TRABALHO PRÁTICO 3 - SISTEMA DE CONSULTAS

Luma Martins Ferreira Guimarães
Matrícula: 2024066067

7 de dezembro de 2025

1. Introdução

O armazenamento e a recuperação eficiente de informações urbanas representam um problema relevante em sistemas computacionais modernos, especialmente quando se trata de bases de dados extensas contendo endereços, logradouros e informações geográficas associadas. À medida que o número de registros cresce, operações simples de busca podem se tornar computacionalmente custosas, impactando diretamente o tempo de resposta das aplicações.

Neste contexto, o presente trabalho aborda o problema da indexação e consulta eficiente de uma base de endereços, permitindo consultas textuais baseadas em palavras-chave associadas a logradouros. O objetivo principal é desenvolver uma solução que seja capaz de lidar com grandes volumes de dados, fornecendo respostas eficientes mesmo sob variações significativas na carga de trabalho.

A solução proposta faz uso de estruturas de dados clássicas, como árvores balanceadas e listas encadeadas, organizadas de forma modular através de tipos abstratos de dados. Além da implementação, o trabalho inclui uma análise de complexidade teórica e uma análise experimental detalhada, investigando o impacto de diferentes parâmetros da base e das consultas sobre o desempenho do sistema.

2. Método

A implementação do sistema foi desenvolvida com foco na modularização e na separação clara de responsabilidades entre os diferentes componentes. Cada entidade fundamental do domínio do problema, como endereços, logradouros e palavras-chave, é representada por classes específicas, o que contribui para uma organização mais clara do código, facilita sua manutenção e permite futuras extensões do sistema sem a necessidade de alterações estruturais profundas.

A estrutura central de indexação do sistema é baseada em árvores balanceadas do tipo AVL. A escolha dessa estrutura se justifica principalmente pela garantia de balanceamento estrito, que assegura que a altura da árvore permaneça proporcional ao logaritmo do número de elementos armazenados. Essa propriedade garante que as operações fundamentais de inserção, remoção e busca apresentem complexidade $O(\log n)$ no pior caso, independentemente da ordem de inserção dos dados. Em um contexto no qual a distribuição das palavras e dos logradouros não é conhecida previamente e pode apresentar padrões adversos, essa garantia é essencial para evitar degradação de desempenho.

Além disso, como o sistema realiza um grande número de operações de busca durante o processamento das consultas, a previsibilidade do tempo de resposta proporcionada pela árvore AVL torna-se um fator determinante. Diferentemente de estruturas não balanceadas, que podem degenerar em listas lineares, a AVL assegura desempenho estável e confiável mesmo à medida que o volume de dados cresce, o que é especialmente relevante para bases de grande porte.

Para armazenar múltiplas associações relacionadas a uma mesma chave, como a associação entre uma palavra e vários logradouros, ou entre um logradouro e diversos endereços, foram utilizadas listas duplamente encadeadas. A escolha dessa estrutura se deve à sua flexibilidade e simplicidade, permitindo inserções e remoções eficientes sem a necessidade de realocação ou cópia de grandes blocos de memória. Esse comportamento é particularmente vantajoso em cenários nos quais a quantidade de associações por

chave pode variar significativamente.

O uso de listas encadeadas também permite percursos sequenciais eficientes durante o processamento das consultas, o que é desejável quando é necessário acessar todos os elementos associados a uma determinada palavra. Em comparação com estruturas baseadas em arranjos contíguos, as listas evitam custos elevados de realocação à medida que o conjunto de associações cresce, oferecendo um comportamento mais estável em termos de consumo de tempo e memória.

No que se refere à leitura dos dados de entrada, optou-se por realizar a leitura completa da base de endereços antes do início do processamento das consultas. Do ponto de vista computacional, essa decisão visa minimizar a influência do acesso à memória secundária nas medições de desempenho.Operações de entrada e saída em disco possuem custo significativamente superior às operações realizadas em memória principal, podendo mascarar o desempenho real das estruturas de dados se realizadas de forma intercalada com as consultas.

