVL mais estável em operações de busca do que a ABB tradicional, pois evitam que a estrutura fique semelhante a uma lista. No entanto, esse processo de balanceamento exige mais verificações durante inserções e remoções, o que pode aumentar o custo dessas operações. Assim, o uso da AVL é mais vantajoso em contextos onde a busca é a operação mais frequente. Em geral, os nós de uma AVL também guardam um campo adicional para armazenar a altura do nó, que é usado no cálculo do balanceamento.

2.4 Funcionalidades

Utilizando a biblioteca ‘time.h‘ para a medição dos tempos de inserção, o desenvolvimento do código foi feito inteiramente em linguagem C, . A subseção 2.4.1 trará os tipos de structs definidos no programa, junto com os atributos associados a cada um. As subseções 2.4.2, 2.4.3 e 2.4.4 apresentarão, de forma resumida, as funções principais do sistema. Vale ressaltar que foi feita uma generalização de uma parte dessas por conta

da mesma lógica algorítmica empregada na maioria. Em 2.4.5, será detalhada a lógica aplicada para registrar testes. Por fim, a subseção 2.4.6 abordará o método usado para gerar sequências aleatórias por meio de valores aleatórios.

2.4.1 Structs

2.4.1.1 Struct `InfoMusica`

- Título da música
- Duração da música

2.4.1.2 Struct `ArvMusica`

- Informações da música (`InfoMusica`)
- Altura do nó na árvore
- Ponteiro para o filho à esquerda
- Ponteiro para o filho à direita

2.4.1.3 Struct `InfoAlbum`

- Título do álbum
- Ano de lançamento
- Quantidade de músicas no álbum
- Árvore de músicas (`ArvMusica`)

2.4.1.4 Struct `ArvAlbum`

- Informações do álbum (`InfoAlbum`)
- Altura do nó
- Ponteiro para o filho à esquerda
- Ponteiro para o filho à direita

2.4.1.5 Struct `InfoArtista`

- Nome do artista
- Tipo do artista (ex: solo, banda)
- Estilo musical
- Quantidade de álbuns
- Árvore de álbuns (`ArvAlbum`)

2.4.1.6 Struct `ArvArtista`

- Informações do artista (`InfoArtista`)
- Altura do nó
- Ponteiro para o filho à esquerda
- Ponteiro para o filho à direita

2.4.1.7 Struct `InfoMusP`

- Nome da música
- Nome do artista
- Nome do álbum

2.4.1.8 Struct `ArvMusP`

- Informações da música para playlist (`InfoMusP`)
- Altura do nó
- Ponteiro para o filho à esquerda
- Ponteiro para o filho à direita

2.4.1.9 Struct `InfoPlaylist`

- Nome da playlist
- Árvore de músicas (`ArvMusP`)

2.4.1.10 Struct `ArvPlaylist`

- Informações da playlist (`InfoPlaylist`)
- Altura do nó
- Ponteiro para o filho à esquerda
- Ponteiro para o filho à direita

2.4.2 Funções de leitura

A leitura não recebe parâmetros. Dentro dela uma struct do tipo de dado que é tratado na situação é criada e são requisitados ao usuário os dados. Ponteiros são inicializados e, por fim, é retornada a variável.

2.4.3 Funções de alocação

A alocação recebe uma struct da informação respectiva e a insere em um nó alocado dinamicamente, retornando este.

2.4.4 Funções de inserção

A inserção recebe como parâmetro o endereço do nó da raiz da árvore e o nó recém alocado. Ela percorre a árvore e retorna um valor inteiro binário: 0 caso a inserção tenha falhado (o valor já se encontra na estrutura) ou 1 caso tenha sido efetuada.

2.4.4.1 Inserção de música em playlist

Para que uma música seja inserida em uma playlist é antes verificada a existência dessa música na árvore de música de algum artista. Caso não exista não é feita a inserção.

