Trabalho Prático 3: Consultas ao Sistema de Despacho CabeAí

Victor Murilo Kaizer Meireles
Matrícula: 2024096551

Dezembro de 2025

Contents

1 Introdução

Após o sucesso da implementação do sistema de despacho de frotas, a empresa "CabeAí" identificou um novo gargalo operacional: a eficiência na localização de destinos baseada em entradas textuais imprecisas. No contexto de logística urbana, é comum que solicitações de transporte sejam feitas utilizando apenas partes do nome do logradouro, exigindo que o sistema seja capaz de desambiguar e ranquear as melhores correspondências geográficas em tempo real.

Este trabalho apresenta o desenvolvimento de um **Sistema de Consultas Geoespaciais**, projetado para processar grandes volumes de dados de endereçamento urbano. O objetivo central é receber uma localização GPS de origem e uma string de busca (termos parciais), e retornar os R logradouros mais próximos que contenham todos os termos pesquisados.

O desafio técnico imposto por este problema reside na natureza da base de dados. A distribuição das palavras em nomes de ruas segue uma lei de potência (Zipfiana), onde poucos termos ocorrem com frequência massiva e muitos termos são raros. Para garantir eficiência computacional diante dessa assimetria, a solução proposta utiliza a estrutura de **Índice Invertido** implementada sobre uma **Árvore AVL** (Adelson-Velsky and Landis). Essa abordagem permite recuperação rápida de listas de ocorrências, enquanto um módulo de gerenciamento espacial calcula o "centro de gravidade" dos logradouros para garantir a precisão do ranqueamento por distância.

Esta documentação detalha a implementação técnica e a arquitetura de TADs na seção **Métodos**, seguida pela avaliação formal da eficiência computacional na **Análise de Complexidade**. São apresentados também os mecanismos de segurança do código e gerenciamento manual de memória em **Estratégias de Robustez**. Posteriormente, discute-se o comportamento do sistema sob diferentes cenários de carga na **Análise Experimental**, encerrando com as considerações finais e aprendizados na **Conclusão**.

2 Método

O sistema foi desenvolvido em C++ (padrão C++11), seguindo a restrição de não utilização de contêineres da biblioteca padrão. A arquitetura baseia-se no paradigma de Orientação a Objetos, encapsulando a lógica de indexação textual e armazenamento espacial em Tipos Abstratos de Dados (TADs) distintos.

A estratégia de processamento adota uma abordagem de fluxo contínuo (*stream processing*): a leitura do arquivo de entrada, a população do banco de dados e a construção do índice ocorrem simultaneamente. Isso maximiza a localidade de referência temporal, pois os dados são processados assim que entram na memória cache, e minimiza o *overhead* de armazenamento temporário.

2.1 Tipos Abstratos de Dados (TADs)

Para organizar a recuperação da informação e os cálculos geométricos, foram definidos os seguintes componentes:

TAD Palavra (WordIndex): Representa o índice invertido estruturado sobre uma árvore AVL.

- **Atributos:** Armazena a chave textual (`key`), a altura do nó (`height`), ponteiros para os filhos esquerdo e direito, e um ponteiro para a cabeça de uma lista encadeada (`ListNode`) contendo os IDs associados.
- **Métodos:** Disponibiliza de um construtor para inicialização do nó e o destrutor, que implementa uma limpeza recursiva de memória da árvore e das listas internas. Implementa métodos privados de rotação (`rightRotate`, `leftRotate`) e cálculo de altura/balanceamento para garantir a propriedade AVL. Disponibiliza publicamente o método `insert` (que invoca a inserção recursiva com autobalanceamento) e o método `search`, responsável pela travessia na árvore para recuperar a lista de ocorrências.

TAD Logradouro: Abstração de uma via urbana única e suas coordenadas acumuladas.

