Relatório do Projeto: Índice Invertido e Análise Comparativa de Estruturas de Dados

Bruno Ferreira Salvi
Henrique Coelho Beltrão
Henrique Gabriel Gasparelo
José Thevez Gomes Guedes
Luiz Eduardo Bravin

Professor orientador: Matheus Telles Werner.

Repositório: `https://github.com/riqueu/a2-ed-2025/`

Abstract

Este trabalho detalha a implementação e a avaliação de desempenho de um índice invertido utilizando três estruturas de dados baseadas em árvores: a Árvore Binária de Busca (BST), a Árvore AVL e a Árvore Rubro-Negra (RBT). O objetivo foi comparar a eficiência de cada estrutura em operações de inserção e busca, aplicadas a um corpus de aproximadamente 10.000 documentos de texto. As métricas analisadas, como tamanhos dos galhos, número de comparações e altura da árvore, demonstraram a superioridade das árvores balanceadas (AVL e RBT) sobre a BST, que se mostrou suscetível à degeneração. Os resultados indicaram que a RBT ofereceu o melhor tempo de inserção e um tempo de busca tão bom quanto a AVL, que se destacou por buscas ligeiramente mais rápidas e estáveis. Conclui-se que a RBT representa a escolha mais equilibrada pois além de ser a melhor para operações de inserção, apresenta um desempenho satisfatório na funções de busca.

Contents

1 Introdução

Neste projeto, visa-se implementar um índice invertido, ou seja, um mapeamento de palavras e os documentos em que elas aparecem, e alocá-lo em uma estrutura de dados que otimize a busca no mesmo. Para isto, serão utilizadas as estruturas denominadas de árvores, que para o escopo deste projeto serão: **Árvore Binária de Busca (BST)**, **Árvore AVL (AVL)** e **Árvore Rubro-Negra (RBT)**, sendo as três árvores binárias de busca, entretanto com diferentes abordagens que serão exploradas nesse projeto.

Como árvores binárias de busca, em qualquer uma das três árvores, os nós possuem no máximo 2 filhos, onde os descendentes à esquerda são anteriores, em ordem, nesse caso alfabética, ao pai e os à direita são posteriores. Desta forma, no caso ideal, a altura da árvore é proporcional a $\log(n)$, possibilitando operações mais rápidas, tanto de busca quanto inserção. Logo, a diferença fundamental entre as árvores é na forma como as propriedades modificam as funções de inserção, o que impacta diretamente na organização dos nós na árvore.

Na BST, não existe nenhuma regra adicional, além das já estabelecidas para uma árvore binária de busca, o que, portanto, possibilita uma fácil implementação das funções de inserção e busca. Apesar disso, a falta de regras que garantam o balanceamento permitem, por exemplo, a existência de casos degenerados, onde a árvore se torna uma lista encadeada e sua performance é $O(n)$ para todas as operações. Já para a AVL, são instauradas regras de balanceamento, onde, para cada nó, a diferença de altura entre suas subárvores esquerda e direita (fator de balanceamento) é no máximo 1, isso é garantido por meio de funções de rotação que alteram as alturas das subárvores. Por conta dessas restrições, as funções de inserção fazem mais alterações na árvore para boa parte dos nós inseridos. Entretanto, o balaceamento garante operações com complexidade $O(\log(n))$, no pior caso. A fim de diminuir o grau de restrição da AVL, a RBT mantém o balanceamento através de um conjunto de regras que envolvem colorir cada nó de vermelho ou preto. Essas regras garantem que o caminho mais longo da raiz a qualquer folha não seja mais que o dobro do caminho mais curto, permitindo um custo operacional de $O(\log(n))$, no pior caso, com uma menor rigidez que a AVL. Mesmo assim, por conta deste comportamento, a RBT pode apresentar uma altura ligeiramente maior que AVL, em determinadas situações.

Portanto, o objetivo principal é analisar e comparar o desempenho dessas estruturas em operações fundamentais como inserção e busca, considerando diferentes volumes de dados.

2 Desenvolvimento

2.1 Metodologia de funcionamento

A comparação entre as três estruturas de dados foi realizada seguindo os passos abaixo:

1. **Corpus de Documentos:** Foi utilizado um conjunto de aproximadamente 10.000 documentos de texto (`.txt`) com palavras já pré-processadas, com todas minúsculas e sem pontuações.

2. **Construção do Índice:** As palavras extraídas foram inseridas como `Node` em cada uma das três estruturas de árvore (BST, AVL, RBT). Para cada palavra, o ID do documento em que ela apareceu foi adicionado à lista de documentos associada ao nó da palavra, conforme a estrutura abaixo:

```
struct Node {
    std::string word;
    std::vector<int> documentIds;
    Node* parent;
    Node* left;
    Node* right;
    int height;    // usado na AVL
    int isRed;     // usado na RBT
};
```

3. **Coleta de Métricas:** Os métricas abaixo foram calculadas para diferentes subconjuntos do corpus, variando o número de documentos processados de 10 a 10.000 documentos, de 10 em 10.
   - Tempo de inserção (média, total);
   - Tempo de busca de palavras (médio, máximo);
   - Número de comparações por operação;
   - Altura da árvore;
   - Tamanho dos galhos (menor e maior caminho);
   - Número de nós;
   - Tamanho da árvore.
4. **Ferramentas:** A implementação das árvores e coleta de métricas foram realizadas por programas em C++. Os gráficos e análises estatísticas foram feitos utilizando a biblioteca Matplotlib de Python.

2.2 Código

Agora segue detalhamento das principais abordagens utilizadas no código do projeto.

Leitura dos documentos (`data.cpp`): Para a extração das palavras dos documentos, foram processados os arquivos do diretório e suas palavras foram colocadas em um vetor, nesse momento, é verificada a existência de palavras duplicadas em um arquivo, percorrendo esse vetor, e apenas os elementos únicos são inseridos. Vale salientar que a ordem de leitura dos documentos pelo iterador do diretório não necessariamente representa a ordem numérica habitual, deste modo, documentos de id superiores podem ser lidos antes dos arquivos com id inferiores.