2.4.5 Funções de busca

A busca recebe a raiz da árvore e o identificador único respectivo ao nó a ser buscado (uma string). Ela percorre a estrutura e em caso positivo retorna o nó; caso contrário é retornado um ponteiro nulo.

2.4.6 Funções de remoção

A remoção recebe o endereço da raiz da árvore e a string que identifica o nó a ser removido. Ela percorre a árvore e caso encontre o nó respectivo, efetua a remoção. É retornado um valor inteiro binário: 1 para uma remoção efetiva e 0 caso contrário.

2.4.6.1 Remoção de música

Para que uma música de um artista seja removida é antes feita uma verificação. Através das funções de busca é antes garantido que a respectiva música não está presente em nenhuma playlist. Só assim é efetuada a remoção.

2.4.7 Funções de impressão

A impressão recebe a raiz da árvore e percorre ela exibindo os dados do nó atual. Em todo o projeto foi mantido a forma de impressão em ordem.

2.4.8 artistaPorTipo

Esta recebe a raiz da árvore de artistas e uma string representando o tipo de artista a ser listado. A função percorre a árvore e imprime os dados dos artistas cujo tipo coincide com a string fornecida.

2.4.9 artistaPorEstilo

A função recebe como parâmetros a raiz da árvore de artistas e uma string contendo o estilo a ser filtrado. A árvore é percorrida em ordem e os dados dos artistas cujo estilo corresponde ao fornecido são exibidos.

2.4.10 artistaPorTipoEstilo

A função recebe a raiz da árvore de artistas, o tipo e o estilo a serem buscados. A árvore é percorrida e são listados apenas os artistas que possuem tanto o tipo quanto o estilo informados.

2.4.11 albunsArtista

Ela recebe a raiz da árvore de artistas e o nome de um artista. Através de uma busca, localiza o artista correspondente e, se encontrado, imprime os dados de seus álbuns.

2.4.12 imprimeAlbumAno (por artista)

A função recebe a árvore de álbuns de um artista previamente buscado e um valor inteiro representando o ano. Ela percorre a árvore e imprime os álbuns que correspondem ao ano informado. Caso o artista não seja encontrado, é exibida uma mensagem informando a ausência.

2.4.13 imprimeArvMus

Essa imprime as músicas contidas na árvore de músicas de um álbum específico. O artista e o álbum são localizados por meio de buscas, e em caso de sucesso, a árvore

de músicas é percorrida e exibida. Caso o artista ou o álbum não sejam encontrados, mensagens apropriadas são exibidas.

2.4.14 imprimeAlbumAno (geral)

A função `imprimeAlbumAno` também pode ser utilizada de forma geral com a raiz da árvore de artistas. Nesse caso, a função percorre todos os artistas e imprime os álbuns de cada um que correspondem ao ano fornecido.

2.4.15 dadosMusica

A função `dadosMusica` recebe a raiz da árvore de artistas, o nome da música a ser buscada, e ponteiros para armazenar a duração e o nome do álbum ao qual a música pertence. A árvore é percorrida e, em caso de sucesso, os dados são preenchidos nas variáveis fornecidas. Caso a música não seja localizada, nenhuma alteração é feita.

3 Testes

Os testes feitos são relativos à inserção em ambas estruturas. Antes de cada um a árvore foi povoada com cem mil nós em um sentido crescente, decrescente e aleatório de dados. Seguindo essa mesma via de inserção, após esse povoamento, foram realizadas trinta inserções de números, também do maior ao menor, menor ao maior e aleatoriamente - cada ordem respectiva à forma como a árvore foi povoada.

Os valores aleatórios foram gerados por randomização através da função *rand* da biblioteca *time.h*. Foi colhida uma semente gerada com base no tempo atual e os valores foram gravados em um arquivo da biblioteca c.*txt* e lidos posteriormente pelos arquivos de teste. Para a coleta do tempo foi feita através da função *clock*. Ela retorna o número de clocks da máquina. Foram utilizados meios de conversão para a impressão dos tempos.

3.1 Especificações de hardware

Segue uma tabela com detalhes da máquina utilizada para os testes.