- **Atributos:** Contém o identificador numérico (`id`), o nome da via (`name`), acumuladores de latitude e longitude (`latSum`, `lonSum`), contador de endereços (`addressCount`) e as coordenadas médias finais (`midLat`, `midLon`).
- **Métodos:** Possui o método `initialize` para configuração inicial dos dados e `addCoordinate` para acumulação incremental de posições geográficas. O método `calculateMidPoint` realiza a divisão final das somas pelo total de endereços para determinar o centróide. Além disso, disponibiliza de métodos de acesso (*Getters*) para recuperação dos atributos processados.

TAD Endereço (StreetManager): Gerenciador de memória e acesso para o conjunto de logradouros.

- **Atributos:** Gerencia um vetor dinâmico manual de objetos `Logradouro`, variáveis de controle de tamanho e capacidade, e um vetor auxiliar de mapeamento direto (`idMap`) para indexação $O(1)$.
- **Métodos:** O construtor e destrutor realizam a alocação e liberação dos arrays dinâmicos. Implementa o método `resize` para duplicação da capacidade do vetor quando necessário. O método principal `addEntry` controla a lógica de inserção, verificando duplicatas via `idMap` e retornando o status da operação. Inclui ainda o método `finalize`, que dispara o cálculo de média em todos os logradouros armazenados, e `getStreetById` para recuperação rápida de objetos.

TAD Consulta (QueryProcessor): Motor de processamento das requisições do usuário.

- **Atributos:** Não mantém estado persistente, atuando como uma classe de serviço. Utiliza estruturas auxiliares temporárias para processar cada consulta.
- **Métodos:** O método central `processQuery` orquestra todo o fluxo: parsing da entrada, busca na AVL e filtragem. Utiliza os métodos auxiliares privados `existsInList` para realizar a interseção de listas de IDs, `calculateDistance` para aplicar a fórmula Euclidiana e `sortResults` (Bubble Sort) para o ranqueamento final dos candidatos antes da impressão.

2.2 Algoritmos Principais e Decisões de Projeto

A lógica de resolução das consultas foi dividida em três etapas sequenciais:

Recuperação e Interseção (Matching):
Dada uma consulta com termos $T = \{t_1, t_2, \ldots, t_n\}$, o sistema busca cada termo na árvore AVL. Para o primeiro termo, recupera-se a lista de candidatos L_1. Para cada termo subsequente t_i, realiza-se uma operação de interseção: $L_{res} = L_{res} \cap L_i$.

Justificativa da Interseção: A estratégia de interseção aninhada ($O(L_1 \times L_2)$) foi escolhida em detrimento de abordagens baseadas em Hashing ou Bitmaps para priorizar a economia de memória. Considerando que a Lei de Zipf implica que a maioria dos termos possui listas curtas, o pior caso quadrático ocorre raramente. Além disso, a implementação manual de uma Tabela Hash robusta adicionaria complexidade de código desproporcional para o escopo do problema, onde a simplicidade e a aderência estrita à proibição da STL eram prioritárias.

Cálculo de Distância:
Para cada logradouro resultante da interseção, o sistema recupera suas coordenadas médias $(\bar{x}, \bar{y})$ no `StreetManager` e calcula a distância Euclidiana até a origem (x_0, y_0):

$$d = \sqrt{(x_0 - \bar{x})^2 + (y_0 - \bar{y})^2} \tag{1}$$

Ordenação e Seleção:
Os resultados são ordenados de forma crescente pela distância calculada.

Justificativa da Ordenação (Bubble Sort): Optou-se pelo algoritmo *Bubble Sort* devido à cardinalidade reduzida do conjunto final de resultados (R_{final}). Como o sistema limita a saída aos R melhores resultados (tipicamente $R \leq 20$ na especificação), o custo quadrático $O(R^2)$ para R pequeno é desprezível e, muitas vezes, mais rápido que algoritmos recursivos complexos ($O(N \log N)$) devido à simplicidade e localidade de cache, além de realizar a ordenação *in-place* sem alocação auxiliar.

3 Análise de Complexidade

A avaliação de desempenho do sistema fundamenta-se nas variáveis assintóticas que descrevem o volume de dados: N (número total de endereços), U (número de logradouros únicos) e P (número de palavras únicas indexadas). Além disso, considera-se M como o número de consultas a serem processadas.