Implementação da árvores (`bst.cpp`, `avl.cpp`, `rbt.cpp`): Para a implementação das árvores, foram desenvolvidas funções de criação, inserção, busca e deleção das mesmas, além da elaboração das funções que conservam as propriedades de balanceamento da AVL e RBT. Para a AVL, foi estipulado que a altura de um nó vazio é -1, consequentemente, a altura das folhas é 0, e foi considerado, como fator de balaceamento, a diferença entre as alturas das subárvores à direita e à esquerda.

Implementação dos testes (`test_bst.cpp`, `test_avl.cpp`, `test_rbt.cpp`): Para implementação dos testes, foram verificadas a consistência das regras de cada árvore, as funções de inserção e de busca.

Geração e coleta de estatísticas (`tree_utils.cpp`, `tree_stats.cpp`, `export_stats.cpp`): Para coleta das métricas, no arquivo `tree_utils.cpp`, foram implementadas funções que geram as estatísticas referentes às árvores. Estas métricas são utilizadas no CLI de estatísticas e na criação de um arquivo CSV com os dados obtidos. É Importante destacar que o arquivo CSV é somente gerado pelo `tree_stats.cpp`, enquanto a CLI de estatísticas gera apenas um overview simplificado.

Arquivos principais (`main_bst.cpp`, `main_avl.cpp`, `main_rbt.cpp`): Para utilização do código por outros usuários, no arquivo main de cada árvore, foram desenvolvidos comandos CLI para busca de uma determinada palavra, geração de estatísticas e visualização da árvore (comando `search`, `stats`, e `print`, respectivamente). Como mencionado no parágrafo anterior, as estatísticas geradas por esses arquivos são apenas impressas no terminal, para gerar o arquivo CSV, usamos `tree_stats.cpp`, que será melhor explicado no próximo tópico.

2.3 Estatísticas

Assim como supracitado, o documento `tree_utils.cpp` contém funções que coletam as estatísticas com base em parâmetros, como o tipo da árvore e a quantidade de documentos que serão inseridos. A partir dessas funções, são coletadas estatísticas como:

- **Número total de documentos inseridos** (`N_docs`);
- **Número médio de comparações na inserção** (`NumComparisonsInsertionMean`);
- **Número total de comparações na inserção** (`NumComparisonsInsertionTotal`);
- **Tempo médio de execução das inserções (ms)** (`ExecutionTimeInsertionMean`);
- **Tempo total de execução das inserções (ms)** (`ExecutionTimeInsertionTotal`);
- **Número médio de comparações na busca** (`NumComparisonsSearchMean`);
- **Número máximo de comparações na busca** (`NumComparisonsSearchMax`);
- **Tempo máximo de execução das buscas (ms)** (`ExecutionTimeSearchMax`);
- **Tempo médio de execução das buscas (ms)** (`ExecutionTimeSearchMean`);
- **Altura final da árvore após as inserções** (`TreeHeight`);
- **Comprimento do menor galho** (`MinBranch`);
- **Comprimento do maior galho** (`MaxBranch`);
- **Número total de nós na árvore** (`NumNodes`);
- **Tamanho total ocupado pela árvore em memória (bytes)** (`TreeSizeBytes`).

Vale ressaltar que, neste contexto, são considerados como número de comparações a quantidade de nós que foram percorridos durante a operação, ou seja, é contabilizado cada comparação com um nó durante a inserção ou busca. Além disso, a fim de reduzir erros de medições de tempo nas estáticas de tempo de busca, para as árvores que tem 800 documentos ou menos, a busca por cada palavra é realizada 50 vezes, e então é calculado a médias desses resultados, mitigando assim, as oscilações causas pelo computador nos tempos calculados.

Essas métricas permitem a análise comparativa dos diferentes tipos de implementação de árvores, possibilitando, por exemplo, a comparação do impacto do número de documentos inseridos na altura de cada uma das árvores.

No documento `tree_stats.cpp`, utilizando as funções responsáveis pela coleta de estatísticas, são armazenadas as métricas obtidas de um tipo especificado de árvore, variando a quantidade de documentos inseridos. Essas estatísticas são então concatenadas em um arquivo CSV, o que viabiliza a plotagem dos dados e facilita a análise entre as diferentes estruturas de árvores. Vale destacar que o `tree_stats.cpp` deve ser executado separadamente para cada tipo de árvore, uma vez que o CSV gerado tem apenas as estatísticas referentes a um único tipo de árvore.

2.4 Divisão de Tarefas

2.4.1 Entrega 1

- **Bruno Ferreira Salvi:** Implementação da função de busca para BST e da CLI (Busca) para BST;
- **Henrique Coelho Beltrão:** Implementação de funções para estatísticas, ajustes na CLI para estatísticas e construção do Makefile;
- **Henrique Gabriel Gasparelo:** Implementação das funções de print da árvore, de inserção na BST e de destroy da BST;

- **José Thevez Gomes Guedes:** Implementação das funções para leitura dos arquivos e construção do índice invertido;
- **Luiz Eduardo Bravin:** Implementação dos teste unitários da BST e inicialização da redação do relatório.

2.4.2 Entrega 2

- **Bruno Ferreira Salvi:** Implementação da CLI para AVL e estatísticas extras;
- **Henrique Coelho Beltrão:** Implementação das funções da AVL, testes unitários da AVL, refatoração do código;
- **Henrique Gabriel Gasparelo:** Implementação das estatísticas para árvores;
- **José Thevez Gomes Guedes:** Implementação das estatísticas para árvores e transição para CSV;
- **Luiz Eduardo Bravin:** Implementação da análise comparativa das árvores, geração dos gráficos em Python e redação do relatório.

2.4.3 Entrega 3

- **Bruno Ferreira Salvi:** Revisão do relatório final, formulação das estatísticas extras e revisão geral do código e documentação;
- **Henrique Coelho Beltrão:** Revisão do relatório final, ajustes no Makefile para descompactar dados e reformatação do código;
- **Henrique Gabriel Gasparelo:** Implementação dos testes unitários da RBT e redação do relatório;
- **José Thevez Gomes Guedes:** Implementação das funções da RBT e redação do relatório;
- **Luiz Eduardo Bravin:** Implementação da CLI para RBT e redação do relatório.