Tabela 1 – Hardware e sistema operacional

Componente	**Especificação**
Modelo	Acer Aspire A515-45
Processador	AMD Ryzen™ 7 5700U with Radeon™ Graphics × 16
Memória RAM	12,0 GiB
Sistema Operacional	Ubuntu 24.04.2 LTS

4 Resultados

Como citado anteriormente, o povoamento da árvore com cem mil nós seguiu o mesmo padrão da inserção inicial de trinta nós, sendo realizadas em três ordens distintas:

aleatória, crescente e decrescente. Esses testes foram executados tanto para a árvore Binária de Busca (BB) quanto para a árvore AVL, permitindo uma comparação direta entre as estruturas.

Tabela 2 – Média das Inserções em Árvore Binária de Busca

Ordem de Inserção	**Tempo (s)**	**Tempo (µs)**
Aleatória	0,0000038000	3,80
Crescente	0,0000022667	2,27
Decrescente	0,0000008000	0,80

Na árvore Binária de Busca os tempos de inserção foram extremamente baixos, indicando uma performance rápida, especialmente para esse volume de dados. O menor tempo foi observado na inserção decrescente, seguida da crescente, sendo a aleatória a mais custosa entre as três. Embora a diferença seja pequena em valores absolutos, esses dados refletem a influência direta da ordem de inserção na estrutura da árvore. No caso da BB, inserções em ordem crescente ou decrescente tendem a gerar árvores degeneradas, semelhantes a listas encadeadas, o que afeta a eficiência de buscas e remoções — embora, curiosamente, o tempo de inserção ainda permaneça baixo, uma vez que não há necessidade de reestruturações complexas.

Tabela 3 – Média das Inserções em Árvore AVL

Ordem de Inserção	**Tempo (s)**	**Tempo (µs)**
Aleatória	0,0196492667	19649,27
Crescente	0,0060657667	6065,77
Decrescente	0,0056731000	5673,10

Por outro lado, os tempos de inserção na árvore AVL foram significativamente mais altos. Essa diferença pode ser atribuída ao mecanismo de balanceamento intrínseco dessa estrutura. Após cada inserção, a AVL executa operações adicionais para manter seu balanceamento — como o cálculo da altura dos nós, avaliação do fator de balanceamento e, quando necessário, a realização de rotações simples ou duplas. Esses procedimentos consomem recursos computacionais adicionais, elevando o tempo de execução.

Interessantemente, a ordem das inserções também influenciou os tempos na AVL. A inserção aleatória gerou o maior tempo médio, provavelmente por exigir mais rotações devido ao padrão imprevisível de dados. Inserções ordenadas (crescente e decrescente), por outro lado, apresentaram tempos menores, mas ainda significativamente maiores do que os da BB. Isso indica que, mesmo que a árvore tenha que ser reestruturada frequentemente nessas ordens, o padrão facilita a previsibilidade dos ajustes de balanceamento.

Em suma, enquanto a BB se mostrou mais rápida para inserções, a AVL oferece a vantagem de manter a árvore balanceada, o que é crucial para garantir eficiência nas operações de busca e remoção em grandes volumes de dados. O custo extra de inserção na AVL é, portanto, compensado por uma performance mais consistente em outras operações.

5 Conclusão

Foi descoberto e aprendido, através dos resultados das inserções em árvores binária de busca e AVL, que há uma clara diferença no desempenho entre os dois tipos de estrutura.

Na BB, os tempos de inserção foram significativamente menores, especialmente nas ordens crescente e decrescente. A estrutura de uma árvore binária de busca simples, onde não há necessidade de balanceamento, facilita a inserção dos elementos, permitindo que o processo aconteça de forma mais rápida. No entanto, esse desempenho pode ser prejudicado em casos de inserção sequencial ou quando os elementos são inseridos de forma ordenada, o que pode resultar em uma árvore degenerada, semelhante a uma lista encadeada.

