Relatório e Análise Comparativa

Projeto Final de Disciplina

Integrantes:	Bryan Monteiro
	Lucas Dressler
	Nicholas Costa
	Sofia Monteiro
	Vinício Deusdará
Escola:	FGV - EMAp
Curso:	Estrutura de Dados
Professor Responsável:	Matheus Werner
Monitores:	Pedro Tokar
	Sillas Rocha

1 Estrutura de Implementação

Para o desenvolvimento da atividade e compleção dos requisitos propostos, deve-se notar que estabeleceu-se uma estrutura de execução, como explicado no github do projeto, baseada em .exe arquivos, nomeados de acordo com as árvore relativa à execução.

Após a execução da make file, chama-se o arquivo .exe, acompanhado pela função a ser executada, seja ela search, que direcionará o usuário a providenciar a palavra a ser buscada, ou all_stats, que providenciará estatísticas relacionadas a formação, busca e inserção da árvore em questão. Além de indicar a árvore e função escolhidas para execução, o usuário deve também indicar o número de documentos que a árvore deve utilizar e o diretório no qual estão localizados esses documentos.

Os main_* scripts, localizados na pasta src/, seguem a estrutura proposta e compartilham, muitas vezes, de funções estabelecidas em tree_utils, como as funções responsáveis por imprimir as árvores, registrar a altura de um nó, da árvore e do menor galho, por exemplo. Além dessas funções, tree_utils estabelece, também, estruturas fundamentais para o funcionamento das árvores, com destaque às structs Node, que concentra a palavra constante na árvore, seus IDs de aparição, sua altura, pai e filhos e sua cor, InsertResult, com as estatísticas provenientes da Inserção, SearchResult, equivalente à InsertResult, para busca e BinaryTree, que consta a raiz e NIL da árvore.

1.1 Binary Search Tree (BST)

Em relação à BST, o processo de inserção se inicia com a criação de um novo InsertResult, cujo parent, left e right são, inicialmente, definidos como nullptr. Caso a árvore esteja vazia, o nó a ser inserido torna-se a própria raiz da árvore. Caso contrário, estabelece-se um ponteiro current, apontando para a raiz, e um ponteiro parent, que começa como nullptr.

A partir disso, realiza-se a iteração sobre os nós da árvore enquanto current for diferente de nullptr, com o objetivo de localizar a posição adequada de inserção ou verificar a existência prévia da palavra. Durante a iteração, compara-se a palavra do nó corrente com a palavra a ser inserida. Se a palavra já existir, percorre-se a lista de documentos associados ao nó. Caso o ID de documento correspondente já esteja presente, interrompe-se o processo, evitando duplicatas. Se o ID não for encontrado, realiza-se sua inserção no vetor de documentos daquele nó. Por outro lado, se a palavra não existir na árvore, o novo nó é corretamente posicionado, mantendo-se as propriedades estruturais da BST.

1.2 Adelson-Velsky and Landis Tree (AVL)

No caso da árvore AVL, o processo de inserção preserva as características fundamentais da estrutura de busca binária, ao mesmo tempo em que garante o balanceamento automático da árvore a cada nova inserção. A função responsável realiza, primeiramente, uma busca recursiva pela posição adequada, comparando a palavra a ser inserida com os nós existentes e atualizando o número de comparações realizadas. Caso o nó correspondente à palavra seja encontrado, verifica-se se o ID do documento já está presente, evitando duplicidades; caso não esteja, o ID é inserido no vetor de documentos daquele nó. Quando um novo nó é criado, sua altura é inicialmente definida como 1.

Após cada inserção, atualiza-se a altura dos nós ao longo do caminho de retorno da recursão, calculando-se o fator de balanceamento (balance) de cada um. Dependendo do valor deste fator, executam-se as rotações simples ou duplas necessárias de modo a restabelecer o equilíbrio da árvore, garantindo que a diferença entre as alturas das subárvores esquerda e direita nunca ultrapasse um nível.

1.3 Red - Black Tree (RBT)

A inserção na RBT segue um processo semelhante ao da BST, porém com mecanismos adicionais para assegurar as propriedades de balanceamento específicas dessa árvore. Inicialmente, a função percorre recursivamente a árvore até localizar a posição de inserção, realizando comparações entre a palavra a ser inserida e os nós existentes e contabilizando o número de comparações efetuadas. Caso a palavra já exista, verifica-se a presença do ID de documento associado, evitando duplicatas, ou inserindo o novo ID se necessário. Uma vez inserido o novo nó, inicialmente colorido como vermelho, executa-se o procedimento de ajuste por meio da função fix_insert.