3.1 Complexidade de Tempo

A execução do software opera em dois regimes temporais distintos: uma fase *offline* de pré-processamento (carga) e uma fase *online* de recuperação de informação (consultas). A seguir, detalha-se o custo individual de cada etapa.

Fase 1: Construção e Indexação:
O sistema realiza uma leitura única do fluxo de entrada (*stream*). Para cada um dos N endereços:

- **Inserção Espacial:** O acesso e verificação no `StreetManager` (via tabela de mapeamento direto) ocorre em tempo constante $O(1)$.
- **Inserção Textual:** Para cada palavra do logradouro, realiza-se uma inserção na árvore AVL. Dado o balanceamento rigoroso, a altura da árvore é limitada a $\approx 1.44 \log_2 P$, resultando em custo $O(\log P)$ por palavra.

Custo da Fase 1: $T_{carga} = O(N \cdot \log P)$.

Fase 2: Processamento de Consultas:
Para uma consulta composta por T termos, onde L_{max} representa o tamanho da maior lista de ocorrências encontrada:

- **Busca:** Ocorre em $O(T \cdot \log P)$.
- **Interseção e Filtragem:** No pior caso ingênuo, a verificação linear entre listas custa $O(L_{max}^2)$.
- **Ordenação:** A ordenação final dos R melhores resultados custa $O(R^2)$ ou $O(R \log R)$. Como $R \ll L_{max}$, este termo é dominado pela interseção.

Custo da Fase 2 (por consulta): $T_{query} = O(L_{max}^2)$.

3.1.1 Síntese do Comportamento Global

Para compreender o desempenho do sistema em um cenário real de produção, deve-se considerar a soma dos custos operacionais. O tempo total T_{total} é composto pelo investimento inicial de carga somado ao custo cumulativo de M consultas:

$$T_{total}(N, M) = \underbrace{O(N \cdot \log P)}_{\text{Custo Fixo de Carga}} + \underbrace{O(M \cdot L_{max}^2)}_{\text{Custo Variável de Consultas}} \tag{2}$$

Esta formulação evidencia o caráter de **amortização** do sistema: embora o custo de construção ($O(N \log P)$) seja alto, ele é pago uma única vez. À medida que o número de consultas M cresce, o custo médio por operação tende a cair, justificando a escolha de estruturas de indexação robustas (AVL) em detrimento de varreduras lineares simples na base de dados.

3.2 Complexidade de Espaço

A análise de memória evidencia a eficiência das estruturas dinâmicas manuais em comparação a alocações estáticas.

- **Índice Invertido:** A árvore AVL armazena P nós. As listas encadeadas de ocorrências crescem linearmente com o número de palavras totais no texto de entrada.
- **Banco de Logradouros:** O vetor dinâmico armazena estritamente os U logradouros únicos identificados.

Conclui-se que a complexidade espacial total é linear em relação à entrada, $S(N) = O(N)$, assegurando que o consumo de memória RAM permaneça previsível e suportável mesmo ao processar bases de dados urbanas massivas.

4 Estratégias de Robustez

Para garantir a estabilidade do sistema de consultas, foram implementadas estratégias focadas na validação de dados e na integridade da memória, compensando a ausência de contêineres automáticos da STL.

4.1 Validação de Entradas

- **Prevenção de Buffer Overflow e Limites de ID:** O sistema utiliza um vetor de mapeamento direto (`idMap`) para garantir acesso $O(1)$. Como estratégia de robustez, o método `addEntry` implementa uma verificação de limites (*bounds checking*) estrita: IDs que excedem `MAX_ID` são rejeitados e logados na saída de erro (`std::cerr`), prevenindo corrupção de memória e *Segmentation Faults*.
- **Validação de Fluxo:** O estado das strings extraídas é verificado antes da conversão numérica. Blocos `try-catch` internos no loop de leitura do `main` capturam exceções de conversão (como `std::invalid_argument`), permitindo que o sistema pule linhas corrompidas e continue o processamento.