Por outro lado, a árvore AVL, que requer o balanceamento a cada inserção, demonstrou tempos de execução mais altos, especialmente para inserções em ordem aleatória. O balanceamento contínuo, necessário para garantir a propriedade de altura balanceada da árvore, implica em cálculos adicionais, como a verificação de fatores de balanceamento e, quando necessário, a realização de rotações. Apesar desse custo extra, a árvore AVL oferece uma estrutura mais balanceada, o que resulta em um desempenho mais eficiente em cenários onde as operações de busca e remoção são frequentes. Embora o tempo de inserção tenha sido maior na AVL, sua eficiência a longo prazo em termos de manutenção da altura equilibrada pode justificar seu uso em aplicações que requerem um alto volume de operações de busca e inserção.

O propósito do experimento, uma vez que é o desenvolvimento do programa e análise das estruturas, foi alcançado.

6 Anexos

Seguem as tabelas de cada uma das inserções.

Tabela 4 – Tempos de inserção dos 30 artistas (ordem aleatória) - Binária de Busca

Inserção	**Tempo (s)**	**Tempo (µs)**
1	0.0000040000	4.00
2	0.0000050000	5.00
3	0.0000030000	3.00
4	0.0000040000	4.00
5	0.0000040000	4.00
6	0.0000040000	4.00
7	0.0000030000	3.00
8	0.0000030000	3.00
9	0.0000040000	4.00
10	0.0000040000	4.00
11	0.0000030000	3.00
12	0.0000030000	3.00
13	0.0000040000	4.00
14	0.0000040000	4.00
15	0.0000030000	3.00
16	0.0000060000	6.00
17	0.0000040000	4.00
18	0.0000030000	3.00
19	0.0000030000	3.00
20	0.0000040000	4.00
21	0.0000030000	3.00
22	0.0000050000	5.00
23	0.0000030000	3.00
24	0.0000050000	5.00
25	0.0000030000	3.00
26	0.0000040000	4.00
27	0.0000040000	4.00
28	0.0000030000	3.00
29	0.0000050000	5.00
30	0.0000040000	4.00
Média	**0.0000038000**	**3.80**

Tabela 5 – Tempos de inserção dos 30 artistas (ordem aleatória) - AVL

Inserção	**Tempo (s)**	**Tempo (µs)**
1	0.0219600000	21960.00
2	0.0168910000	16891.00
3	0.0200340000	20034.00
4	0.0182800000	18280.00
5	0.0181730000	18173.00
6	0.0200630000	20063.00
7	0.0201120000	20112.00
8	0.0241880000	24188.00
9	0.0195030000	19503.00
10	0.0269410000	26941.00
11	0.0212590000	21259.00
12	0.0161340000	16134.00
13	0.0193680000	19368.00
14	0.0166680000	16668.00
15	0.0210630000	21063.00
16	0.0200330000	20033.00
17	0.0196650000	19665.00
18	0.0194730000	19473.00
19	0.0168540000	16854.00
20	0.0208460000	20846.00
21	0.0195230000	19523.00
22	0.0190800000	19080.00
23	0.0169940000	16994.00
24	0.0218180000	21818.00
25	0.0185250000	18525.00
26	0.0164180000	16418.00
27	0.0161740000	16174.00
28	0.0198180000	19818.00
29	0.0218810000	21881.00
30	0.0217390000	21739.00
Média	**0.0196492667**	**19649.27**

Tabela 6 – Tempos de inserção dos 30 artistas (ordem crescente) - Binária de Busca