3 Resultados e Discussões

Nesta seção, serão apresentados os resultados numéricos obtidos e uma discussão sobre o desempenho comparativo das estruturas.

3.1 Dados Brutos

Para análise dos resultados obtidos, foram criados arquivos CSV com as métricas calculadas, onde esses dados brutos podem ser acessados pelos links de acesso abaixo.

Tabela BST
Tabela AVL
Tabela RBT

3.1.1 Palavras Mais Frequentes por Tamanho

A tabela abaixo apresenta, para cada tamanho de palavra (de 1 a 20 letras), a palavra que aparece em mais documentos do corpus, juntamente com a quantidade de documentos em que ela ocorre. Isso evidencia a presença de termos comuns e recorrentes, muitos deles funcionais ou relacionados ao contexto dos textos analisados. Palavras curtas, como preposições e artigos, tendem a aparecer em praticamente todos os documentos, enquanto palavras mais longas são naturalmente mais raras.

Table 1: Palavras mais frequentes por tamanho.

Nº de Letras	**Palavra**	**Nº de Documentos**
1	a	9.999
2	in	10.000
3	and	9.999
4	with	9.950
5	which	9.647
6	during	8.326
7	between	7.647
8	category	9.825
9	including	6.935
10	references	9.670
11	development	3.353
12	particularly	2.646
13	international	3.093
14	administration	1.184
15	characteristics	900
16	responsibilities	293
17	disestablishments	138
18	telecommunications	83
19	counterintelligence	14
20	uncharacteristically	28

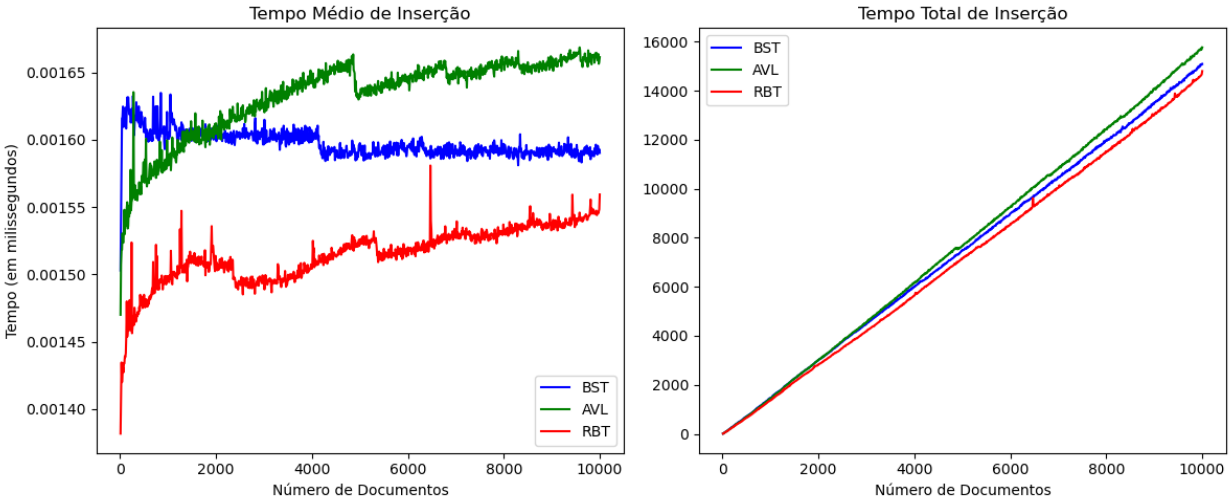
Esses dados reforçam a importância de considerar a frequência e o tamanho das palavras ao analisar índices invertidos, pois termos muito comuns podem impactar o desempenho das operações de busca e o balanceamento das árvores.

3.2 Gráficos e Estatísticas

A seguir, serão analisados graficamente os resultados obtidos, comparando o desempenho de cada árvore implementada em uma estatística específica.

3.2.1 Tempo de Inserção

Figure 1: Tempo de Inserção de Palavra

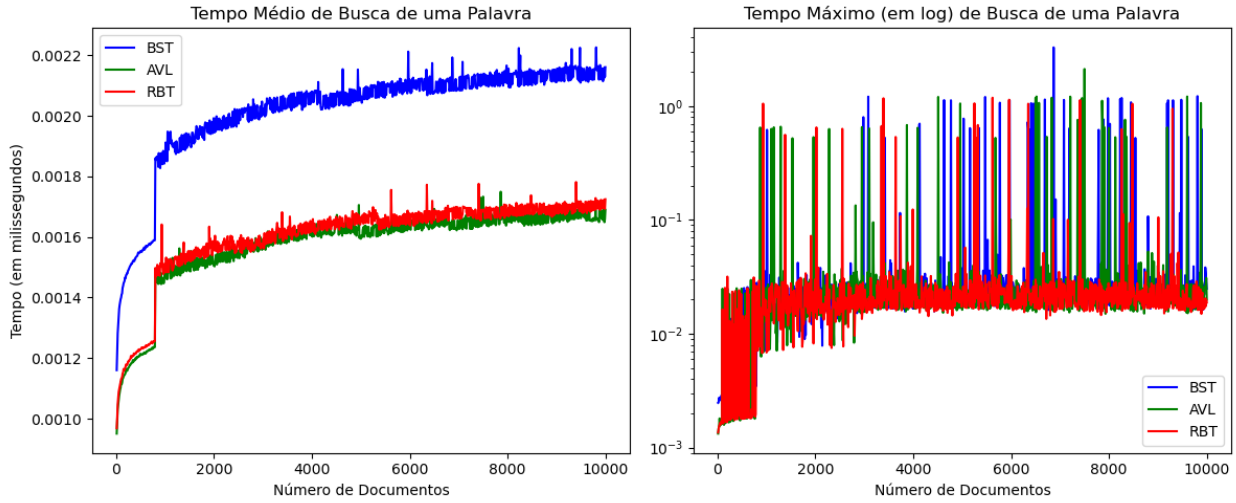


Fonte: Autoria própria.