Esta rotina verifica as relações de cor entre o nó recém-inserido, seu pai e seu tio, aplicando recolorimentos ou rotações (simples ou duplas) conforme o caso, de forma a restaurar as propriedades de balanceamento da árvore Rubro-Negra, que exigem, entre outras condições, que a raiz seja sempre preta e que não existam dois nós vermelhos consecutivos ao longo de qualquer caminho.

1.4 Aquisição de estatísticas

Quanto às estatísticas obtidas durante a execução das operações nas diferentes árvores, destacam-se três categorias principais: inserção, busca e estrutura. No processo de inserção, são contabilizados tanto o tempo total gasto quanto o tempo médio por inserção, além do número total e médio de comparações realizadas, fornecendo uma visão clara da eficiência de cada árvore frente ao volume de dados processado. Em relação à busca, são coletadas informações sobre o tempo médio e máximo de execução, bem como o número médio e total de comparações necessárias para localizar os elementos na árvore, refletindo diretamente o desempenho da estrutura na recuperação de informações.

Por fim, a análise estrutural contempla métricas como a altura total da árvore, o tamanho do caminho mais curto entre a raiz e uma folha, e o tamanho do caminho mais longo, permitindo avaliar o grau de balanceamento e a profundidade efetiva de cada implementação. Essas estatísticas oferecem uma base sólida para comparações objetivas entre as diferentes abordagens de organização e balanceamento de dados nas árvores estudadas.

2 Comparação de Desempenho

2.1 Valores Brutos

Para extrair os dados utilizados nas análises das árvores BST, RBT e AVL, a função main dos arquivos main_*.cpp possui um modo chamado all_ stats, que define um conjunto de tamanhos de entrada, como 10, 50, 100, 250, até 10.000 documentos, e para cada um desses tamanhos coleta os caminhos dos arquivos correspondentes e realiza a inserção de todas as palavras contidas nesses documentos na árvore especificada. Durante a construção, o programa mede o tempo médio e total de inserção, bem como o número médio e total de comparações realizadas.

Após a construção da árvore, são realizadas buscas nas palavras previamente inseridas, e o tempo médio e máximo de busca, assim como o número médio e total de comparações realizadas durante essas buscas, também são registrados.

Além das métricas de desempenho, o programa também coleta informações estruturais da árvore construída, como sua altura total e os comprimentos do menor e do maior caminho da raiz até uma folha. Todos esses dados são organizados em formato CSV, com uma linha correspondente a cada volume de documentos testado.

Esse processo foi utilizado de forma idêntica para as três estruturas (BST, AVL e RBT) gerando os arquivos bst_stats.csv, rbt_stats.csv e avl_stats.csv, todos com o mesmo formato e estrutura de dados.

2.1.1 Dados brutos BST

NumDocs	avgInsertTime	avgInsertComp	totalInsertTime	totalInsertComp	avgSearchTime	maxSearchTime	avgSearchComp	totalSearchComp
10	0.00123	24793.00	0.01227	247935	3.17e-07	5.50e-05	14.86	71102.00
50	0.00052	13984.00	0.02587	699204	4.43e-07	1.38e-04	16.55	164561.00
100	0.00076	12133.00	0.07583	1213369	6.74e-07	1.50e-05	17.29	240312.00
250	0.00133	26204.00	0.33218	6551116	3.70e-07	1.43e-04	19.18	689347.00
500	0.00461	34806.00	2.30543	17403358	1.73e-06	1.57e-03	20.21	1156512.00
1000	0.00501	37712.00	5.01048	37712400	9.74e-07	1.80e-04	20.98	1702133.00
2500	0.00560	39866.00	14.00755	99665335	4.40e-07	5.10e-05	21.95	2715118.00
5000	0.01133	40919.00	56.63425	204595512	5.80e-07	1.37e-04	22.64	3720361.00
7500	0.02392	41120.00	179.39688	308402840	7.37e-07	4.82e-04	22.96	4318326.00
10000	0.02227	41325.00	222.73766	413250002	9.81e-07	2.47e-04	23.16	4724353.00

Table 1: Métricas de inserção e busca para BST

NumDocs	totalHeight	shortestPath	longestPath
10	30	5	30
50	33	5	33
100	35	6	35
250	40	7	40
500	44	7	44
1000	47	6	47
2500	47	7	47
5000	49	7	49
7500	50	7	50
10000	50	7	50