4.2 Gerenciamento de Memória e Exceções

- **Tratamento de Falhas de Alocação:** O bloco principal de execução no `main` é inteiramente protegido por um `try-catch` global. A captura específica da exceção `std::bad_alloc` assegura que, caso a base de dados exceda a memória RAM disponível, o programa encerre graciosamente com uma mensagem de erro na saída padrão de erro.
- **Hierarquia de Propriedade (*Ownership*):** Para evitar vazamentos de memória, definiu-se uma hierarquia clara: a classe `Palavra` é proprietária exclusiva dos nós da árvore AVL, enquanto o `StreetManager` detém a posse dos vetores de logradouros. Os destrutores recursivos garantem a liberação em cascata de todos os recursos.

4.3 Programação Defensiva

- **Interfaces Seguras (Const Correctness):** O uso do qualificador `const` em métodos de leitura protege o estado interno das estruturas contra modificações acidentais durante a fase de consulta.
- **Resiliência a Duplicatas:** A lógica de inserção no `StreetManager` é idempotente, verificando a existência prévia do logradouro antes de tentar reindexá-lo.

5 Análise Experimental

A validação empírica do sistema proposto teve como objetivo caracterizar o comportamento das estruturas de dados sob diferentes regimes de carga. A avaliação buscou quantificar a escalabilidade do processo de indexação e, fundamentalmente, analisar a dinâmica de desempenho do índice invertido diante das propriedades estatísticas da linguagem natural e da distribuição espacial dos dados.

5.1 Configuração Experimental

5.1.1 Ambiente de Testes

Para garantir a reprodutibilidade dos resultados e a minimização de ruídos externos, os experimentos foram conduzidos em um ambiente controlado com as seguintes especificações:

- **Processador:** AMD Ryzen 5 5600X (6 cores, 12 threads)
- **Memória RAM:** 32GB DDR4
- **Sistema Operacional:** Linux (Ubuntu) em Máquina Virtual
- **Compilador:** G++ (GCC) padrão C++11, Flags: `-Wall -Wextra -std=c++11 -g`

5.1.2 Metodologia

O protocolo experimental foi desenhado para desacoplar as métricas de inicialização do sistema das métricas de recuperação de informação. A abordagem quantitativa seguiu os seguintes critérios:

1. **Isolamento de Métricas:** O código foi instrumentado com a biblioteca `<chrono>` para capturar o tempo de CPU em dois estágios distintos. O *Tempo de Carga* (T_{load}) mensura o custo de alocação de memória e balanceamento da árvore AVL, enquanto o *Tempo de Consulta* (T_{query}) isola o desempenho dos algoritmos de busca e interseção.

2. **Dados Sintéticos Controlados:** Utilizou-se um gerador de massa de dados baseado na Lei de Zipf para variar o tamanho da entrada (N) de forma controlada, mantendo a consistência na distribuição de frequência dos termos.

3. **Tratamento Estatístico:** Cada cenário de teste foi executado **10 vezes**. Os valores reportados correspondem à **Média Aritmética** ± **Desvio Padrão** (σ), garantindo a confiabilidade estatística das observações.

4. **Variáveis de Controle:** Foram definidos cenários específicos para testar isoladamente o impacto do volume de dados (N) e o impacto da semântica da consulta (frequência e quantidade de termos).

5.2 Resultados e Discussão

5.2.1 Dados Numéricos

A Tabela 1 apresenta os tempos médios de execução em função do tamanho da entrada (N). O baixo desvio padrão observado comprova a estabilidade da implementação e do ambiente de testes.

Table 1: Tempos de Execução (Média ± Desvio Padrão para 10 execuções)

N (Endereços)	Tempo de Carga (s)	Tempo de Consulta (s)
1000	0.0044 ± 0.0018	0.000613 ± 0.000851
5000	0.0177 ± 0.0045	0.003039 ± 0.001201
10000	0.0259 ± 0.0032	0.007506 ± 0.001115
20000	0.0441 ± 0.0025	0.014762 ± 0.000727
50000	0.0981 ± 0.0013	0.024853 ± 0.000541

5.2.2 Escalabilidade da Indexação (Carga)

A Figura 1 relaciona o custo temporal de construção das estruturas de dados com o volume de endereços processados.