A leitura integral da base permite que toda a estrutura de dados seja construída previamente, garantindo que, durante a fase de consultas, o sistema opere exclusivamente sobre dados já carregados na memória. Essa abordagem melhora a localidade de referência, uma vez que os dados mais frequentemente acessados tendem a permanecer em cache, o que reduz o número de acessos à memória principal e contribui para um melhor desempenho prático.

Essa separação clara entre a fase de construção da base e a fase de processamento das consultas também facilita a análise experimental, permitindo que o tempo de construção das estruturas de dados e o tempo das consultas sejam medidos de forma independente. Com isso, torna-se possível avaliar com maior precisão o impacto do tamanho da base, da quantidade de logradouros e das características das consultas sobre o desempenho do sistema, estabelecendo uma relação mais direta entre os resultados experimentais e a análise teórica de complexidade apresentada neste trabalho.

3.Análise de complexidade

A análise de complexidade do sistema pode ser dividida em duas fases principais: a fase de construção da base de dados e a fase de processamento das consultas.

Durante a construção da base, cada endereço é inserido em estruturas de dados balanceadas. Considerando um conjunto com n endereços e m logradouros distintos, cada operação de inserção em uma árvore AVL ocorre em tempo $O(\log m)$. Como essa operação é repetida para cada endereço, o custo total de construção da base pode ser caracterizado como $O(n \log m)$. Esse resultado indica que o tempo de construção cresce de forma sublinear em relação ao número total de logradouros, garantindo escalabilidade adequada para grandes bases.

Em termos de complexidade espacial, o sistema armazena cada endereço e logradouro uma única vez, além das estruturas auxiliares necessárias para a indexação das palavras-chave. Assim, o consumo total de memória cresce de forma linear, sendo proporcional a $O(n + m)$, o que é adequado para aplicações que lidam com grandes volumes de dados.

No processamento das consultas, cada palavra contida na consulta é inicialmente buscada na árvore de indexação, operação que ocorre em tempo O(log n). Após essa busca, é necessário percorrer a lista de referências associadas à palavra encontrada, cujo custo depende diretamente do número de ocorrências dessa palavra na base. Dessa forma, o custo total de uma consulta pode ser expresso por O(k log n + r), onde *k* representa o número de palavras na consulta e *r* corresponde ao número total de resultados recuperados. Esse comportamento evidencia que consultas envolvendo palavras muito frequentes tendem a ser mais custosas, apesar da eficiência proporcionada pela estrutura de indexação.