Discussão: A análise do tempo médio de inserção revela que a RBT apresentou o melhor desempenho, sendo consistentemente mais rápida que as outras duas estruturas. Esse resultado era esperado e pode ser atribuído ao menor número de rotações necessárias para manter as propriedades, em comparação à AVL, e um maior balanceamento, em comparação à BST. O gráfico também torna explícito que a BST exige mais tempo de execução para o processo de inserção do que a RBT, entretanto, a AVL, que possui tempo de inserção menor que a BST para um número pequeno de documentos mostra-se menos eficiente que a mesma a partir de um determinado número de arquivos, este comportamento, apesar de ser diferente do esperado, que tanto a AVL quanto a RBT tivessem tempos de inserção inferiores à BST, pode ser explicado pelo alto custo operacional para realizar as funções de inserção na AVL, uma vez que frequentemente são necessárias rotações para manutenção do balanceamento. Já a análise do tempo total de inserção evidencia o mesmo comportamento, da RBT ser a mais eficiente no processo de inserção e a AVL ser a menos eficiente, devido as rotações frequentes. A mesma análise também revela que o tempo para inserção de todos os documentos cresce de maneira linear, para todas as árvores, o que apesar de inusitado, reflete o aumento do número total de palavras a medida que se aumenta o número de arquivos, aumentando assim também o tempo para inserção de novas palavras.

3.2.2 Tempo de Busca

Figure 2: Tempo de Busca de Palavra

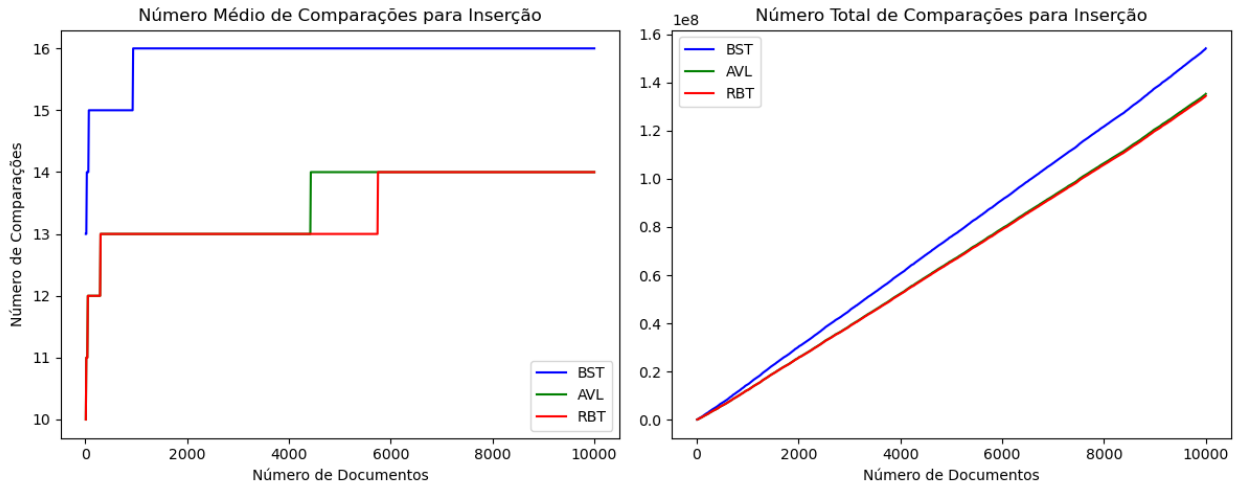


Fonte: Autoria própria.

Discussão: Na análise do tempo médio para busca, é possível notar que as árvores balanceadas, AVL e RBT, reduziram significativamente o tempo de busca em relação à BST. A AVL destacou-se com o menor tempo médio de busca, evidenciando que seu balanceamento rigoroso é vantajoso para consultas. A RBT apresentou desempenho muito próximo ao da AVL, sendo também uma excelente opção para buscas. A BST, por sua vez, foi a mais lenta, com picos de tempo de busca que refletem os problemas causados por seu desbalanceamento. Observando o crescimento no tempo para busca a medida que se aumenta o número de documentos, é possível perceber um crescimento proporcionalmente logarítmico, o que evidencia que o tempo necessário para busca de palavras não cresce tanto a medida que aumenta-se o número de arquivos, já que menos nós são inseridos nas árvores, uma vez que muitos documentos possuem palavras repetidas, e a RBT e AVL possuem funções para balanceamento. Já na análise do tempo máximo, pode-ser notar uma taxa de crescimento similiar ao gráfico anterior. O que difere para o primeiro caso é a maior similaridade entre as três árvores e a existência de valores atípicos que provavelmente se explica por eventuais galhos degenerados, maiores que a média ou desproporcionais a $\log(n)$, que exigem mais tempo para busca, e pela falta de balanceamento da BST, uma vez que estes valores são mais frequentes nessa árvore, além disso, não pode-se descartar eventuais erros de medição no tempo de execução que são comuns no computador durante a geração das estatísticas. Outra observação importante, é que o gráfico de tempo médio para busca é bem refinido e proporcional a $\log(n)$ nos primeiros documentos, e depois apresenta oscilações, isso é explicado pois, para estes valores, como explicado, foram realizadas 50 buscas e calculadas a média dos tempos registrados, provavelmente, se o processo for repetido para mais documentos, o gráfico apresentará menos oscilações e será mais próximo de $\log(n)$.