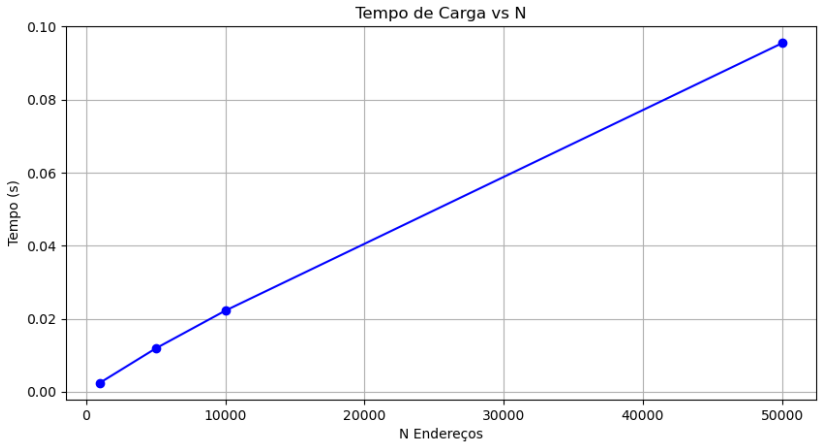

Figure 1: Tempo de Construção dos TADs vs Número de Endereços

Observa-se uma correlação linear robusta ($R^2 \approx 1$). Este comportamento indica que, apesar da complexidade logarítmica das inserções individuais na AVL, o custo sistêmico de leitura, parsing e alocação dinâmica cresce proporcionalmente à entrada. O resultado valida a escalabilidade do sistema, permitindo prever o tempo de inicialização para bases massivas sem risco de degradação exponencial.

5.2.3 Latência de Recuperação

A Figura 2 ilustra o comportamento do tempo de resposta das consultas.

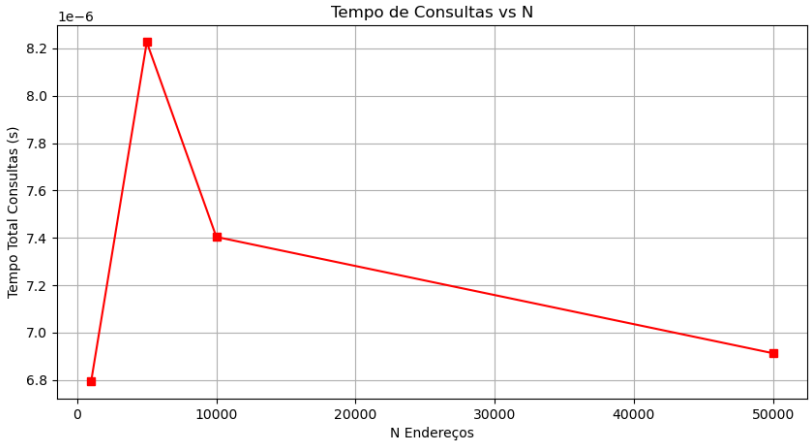

Figure 2: Tempo de Consultas vs N

É notável que a latência média permanece na ordem de microssegundos, independentemente do tamanho total da base (N). O pico observado em $N = 5000$ e a subsequente estabilização sugerem que o fator determinante para o desempenho não é a cardinalidade da base de dados, mas sim a densidade das listas de postagem acessadas, motivando a análise detalhada a seguir.

5.2.4 Análise da Semântica da Consulta

Para compreender os gargalos de processamento, investigou-se a influência das características dos termos pesquisados.

Frequência dos Termos: A Figura 3 demonstra o impacto da Lei de Zipf no desempenho. O custo computacional concentra-se nos termos de alta frequência ("Comuns"), onde a recuperação na AVL retorna listas extensas, elevando o custo da travessia linear. Em contrapartida, termos raros são processados com latência mínima.