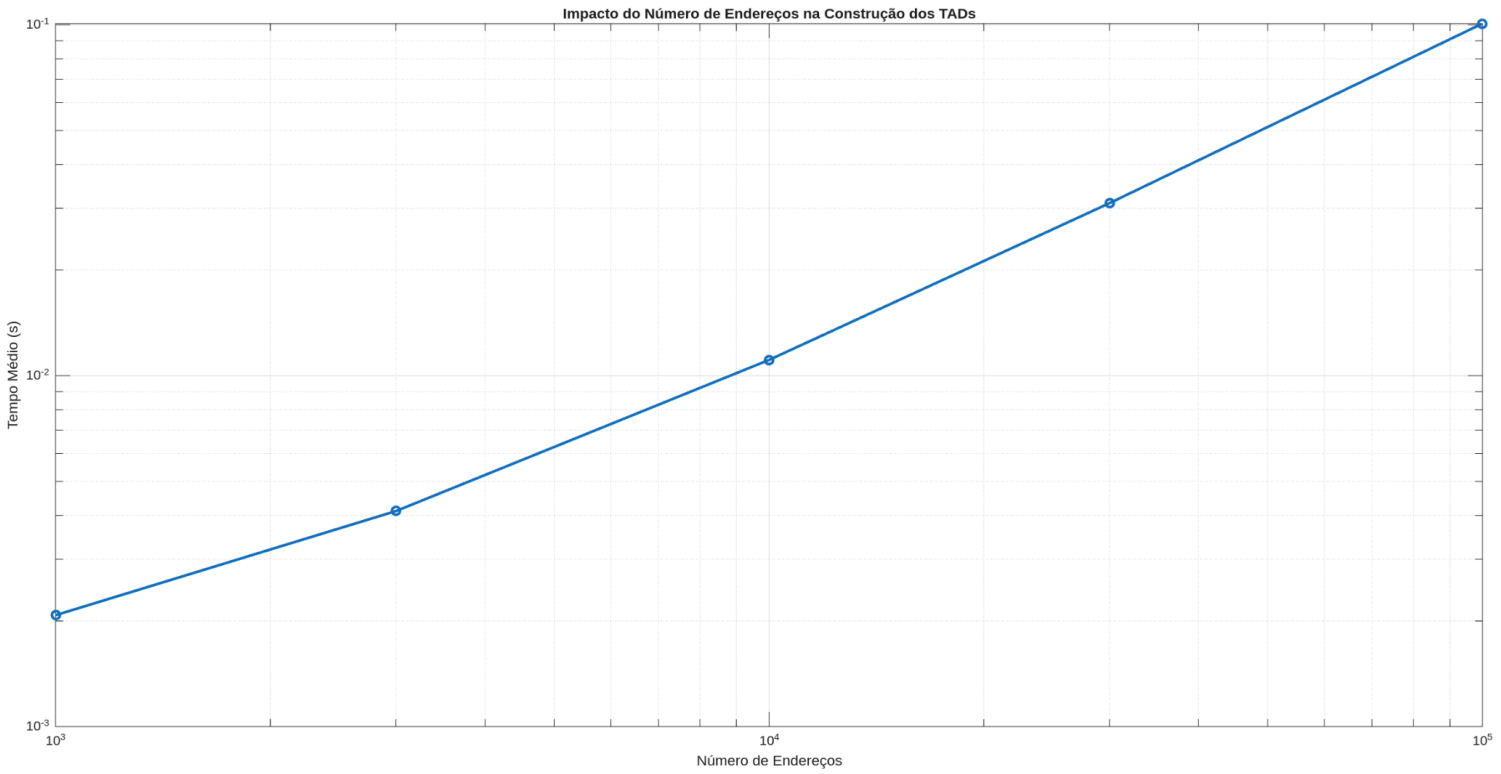
4.Estratégias de robustez

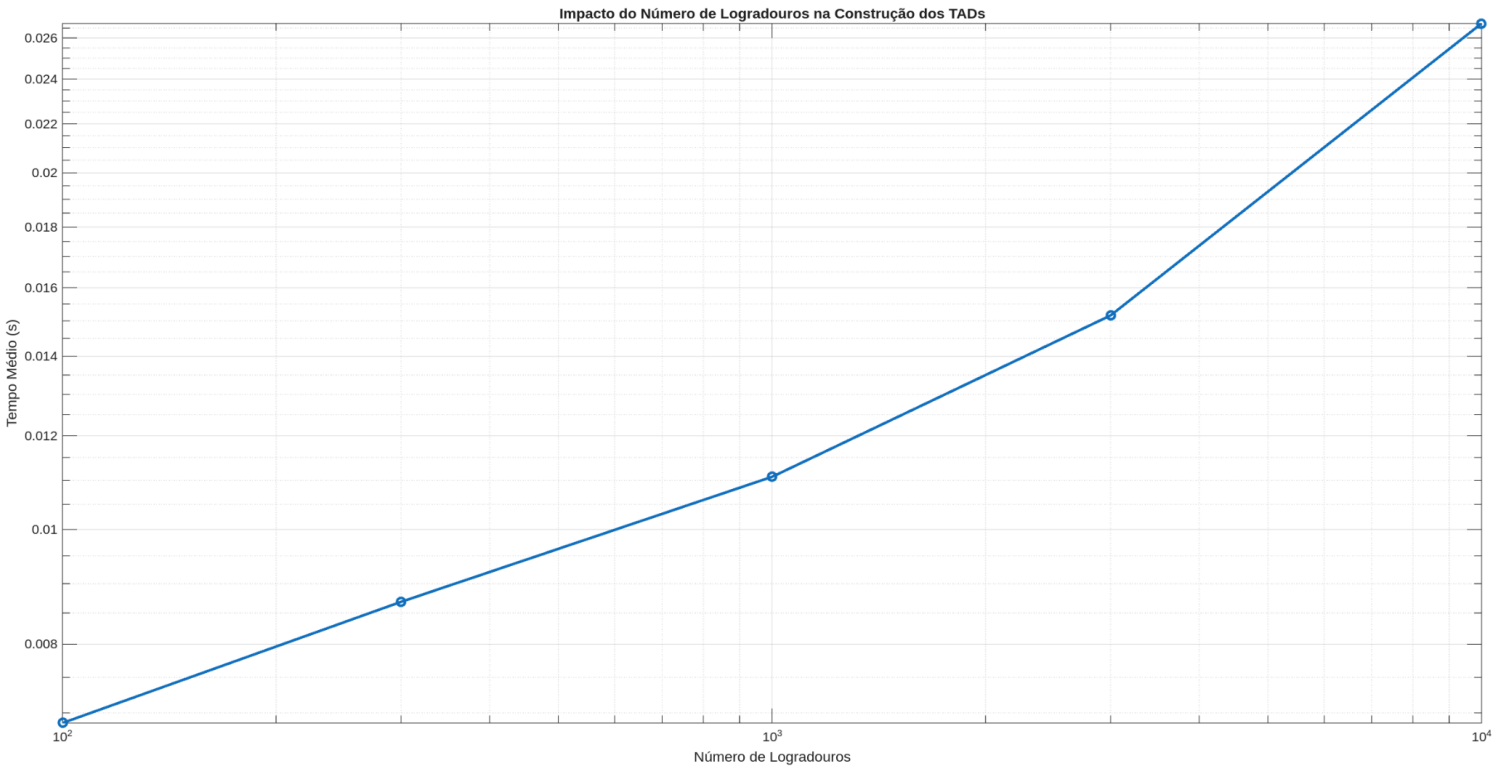
Ao longo do desenvolvimento do sistema, foram adotadas diversas práticas de programação defensiva com o intuito de aumentar sua robustez e confiabilidade. Uma dessas práticas é a verificação sistemática do uso correto de ponteiros, evitando acessos inválidos à memória e situações de desreferenciação nula que poderiam resultar em falhas de execução ou comportamento indefinido.