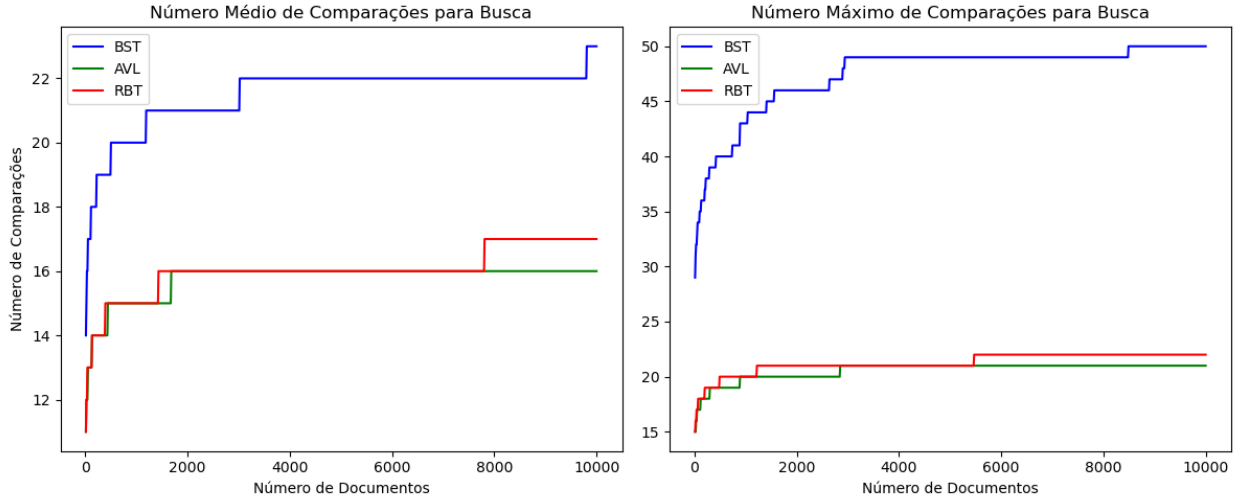
3.2.3 Número de Comparações

Figure 3: Número de Comparações por Inserção de Palavra



Fonte: Autoria própria.

Figure 4: Número de Comparações por Busca de Palavra

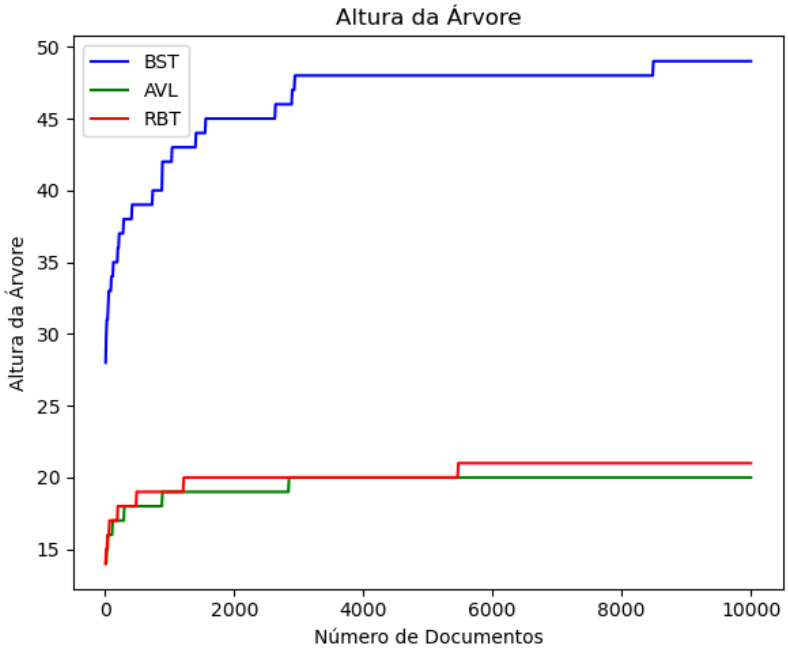


Fonte: Autoria própria.

Discussão: Tanto na inserção quanto na busca, as árvores AVL e RBT realizaram um número de comparações significativamente menor que a BST. Para inserções, a RBT exigiu o menor número total de comparações, seguida de perto pela AVL, o que corrobora com seus tempos de inserção mais baixos. Para buscas, a AVL foi marginalmente mais eficiente, realizando, em média e no pior caso, o menor número de comparações, enquanto a RBT apresentou resultados quase idênticos. A BST, por sua vez, teve o maior número de comparações, refletindo sua tendência ao desbalanceamento. Além disso, percebe-se um crescimento proporcional a $\log(n)$ para o número de comparações médio das árvores, o que é explicado pelo balanceamento, no caso da AVL e RBT, e pelo aumento na frequência de palavras repetidas, da mesma forma que na análise do tempo total de inserção da figura 1, a direita, pode-se observar um crescimento próximo de linear no número de comparações total para inserção, que se explica de forma análoga ao tempo de execução.

3.2.4 Altura da Árvore

Figure 5: Altura da Árvore

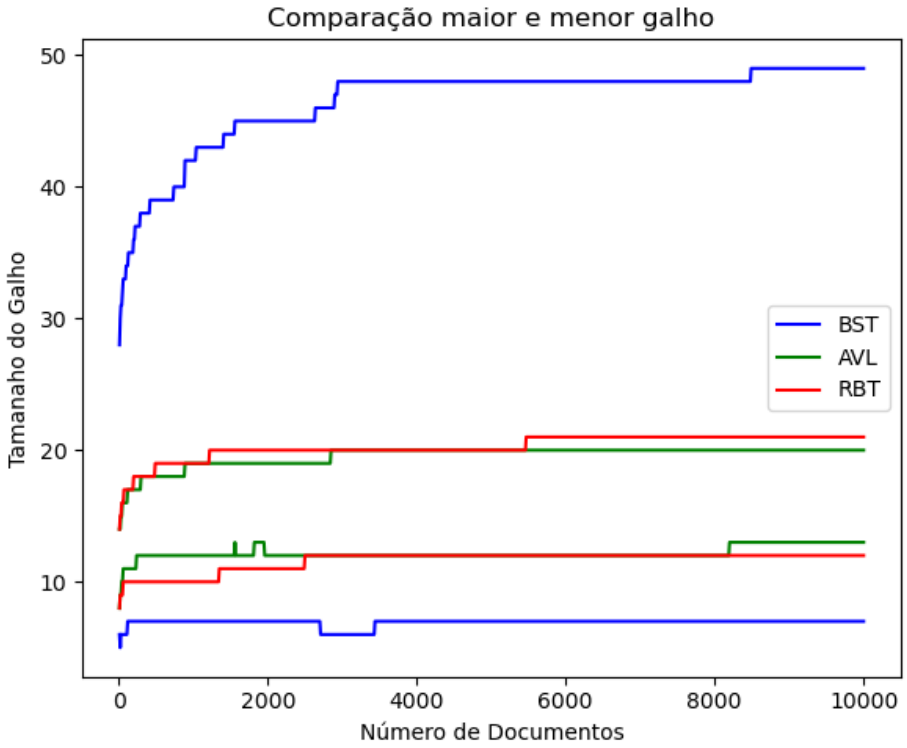


Fonte: Autoria própria.