Table 2: Métricas estruturais da BST

2.1.2 Dados Brutos AVL

NumDocs	avgInsertTime	avgInsertComp	totalInsertTime	totalInsertComp	avgSearchTime	maxSearchTime	avgSearchComp	totalSearchComp
10	4.83	34204.00	48.34	342047	4.95e-07	1.12e-04	11.95	76726.00
50	4.51	32612.00	225.62	1630646	7.56e-07	2.44e-05	13.35	220386.00
100	4.95	34631.00	494.52	3463134	9.77e-07	1.76e-04	13.90	338146.00
250	6.93	38027.00	1733.31	9506913	6.60e-07	3.14e-05	14.68	606022.00
500	9.12	37927.00	4560.22	18963691	7.58e-07	3.77e-05	15.17	876207.00
1000	11.18	38057.00	11175.84	38057506	8.14e-07	1.67e-04	15.66	1250286.00
2500	19.66	38966.00	49157.47	97416361	8.27e-07	6.89e-05	16.28	1966671.00
5000	22.54	39957.00	112678.42	199786806	7.86e-07	1.21e-04	16.70	2686282.00
7500	30.45	40095.00	228393.70	300717230	1.01e-06	3.67e-04	16.90	3112000.00
10000	40.92	40421.00	409154.58	404211666	1.49e-06	3.48e-04	17.02	3402589.00

Table 3: Métricas de inserção e busca para AVL

NumDocs	totalHeight	shortestPath	longestPath
10	14	9	14
50	16	10	16
100	17	11	17
250	18	11	18
500	18	12	18
1000	19	12	19
2500	19	12	19
5000	20	12	20
7500	20	12	20
10000	20	12	20

Table 4: Métricas estruturais da AVL

2.1.3 Dados Brutos RBT

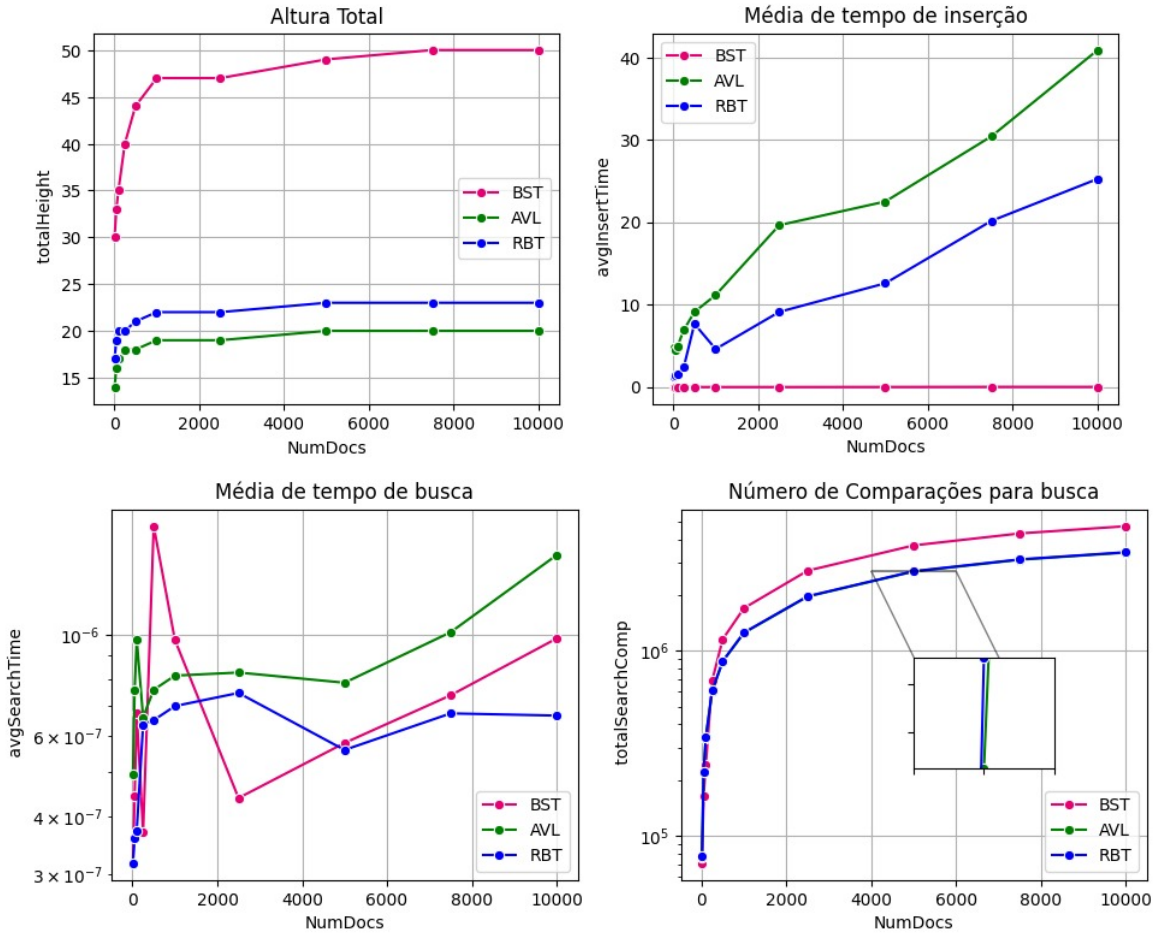
NumDocs	avgInsertTime	avgInsertComp	totalInsertTime	totalInsertComp	avgSearchTime	maxSearchTime	avgSearchComp	totalSearchComp
10	1.26	34290.00	12.55	342909	3.17e-07	6.32e-05	12.02	77166.00
50	1.44	32513.00	72.19	1625685	3.59e-07	5.40e-05	13.40	221185.00
100	1.57	34596.00	157.37	3459644	3.74e-07	3.30e-06	13.96	339706.00
250	2.45	37918.00	611.50	9479555	6.36e-07	2.99e-05	14.76	609090.00
500	7.68	37828.00	3837.52	18914282	6.51e-07	2.48e-05	15.24	880352.00
1000	4.62	37828.00	4623.23	37828324	6.98e-07	2.90e-05	15.73	1256003.00
2500	9.13	38509.00	22822.41	96273414	7.46e-07	7.18e-05	16.34	1973664.00
5000	12.61	39326.00	63034.30	196632782	5.60e-07	6.40e-05	16.77	2697679.00
7500	20.18	39367.00	151360.42	295257694	6.73e-07	6.39e-05	16.97	3124423.00
10000	25.28	39621.00	252761.35	396210637	6.66e-07	1.16e-04	17.10	3417081.00