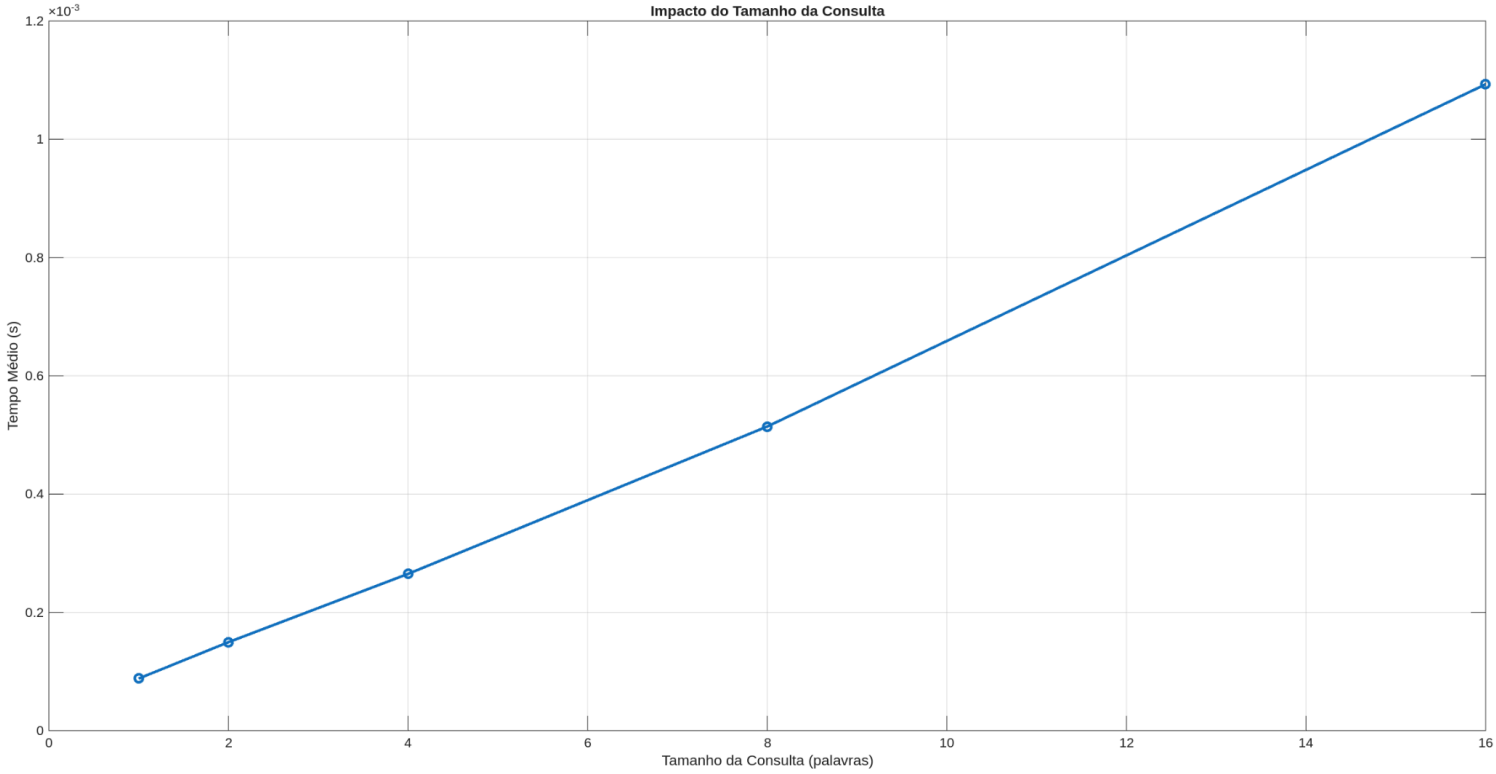
Além disso, o sistema verifica explicitamente se as estruturas de dados estão vazias antes da realização de operações como remoções ou acessos a elementos, garantindo comportamento bem definido mesmo em cenários extremos, como consultas realizadas sobre uma base ainda não inicializada. O encapsulamento das estruturas de dados em classes também contribui significativamente para a segurança do sistema, uma vez que impede a manipulação direta de seus dados internos, reduzindo o risco de inconsistências.