Discussão: Observa-se uma diferença significativa de altura entre a BST e as árvores balanceadas (AVL e RBT). Essa disparidade ocorre porque a BST, por não possuir mecanismos de balanceamento, pode atingir alturas próximas a n em casos desfavoráveis, enquanto AVL e RBT mantêm a altura próxima de $\log(n)$ mesmo no pior cenário. Esse comportamento está de acordo com a teoria, já que a RBT, por adotar regras de balanceamento menos rígidas que a AVL, apresenta uma altura levemente superior, embora ambas permaneçam próximas ao ideal. Além disso, nota-se que a altura das árvores cresce lentamente à medida que aumenta o número de documentos. Isso pode ser atribuído a dois fatores principais: a presença de muitas palavras comuns entre os documentos, que não geram novos nós, e o fato de que, para aumentar a altura de árvores grandes, é necessário inserir novos nós em posições específicas (filhos de nós de altura máxima), o que se torna cada vez mais raro tanto na BST quanto nas árvores balanceadas.

3.2.5 Tamanho dos Galhos (Menor e Maior Caminho)

Figure 6: Tamanho do Maior e Menor Galho

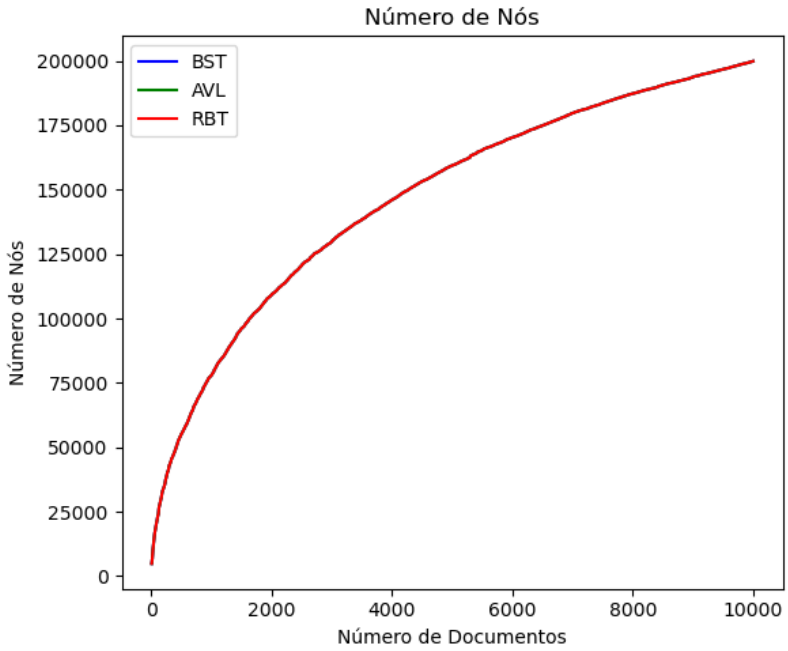


Fonte: Autoria própria.

Discussão: A diferença entre o maior (acima) e menor (abaixo) galho foi muito mais acentuada na BST, evidenciando sua falta de balanceamento. Em contraste, a AVL e a RBT mantiveram uma distância curta entre os galhos, refletindo suas propriedades de balanceamento. A RBT apresentou uma diferença ligeiramente maior que a AVL, devido ao seu balanceamento mais flexível, mas ainda dentro de limites aceitáveis.

3.2.6 Números de nós

Figure 7: Número de nós

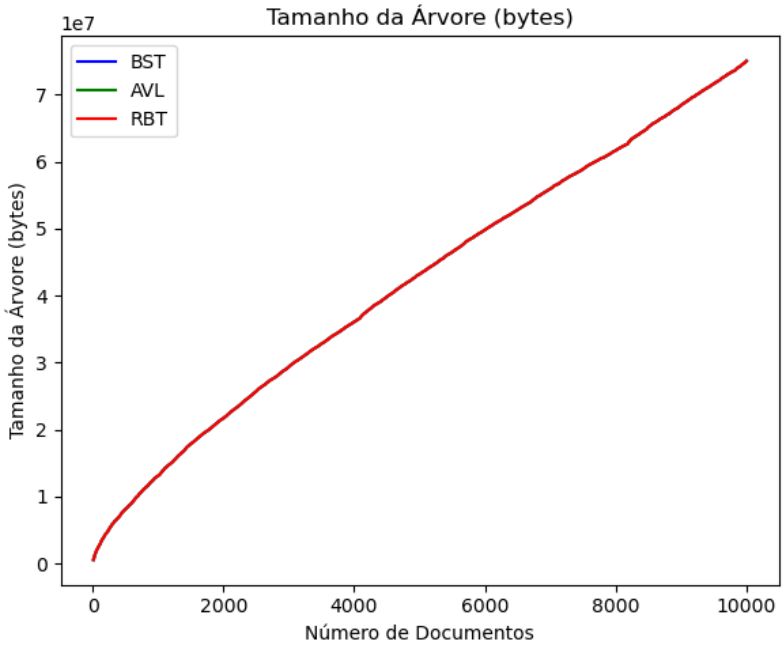


Fonte: Autoria própria.