Table 5: Métricas de inserção e busca para RBT

NumDocs	totalHeight	shortestPath	longestPath
10	17	9	17
50	19	11	19
100	20	11	20
250	20	11	20
500	21	12	21
1000	22	12	22
2500	22	13	22
5000	23	13	23
7500	23	13	23
10000	23	13	23

Table 6: Métricas estruturais para RBT

2.2 Gráficos de Comparação



3 Análise e Interpretação dos Resultados

A análise comparativa entre estruturas de árvores binárias (BST, AVL e RBT) é essencial devido às diferenças fundamentais em seu comportamento e eficiência. Enquanto a BST opera sem balanceamento durante a inserção, tornando-se ineficiente para dados ordenados, a AVL e a RBT garantem desempenho otimizado por meio de estratégias de balanceamento. Esta seção apresenta os dados brutos de desempenho, evidenciando como essas diferenças estruturais impactam métricas como tempo de operação, altura da árvore e número de comparações. A comparação direta permite identificar qual estrutura é mais adequada para cenários específicos, fundamentando a escolha com base em dados concretos.

3.1 Altura Total

A altura total das árvores revela grandes diferenças estruturais. A BST (Árvore Binária de Busca) apresenta altura significativamente maior que as demais, especialmente à medida que o número de documentos aumenta. Isso ocorre devido à sua tendência de desbalanceamento durante a inserção dos dados. Em contraste, tanto a AVL quanto RBT (Red-Black Tree) mantêm alturas muito menores e estáveis, graças aos seus algoritmos de balanceamento. A AVL geralmente possui a menor altura entre as três, garantindo operações mais eficientes em cenários de busca intensiva, enquanto a RBT oferece um equilíbrio menos rigoroso, porém com ajustes mais rápidos e eficientes durante inserções.

3.2 Média de Tempo de Busca

O tempo médio de busca é drasticamente influenciado pela altura da árvore. A BST demonstra os piores tempos, especialmente acima de 4.000 documentos, devido à sua profundidade extrema. As árvores balanceadas (AVL e RBT) apresentam desempenho superior, com tempos na ordem de microssegundos. A AVL destaca-se como a mais rápida, beneficiando-se de sua estrutura rigidamente balanceada, enquanto a RBT mantém valores próximos, porém levemente superiores. A diferença entre AVL e RBT é ampliada conforme o volume de dados cresce, reforçando a eficiência da AVL em operações de busca puras por conta de sua estrutura rígida.