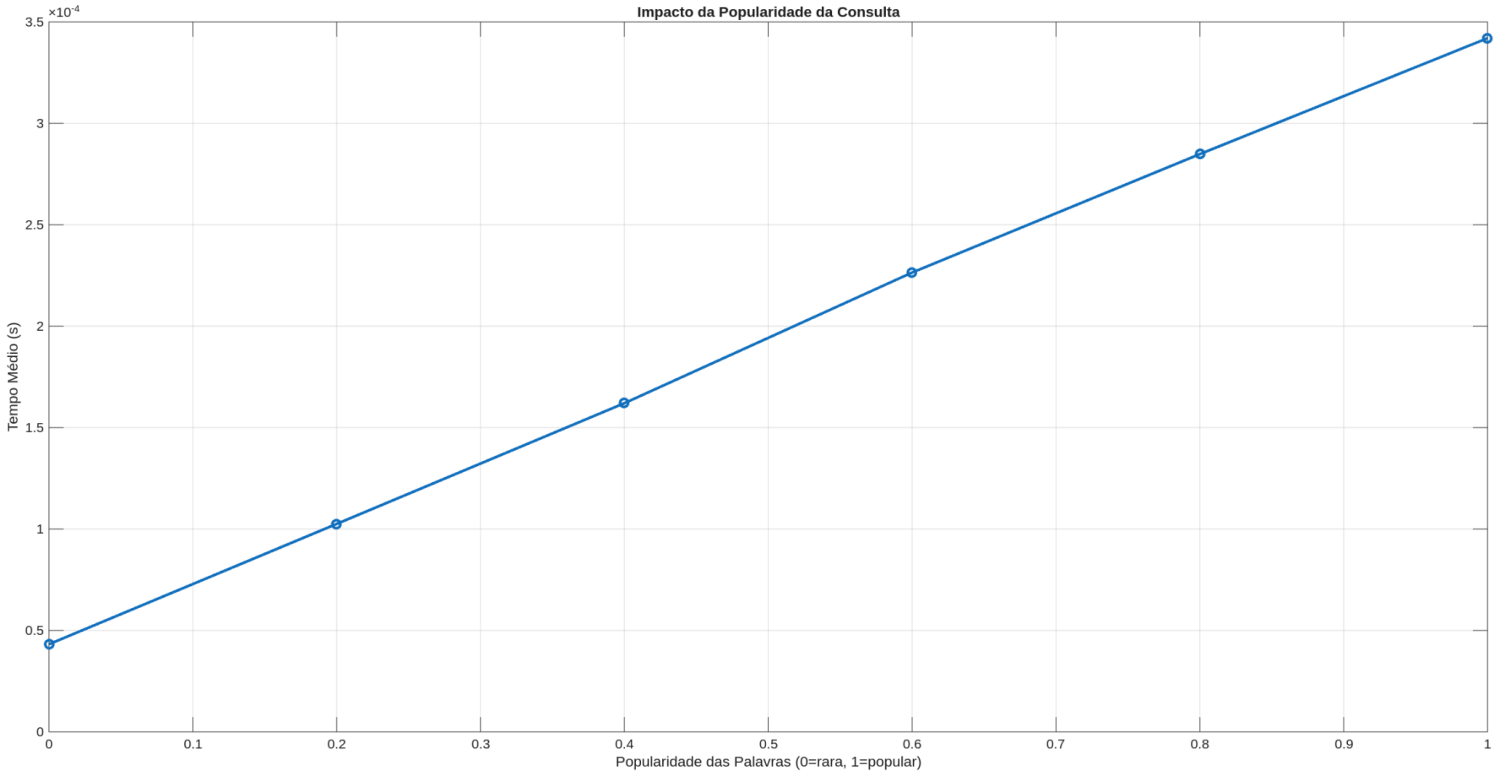
Essas estratégias, em conjunto, aumentam a confiabilidade da aplicação e facilitam a identificação e correção de erros, tanto durante o desenvolvimento quanto durante a execução do programa.

5.Análise experimental


Impacto do Número de Endereços na Construção dos TADs
Tempo Médio (s)
10-1
10-2
10-3
10³
10⁴
10⁵
Número de Endereços


Impacto do Número de Logradouros na Construção dos TADs
Tempo Médio (s)
0.026
0.024
0.022
0.02
0.018
0.016
0.014
0.012
0.01
0.008
10²
10³
10⁴
Número de Logradouros


Impacto do Tamanho da Consulta
×10^-3
1.2
1
0.8
0.6
0.4
0.2
0
Tempo Médio (s)
0
2
4
6
8
10
12
14
16
Tamanho da Consulta (palavras)


Impacto da Popularidade da Consulta
×10^-4
3.5
3
2.5
2
1.5
1
0.5
0
Tempo Médio (s)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Popularidade das Palavras (0=rara, 1=popular)

Todo o processo experimental foi automatizado por meio de um código desenvolvido em MATLAB, responsável por gerar instâncias aleatórias da base de dados, controlar os parâmetros dos experimentos, executar as medições repetidas e produzir os gráficos apresentados nesta análise. O uso do MATLAB permitiu maior controle sobre a variação dos parâmetros, além de facilitar o tratamento estatístico dos resultados e a visualização clara das tendências observadas.