Discussão: Observa-se que o número de nós é idêntico nas três estruturas, pois a diferença entre elas está apenas na forma de organização e balanceamento, não na quantidade de elementos inseridos. Cada palavra única resulta em um nó, independentemente da árvore utilizada. Inicialmente, o número de nós cresce rapidamente à medida que novas palavras são encontradas. No entanto, à medida que mais documentos são processados, a frequência de palavras repetidas aumenta, fazendo com que a inserção de novos nós se torne menos comum. Isso faz com que o crescimento do número total de nós tenda a ser côncavo, ou seja, a taxa de crescimento diminui ao longo do processamento de mais documentos, já que a maioria das novas inserções corresponde a palavras já existentes. Assim, mesmo com o aumento do corpus, a presença de palavras repetidas limita o crescimento do número total de nós.

3.2.7 Tamanho da árvore

Figure 8: Tamanho da árvore

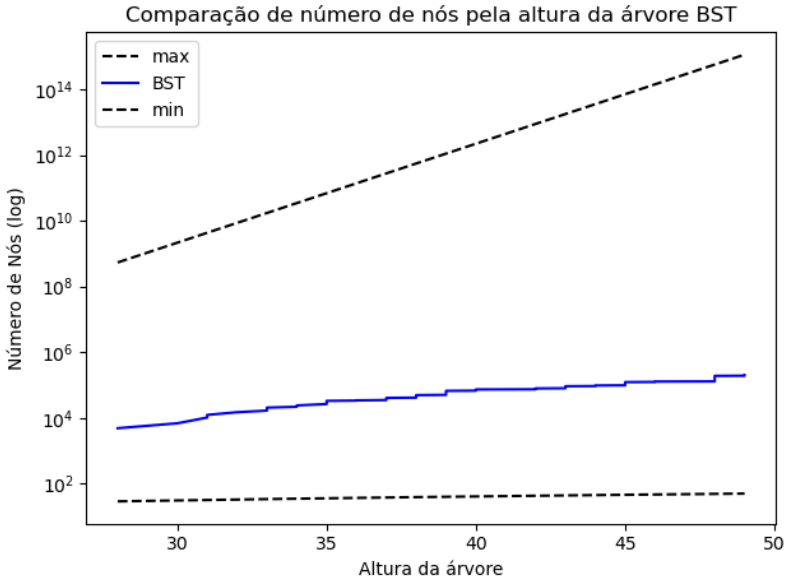


Fonte: Autoria própria.

Discussão: Como analisado, o crescimento do número de nós diminui a medida que se aumenta o número de documentos, o que resulta em um gráfico côncavo, entretanto, apesar do tamanho da árvore aumentar a medida que se aumenta o número de nós, independente da árvore utilizada, uma vez que o que se altera é a posição do nós em cada árvore, o crescimento se demonstra linear, o que pode ser explicado não somente pelo aumento no número de nós, como também pelo aumento do vetor de documentos de cada palavra, já que a medida que se processa mais documentos, se torna mais provável a existência de palavras repetidas, e, consequentemente, aumenta-se o tamanho do vetor de documentos de cada nó.

3.2.8 Altura da árvore vs Número de nós

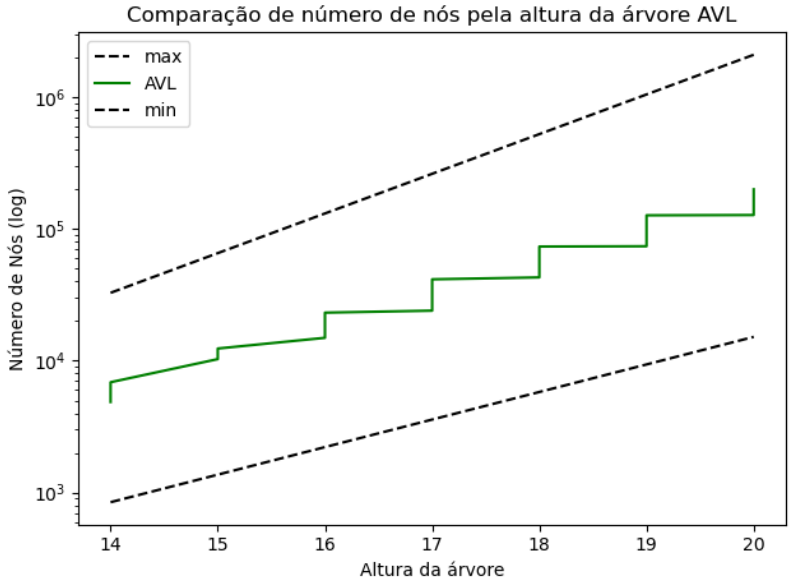
Figure 9: Altura da Árvore vs. Número de nós BST



Fonte: Autoria própria.

Discussão: A BST apresentou resultados dentro dos limites teóricos, com o número de nós variando entre o mínimo ($n \geq h+1$) e o máximo ($n \leq 2^{h+1}-1$). No entanto, sua tendência ao desbalanceamento foi evidente, com muitos casos próximos ao limite mínimo. Isso se mostra um problema, uma vez que uma árvore possuir poucos nós para uma determinada altura indica que ela está alongada, evidenciando um desbalanceamento, ao contrário de árvores com muitos nós para um determinada altura que mostra que ela está mais compactada.

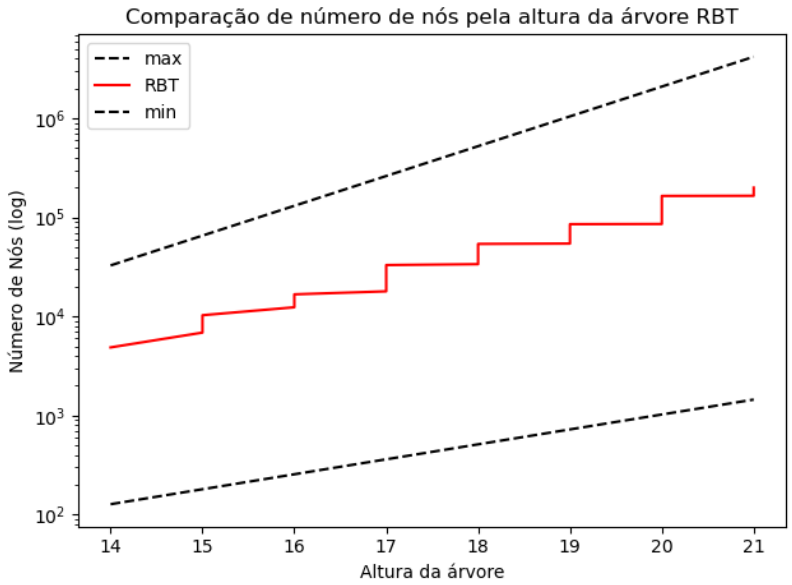
Figure 10: Altura da Árvore vs. Número de nós AVL



Fonte: Autoria própria.