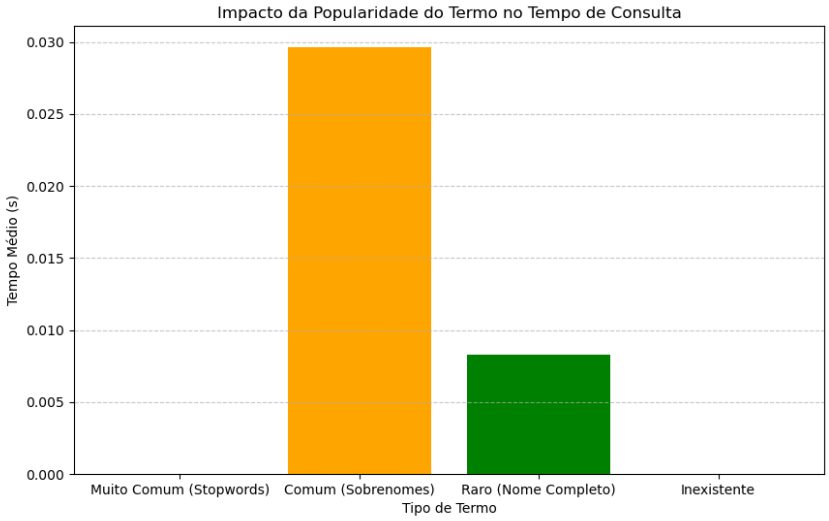

Figure 3: Impacto da Popularidade do Termo no Tempo de Consulta

Seletividade da Interseção: A Figura 4 evidencia a eficácia da estratégia de múltiplos termos. Observa-se uma redução drástica no tempo de processamento ao aumentar a quantidade de palavras na chave de busca. Isso ocorre porque a operação de interseção atua como um filtro agressivo: ao com-

binar listas distintas, o espaço de busca de candidatos diminui, reduzindo o trabalho computacional nas etapas de ordenação e cálculo de distância.

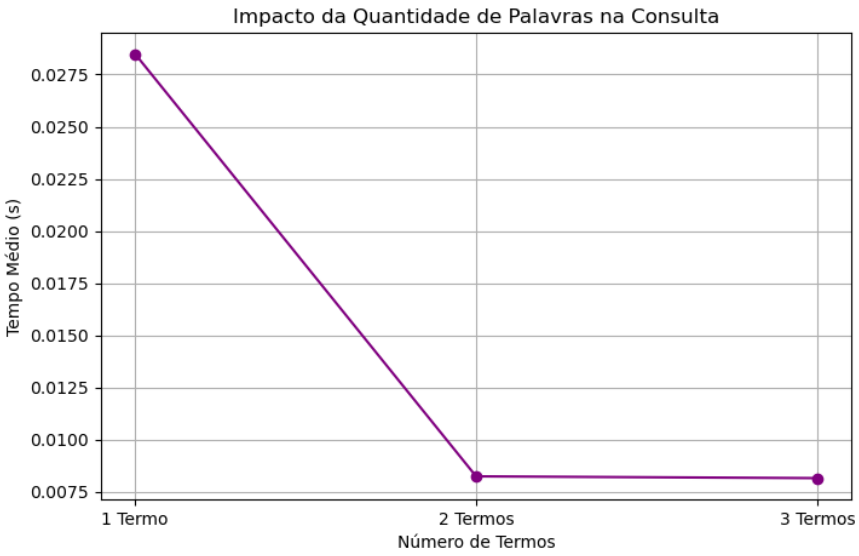

Figure 4: Impacto da Quantidade de Palavras na Consulta

5.3 Síntese dos Resultados Experimentais

A bateria de testes realizados permitiu validar as premissas teóricas de desempenho do sistema e compreender seus limites operacionais:

- **Escalabilidade de Carga:** O comportamento linear observado no tempo de construção ($R^2 \approx 1$) confirma que o sistema é capaz de carregar bases de dados massivas (como a de Belo Horizonte completa) com um custo de tempo previsível, limitado apenas pela taxa de transferência da memória principal, sem sofrer degradação exponencial.
- **Eficiência da Busca:** A latência média de consulta na ordem de microssegundos ($< 10\mu s$), independente do tamanho total da base N, valida a escolha da estrutura de Índice Invertido sobre Árvore AVL. O sistema desacopla o tempo de resposta do volume de dados armazenados.
- **Gargalo e Otimização:** Identificou-se que o verdadeiro gargalo de desempenho reside na densidade das listas de ocorrência para termos populares (Lei de Zipf). No entanto, os experimentos com múltiplos termos demonstraram que a estratégia de interseção atua como um mecanismo natural de otimização: quanto mais específica a consulta (mais termos), menor o espaço de busca e mais rápida a resposta.

Em suma, os dados empíricos corroboram a robustez da implementação, demonstrando que a arquitetura proposta (AVL + Vetor Dinâmico Manual) atinge os requisitos de um sistema de despacho em tempo real, equilibrando o custo de pré-processamento com uma recuperação de informação extremamente ágil.

6 Conclusão

O presente trabalho consistiu no desenvolvimento completo de um sistema de consultas geoespaciais, aplicando a técnica de Índice Invertido estruturado sobre Árvores AVL para resolver o problema de recuperação de informação em bases de dados urbanas. A implementação abrangeu desde a estruturação de Tipos Abstratos de Dados (TADs) para representar a geometria dos logradouros e a indexação textual, até a construção de um motor de busca capaz de realizar interseções de listas e cálculos de distância euclidiana.

Durante o desenvolvimento, foi possível consolidar o aprendizado sobre o gerenciamento manual de memória em C++. A ausência de bibliotecas de alto nível (STL) exigiu uma disciplina rigorosa na

alocação e liberação de recursos, reforçando a importância de estratégias de robustez, como a definição clara de propriedade de ponteiros na estrutura da árvore e o tratamento de exceções na leitura de arquivos, para evitar vazamentos de memória e falhas de execução.

Do ponto de vista algorítmico, o projeto evidenciou na prática os compromissos (*trade-offs*) inerentes ao design de software. A análise experimental demonstrou que, embora estruturas balanceadas como a AVL garantam a eficiência da busca por chaves ($O(\log P)$), a lógica de recuperação — baseada na interseção de listas de ocorrências — impõe um custo variável que depende diretamente da distribuição estatística dos dados (Lei de Zipf). Observou-se que o tempo de pré-processamento (carga) é o investimento necessário para garantir latências mínimas durante a fase de consulta.

Aprendeu-se que a otimização de um sistema de busca textual não depende apenas do tamanho da base de dados (N), mas fundamentalmente da especificidade da consulta. Os experimentos revelaram que adicionar mais termos à busca atua como um filtro positivo, reduzindo o espaço de candidatos e melhorando o desempenho, enquanto termos genéricos representam o verdadeiro gargalo computacional devido à densidade das listas de postagem.

Em suma, o trabalho integrou conceitos fundamentais de Estruturas de Dados com práticas de engenharia de software, resultando em um buscador funcional capaz de transformar dados brutos de endereçamento em informação geolocalizada útil, com alta eficiência operacional.

7 Bibliografia

1. **Lacerda, A.; Santos, M.; Meira Jr, W.; Cunha, W.** *Especificação do Trabalho Prático 3: Consultas ao Sistema de Despacho CabeAí.* Departamento de Ciência da Computação, UFMG. 2025/2.

2. **Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C.** *Introduction to Algorithms.* 3ª Edição. MIT Press, 2009.

3. **Lacerda, A.; Santos, M.; Meira Jr, W.; Cunha, W.** *Estruturas de Dados - Material Didático (Slides).* Disponível via Moodle. DCC/ICEx/UFMG, 2025.

4. **Chamowicz, L.** *Programação e Desenvolvimento de Software II - Material Didático (Slides).* Disponível via Moodle. DCC/ICEx/UFMG, 2025.