Inserção	**Tempo (s)**	**Tempo (µs)**
1	0.0000040000	4.00
2	0.0000020000	2.00
3	0.0000020000	2.00
4	0.0000020000	2.00
5	0.0000020000	2.00
6	0.0000020000	2.00
7	0.0000030000	3.00
8	0.0000020000	2.00
9	0.0000020000	2.00
10	0.0000020000	2.00
11	0.0000020000	2.00
12	0.0000020000	2.00
13	0.0000020000	2.00
14	0.0000030000	3.00
15	0.0000020000	2.00
16	0.0000020000	2.00
17	0.0000020000	2.00
18	0.0000030000	3.00
19	0.0000020000	2.00
20	0.0000020000	2.00
21	0.0000030000	3.00
22	0.0000030000	3.00
23	0.0000030000	3.00
24	0.0000020000	2.00
25	0.0000020000	2.00
26	0.0000020000	2.00
27	0.0000020000	2.00
28	0.0000020000	2.00
29	0.0000020000	2.00
30	0.0000020000	2.00
Média	**0.0000022667**	**2.27**

Tabela 7 – Tempos de inserção dos 30 artistas (ordem crescente) - AVL

Inserção	**Tempo (s)**	**Tempo (µs)**
1	0.004973	4973
2	0.005422	5422
3	0.005954	5954
4	0.006698	6698
5	0.005695	5695
6	0.007390	7390
7	0.005450	5450
8	0.007000	7000
9	0.005405	5405
10	0.005427	5427
11	0.006618	6618
12	0.005508	5508
13	0.007264	7264
14	0.005222	5222
15	0.006789	6789
16	0.006876	6876
17	0.005091	5091
18	0.006659	6659
19	0.004989	4989
20	0.005687	5687
21	0.005750	5750
22	0.007091	7091
23	0.006064	6064
24	0.006570	6570
25	0.005242	5242
26	0.006909	6909
27	0.006073	6073
28	0.005401	5401
29	0.006085	6085
30	0.006671	6671
Média	**0.0060657667**	**6065.77**

Tabela 8 – Tempos de inserção dos 30 artistas (ordem decrescente) - Binária de Busca

Inserção	**Tempo (s)**	**Tempo (µs)**
1	0.0000020000	2.00
2	0.0000010000	1.00
3	0.0000000000	0.00
4	0.0000000000	0.00
5	0.0000000000	0.00
6	0.0000000000	0.00
7	0.0000010000	1.00
8	0.0000010000	1.00
9	0.0000010000	1.00
10	0.0000010000	1.00
11	0.0000000000	0.00
12	0.0000000000	0.00
13	0.0000010000	1.00
14	0.0000010000	1.00
15	0.0000010000	1.00
16	0.0000010000	1.00
17	0.0000010000	1.00
18	0.0000010000	1.00
19	0.0000010000	1.00
20	0.0000010000	1.00
21	0.0000010000	1.00
22	0.0000010000	1.00
23	0.0000010000	1.00
24	0.0000010000	1.00
25	0.0000010000	1.00
26	0.0000010000	1.00
27	0.0000010000	1.00
28	0.0000000000	0.00
29	0.0000010000	1.00
30	0.0000010000	1.00
Média	**0.0000008000**	**0.80**

Tabela 9 – Tempos de inserção dos 30 artistas (ordem decrescente) - AVL

Inserção	**Tempo (s)**	**Tempo (µs)**
1	0.0041480000	4148.00
2	0.0047650000	4765.00
3	0.0062450000	6245.00
4	0.0077180000	7718.00
5	0.0072060000	7206.00
6	0.0049780000	4978.00
7	0.0054810000	5481.00
8	0.0068520000	6852.00
9	0.0047010000	4701.00
10	0.0056190000	5619.00
11	0.0069310000	6931.00
12	0.0047980000	4798.00
13	0.0058320000	5832.00
14	0.0068550000	6855.00
15	0.0046870000	4687.00
16	0.0055750000	5575.00
17	0.0067380000	6738.00
18	0.0047410000	4741.00
19	0.0052570000	5257.00
20	0.0068420000	6842.00
21	0.0045410000	4541.00
22	0.0053110000	5311.00
23	0.0063970000	6397.00
24	0.0052880000	5288.00
25	0.0055080000	5508.00
26	0.0064380000	6438.00
27	0.0048690000	4869.00
28	0.0059240000	5924.00
29	0.0052360000	5236.00
30	0.0047120000	4712.00
Média	**0.0056731000**	**5673.10**