Discussão: A AVL manteve o número de nós dentro dos limites teóricos, com o mínimo ($n \geq 2^{\frac{h}{1.44}}$) e o máximo ($n \leq 2^{h+1} - 1$). Deste modo, isso evidencia o correto funcionamento das implementações da árvore, que, como esperado, possui número de nós próximo ao centro das linhas correspondentes aos limites máximo e mínimo.

Figure 11: Altura da Árvore vs. Número de nós RBT



Fonte: Autoria própria.

Discussão: A RBT também apresentou resultados dentro dos limites teóricos, com o mínimo ($n \geq 2^{\frac{h}{2}} - 1$) e o máximo ($n \leq 2^{h+1} - 1$). Assim, constata-se que as implementações da árvore estão funcionando corretamente, uma vez que o número de nós se aproxima do meio das linhas correspondentes aos valores mínimo e máximo, como era esperado.

3.3 Resultados

Após a implementação das três estruturas de dados para aplicação do índice invertido e análise dos dados, pode-se extrair as seguintes conclusões:

3.3.1 Árvore Binária de Busca (BST)

- **Vantagens:** Simplicidade de implementação, sem a necessidade de criação de funções satélites para manter regras de balanceamento.
- **Desvantagens:** Propensa à degeneração, por não possuir regras de balanceamento, sua altura, bem como a diferença entre as alturas das subárvores irmãs, depende diretamente da ordem em que as palavras são inseridas. Isso faz com que dificilmente atinja uma altura proporcional a $\log(n)$, o que, por consequência, mina a eficiência das operações de busca e inserção.
- **Observado:** Os resultados confirmaram a tendência da BST ao desbalanceamento (Figuras 5 e 6), com uma altura significativamente maior e uma grande disparidade entre os galhos. O que impactou diretamente seu tempo de busca (Figura 2), que foi o mais lento entre as estruturas. E também impactou o número de comparações, que se mostraram os piores dentre as árvores (Figuras 3), já que encontrar o local para realizar a inserção é semelhante a realizar uma busca, que como observado é o mais custoso.

3.3.2 Árvore Adelson-Velsky e Landis (AVL)

- **Vantagens:** Garante um tempo de busca proporcionalmente logarítmico, mantendo a altura da árvore próxima do mínimo teórico ($\log(n)$).
- **Desvantagens:** Para manter o balanceamento, a árvore AVL implementa regras muito restritivas, o que faz com que muitas das novas inserções exijam ajustes na estrutura. Ajustes esses que envolvem a realização de rotações, que embora garantam a altura próxima a $\log(n)$, podem aumentar o tempo da realização da operação de inserção.
- **Observado:** A AVL manteve uma altura controlada e muito menor que a BST (Figura 5), resultando em número de comparações e tempo de busca (Figuras 2 e 4) consideravelmente inferiores aos da BST, provando sua eficácia para aplicações focadas em consulta. Embora apresente um custo de inserção mais elevado em comparação à BST, esse custo foi compensado pelo balanceamento eficiente da estrutura, refletindo em números de comparações menores que a BST nas operações de inserção (Figuras 3).

3.3.3 Árvore Rubro-Negra (RBT)

- **Vantagens:** Seu critério de balanceamento menos rígido que o da AVL geralmente resulta em menos rotações durante as inserções, tornando a operação de inserção potencialmente mais rápida.
- **Desvantagens:** A altura pode ser ligeiramente maior que a de uma AVL, podendo levar a um tempo de busca um pouco maior.
- **Observado:** A RBT apresentou um tempo de inserção mais rápido que a AVL (Figura 1), resultado do menor número de rotações necessitadas pelas suas regras. Em relação à busca, a RBT mostrou-se ligeiramente menos eficiente, realizando um número maior de comparações em comparação à AVL, embora o tempo de execução tenha se mantido bastante próximo (Figuras 2 e 4). Como esperado, sua altura foi levemente superior à da AVL (Figura 5), e sua diferença entre o maior e o menor galho também foi mais acentuada (Figura 6), refletindo seu balanceamento mais afrouxado.

3.4 Dificuldades encontradas

- **Complexidade de Implementação:** A lógica de balanceamento, especialmente as rotações simples e duplas da AVL e as funções de balaceamento da RBT, foram desafiadoras e exigiram depuração extensiva.
- **Sistemas Operacionais:** Adaptar o código e o Makefile para diferentes sistemas operacionais e preparar o repositório para ser acessível e reprodutível demonstrou-se algo difícil que necessitou de bastante esforço.
- **Trabalho em Equipe:** Coordenar a integração dos diferentes branchs desenvolvidas pelos membros e manter a consistência do código demandou comunicação eficaz e uso disciplinado do Git.
- **Validação dos Resultados:** Verificar e validar se os resultados obtidos eram similares aos esperados foi uma tarefa desafiadora e exigiu a produção de um grande número de testes.

4 Conclusão

A partir das análises, pode-se concluir que o desempenho da BST, no geral, foi insatisfatório, ao contrário da performance da AVL e RBT que obtiveram em geral bons resultados para as métricas estipuladas, exceto pelo tempo de inserção da AVL. Mesmo assim, apesar da RBT apresentar um maior desbalanceamento quando comparada a AVL, indicada pela diferença entre o maior e menor galho e a altura da árvore, a RBT possui um desempenho melhor que o da AVL, uma vez que os tempos de inserção são menores ou iguais que os da AVL e os tempo de busca são similares, o que é refletido no número de comparações semelhantes entre ambas as árvores. Deste modo, conclui-se que a Árvore Rubro-Negra (RBT) é a árvore recomendada para implementação do índice invertido.