Os experimentos consideraram variações independentes no número de endereços, no número de logradouros distintos, no tamanho das consultas e na popularidade das palavras utilizadas. A popularidade foi definida como o número médio de ocorrências associadas a cada palavra, simulando cenários realistas em que determinados termos aparecem com maior frequência na base. Para cada configuração experimental, os testes foram executados cem vezes com parâmetros aleatorizados, e os valores apresentados correspondem às médias dos tempos medidos, reduzindo o impacto de variações pontuais do ambiente de execução.

A fase de construção da base envolveu exclusivamente a inserção dos registros nas estruturas de dados, enquanto a fase de consultas foi realizada sobre uma base previamente construída e mantida integralmente em memória. Essa abordagem evitou que custos de leitura da memória secundária influenciassem os resultados, permitindo que os gráficos refletissem predominantemente o custo computacional das operações de busca e agregação de resultados.

Os gráficos gerados pelo código em MATLAB mostram que o tempo de construção da base cresce de forma consistente com o aumento do número de endereços e logradouros, comportamento compatível com a complexidade O(n log m) prevista teoricamente. Já no processamento das consultas, observou-se que o tamanho da consulta exerce influência direta no tempo de execução, uma vez que cada palavra adiciona uma operação de busca adicional na estrutura de indexação.

Entretanto, os resultados indicam que a popularidade das palavras utilizadas nas consultas tem impacto ainda mais significativo sobre o desempenho. Consultas formadas por palavras muito frequentes tendem a produzir listas de associações extensas, o que aumenta o custo de travessia e combinação dos resultados. Em contraste, consultas contendo termos menos frequentes apresentam tempos de resposta menores, mesmo quando compostas por um número maior de palavras.

De forma geral, os resultados apresentados nos gráficos confirmam a análise teórica de complexidade discutida anteriormente. O comportamento logarítmico das buscas nas árvores AVL mostrou-se adequado para sustentar o crescimento da base de dados, enquanto o custo linear associado às listas encadeadas tornou-se relevante apenas em cenários com elevada frequência de ocorrências. Assim, os experimentos reforçam que o desempenho do sistema depende não apenas do tamanho da base, mas também das características estatísticas das consultas realizadas.

6.Conclusões

Este trabalho apresentou o desenvolvimento e a análise de um sistema para indexação e consulta eficiente de endereços, fundamentado no uso de estruturas de dados clássicas, como árvores balanceadas do tipo AVL e listas duplamente encadeadas. A solução adotada mostrou-se adequada para lidar com grandes volumes de dados, fornecendo tempos de resposta estáveis e previsíveis mesmo em cenários de elevada carga.

Ao longo do desenvolvimento, foi possível observar, de forma prática, como as garantias teóricas

fornecidas pelas estruturas de dados influenciam diretamente o desempenho observado experimentalmente. O uso de árvores AVL assegurou eficiência nas operações de busca, inserção e remoção, enquanto as listas encadeadas permitiram flexibilidade no armazenamento de múltiplas associações sem incorrer em custos elevados de realocação de memória.

A análise experimental reafirmou a importância de separar claramente as fases de construção da base e de processamento das consultas, evidenciando que o desempenho do sistema é fortemente influenciado pelo número de ocorrências associadas aos termos consultados. Esse resultado destaca que, além do tamanho da base, a distribuição das palavras desempenha papel central na eficiência do sistema, um aspecto frequentemente negligenciado em análises mais superficiais.

Do ponto de vista do aprendizado, o trabalho possibilitou a consolidação de conceitos fundamentais relacionados a estruturas de dados dinâmicas, análise de complexidade e avaliação empírica de desempenho. A implementação modular e o uso de templates em C++ contribuíram para a reutilização de código e para um gerenciamento seguro da memória, reforçando boas práticas de programação.

Por fim, os resultados obtidos demonstram que a escolha criteriosa das estruturas de dados, aliada a uma análise cuidadosa do perfil de uso do sistema, é essencial para o desenvolvimento de aplicações eficientes e escaláveis. O sistema desenvolvido atende aos objetivos propostos e estabelece uma base sólida para extensões futuras, como a inclusão de novos critérios de consulta ou a adoção de estratégias adicionais de otimização.

7.Bibliografia

Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.