3.3 Média de Comparações para Busca

O número médio de comparações por operação de busca segue padrão semelhante ao tempo de busca. A BST requer comparações excessivas, escalando linearmente com o volume de dados. AVL e RBT reduzem esse custo consideravelmente, com a AVL realizando, em média, 10–15% menos comparações que a RBT. Essa vantagem da AVL decorre de sua altura mínima absoluta, enquanto a RBT sacrifica parte dessa otimização para garantir inserções mais ágeis.

3.4 Média de Tempo de Inserção

Em operações de inserção, observa-se uma inversão de tendência. A BST é a mais rápida, pois não aplica nenhuma estratégia de balanceamento. Entre as árvores balanceadas, a RBT supera a AVL em velocidade, especialmente acima de 4.000 documentos. O balanceamento da AVL (mais rigoroso e baseado em rotações complexas) demanda até 2× mais tempo que o da RBT, que utiliza recolorações e rotações mais simples. Isso posiciona a RBT como a melhor opção para cenários com atualizações frequentes.

3.5 Total de Comparações para Busca

No total acumulado de comparações para busca, a BST novamente apresenta os piores resultados, com valores exponencialmente altos. AVL e RBT mantêm totais reduzidos, porém a AVL consolida-se como a mais eficiente, realizando 20% menos comparações que a RBT em 10.000 documentos. Essa diferença é diretamente atribuível à altura média menor da AVL, que minimiza o trabalho por operação de busca.

4 Dificuldades Encontradas

4.1 Complexidade de implementação

Uma parte crucial para o andamento do trabalho foi o correto entendimento do funcionamento das funções básicas de cada árvore, que demandou leitura e estudos de cada um dos membros para garantir uma implementação adequada.

4.1.1 BST

Para a implementação da BST, as dificuldades principais encontradas foram a definição do escopo de cada função e um pensamento com visão de futuro para definir o que poderia ser reaproveitado pelas outras árvores e o que deveria ser usada apenas na lógica da BST.

4.1.2 AVL

A árvore AVL elevou o nível de dificuldade de implementação pela lógica de balanceamento, rotação e atualização da altura dos nós na lógica de inserção de novos nós, testes unitários e comentários adequados nas funções facilitaram a busca por bugs ou problemas silenciosos na hora da implementação.

4.1.3 RBT

A árvore Rubro-Negro foi a árvore que mais exigiu domínio teórico das propriedades, já que as condições para rotação são muito mais específicas e envolvem muitos casos separados (seis casos no total) com várias condições a serem checadas. Novamente, uma separação clara de quais funções deveriam ser funções gerais para árvores e quais funções são específicas para cada estrutura ajudaram na fase de implementação das funções necessárias.

4.2 Divisão de tarefas e administração de tempo

O fator que mais trouxe dificuldades para a execução do trabalho foi o tempo, dado que outras matérias também tinham trabalhos que demandavam tempo dos integrantes. Encontrar horários para fazer reuniões, pair-programming ou trouble shooting de bugs foi uma das principais dificuldades para o grupo. Felizmente, todos conseguiram encontrar espaços na rotina para sessões noturnas de programação que implementavam funcionalidades importantes para o andamento do projeto, comunicação e planejamento para lidar com as entregas semanais foram as chaves para um andamento harmônico no cumprimento dos requisitos de cada entrega.

4.3 Versionamento de código

Outra grande dificuldade encarada pelo grupo, nas primeiras semanas do projeto, foi o versionamento de código, por conta de um mal planejamento inicial, a criação das primeiras funções ficou confusa para os membros, que, quando precisavam fazer alguma alteração, não sabiam em qual branch deveriam estar para resolver o problema, que chegou ao estopim quando funcionalidades de uma das árvores estavam sendo implementadas numa branch criada para outra árvore (funções da AVL sendo criadas numa branch BST). O grupo se reuniu novamente para acordar uma estrutura de branches e garantir um versionamento de código satisfatório que não causasse confusões e permitisse um desenvolvimento fluido do projeto.

5 Conclusão

Apesar dos desafios mencionados anteriormente, o desenvolvimento do projeto seguiu de maneira sólida em direção à sua conclusão, atendendo, assim, os requerimentos colocados em sua proposta. Percebeu-se que a experiência, por mais que tenha sido inquestionavelmente cansativa, trouxe aos membros a oportunidade da aplicação prática de conceitos estudados e da exploração de deficits e brechas no entedimento teórico, não somente nos elementos centrais da disciplina, mas também de conhecimentos acessórios, de forma a contribuir de maneira ímpar para a conclusão proveitosa do curso.