Trabalho Prático 3: Consultas ao Sistema de Despacho CabeAí

Marney Santos de Melo
Matrícula: [Oculta para versão pública]

Dezembro de 2025

Sumário

1 Introdução

O problema de indexação e recuperação eficiente de informações é um dos pilares da Ciência da Computação, sendo a base para o funcionamento de motores de busca modernos. Este trabalho apresenta a implementação de um sistema de consulta de logradouros que utiliza uma lista de palavras associadas a IDs em memória principal. O objetivo central é explorar o uso de estruturas de dados clássicas, combinando a rapidez de busca das Árvores AVL com a funcionalidade das Listas Encadeadas, para processar consultas textuais em grandes volumes de dados.

2 Método

O sistema foi desenvolvido em C++ (padrão C++11) em ambiente Linux. A solução é baseada em Tipos Abstratos de Dados (TADs) para encapsular as entidades de indexação e recuperação, priorizando a modularidade.

2.1 Tipos Abstratos de Dados (TADs)

Foram implementadas estruturas genéricas e específicas para compor o índice invertido (palavras-¿IDs) e gerenciar os dados espaciais.

- **Classe `Logradouro`:** Representa a entidade espacial. Armazena o ID único, o nome completo e uma lista de coordenadas (latitude e longitude), permitindo o cálculo do centro para medição de distâncias.
- **Classe `Lista` (Encadeada):** Estrutura de dados genérica utilizada para armazenar dados dinamicamente. No sistema, é utilizada tanto para armazenar a base de logradouros quanto para as listas de ocorrências (IDs) dentro de cada nó do índice.
- **Classe `IndicePalavras` (AVL):** Implementa uma Árvore Binária de Busca Balanceada (AVL). Cada nó da árvore mapeia uma palavra-chave (string) para uma `Lista` de IDs (inteiros), constituindo a estrutura central do índice invertido.

2.2 Algoritmo de Busca e Processamento

O processamento das consultas, implementado na classe `Consulta`, segue um fluxo de execução dividido em três fases principais para garantir a recuperação precisa e ordenada dos dados.

2.2.1 Fase 1: Recuperação e Interseção (Filtragem)

A consulta textual é dividida em termos individuais. Para cada termo, realiza-se uma busca na Árvore AVL para recuperar a lista de IDs onde a palavra ocorre. O algoritmo aplica uma interseção sequencial (lógica *AND*) entre as listas retornadas: apenas os IDs presentes em **todas** as listas de termos consultados são mantidos para a próxima fase.

2.2.2 Fase 2: Recuperação de Objetos

Com a lista final de IDs filtrados, o sistema realiza a recuperação dos objetos `Logradouro` completos armazenados na memória. Nesta etapa, é calculada a distância euclidiana entre a coordenada de origem fornecida na consulta e as coordenadas médias do logradouro encontrado.

2.2.3 Fase 3: Ordenação

Os resultados válidos são submetidos a um algoritmo de ordenação (*Bubble Sort*) baseando-se na distância calculada. A lista final é apresentada ao usuário ordenada do logradouro mais próximo para o mais distante, limitando-se ao número máximo de resultados requisitado.

2.3 Considerações sobre Gerenciamento de Memória

A base de dados é carregada integralmente na memória principal para reduzir a latência de I/O, um fator crítico em sistemas de recuperação de informação. Apesar de Listas Encadeadas apresentarem baixa localidade espacial, esse efeito é minimizado na etapa de ordenação: antes de ordenar, os dados filtrados são copiados para um vetor. Essa estratégia explora a localidade espacial de referência, permitindo melhor uso dos caches L1/L2 durante as comparações intensivas do Bubble Sort e resultando em menor tempo de processamento.

3 Análise de Complexidade

Para compreender o comportamento do algoritmo implementado, é indispensável a análise de complexidade assintótica do mesmo. Definimos as seguintes variáveis para a análise:

- N: Número total de endereços na base de dados.
- M: Número de palavras únicas indexadas (tamanho da árvore AVL).
- K: Número de palavras na consulta (geralmente pequeno).
- R: Número de resultados encontrados (tamanho da interseção final).

As consultas seguem as etapas abaixo, onde a complexidade de cada uma dessas etapas é analisada.

3.1 Busca por Palavras na AVL

Para cada uma das K palavras a serem consultadas, o algoritmo realiza uma busca na Árvore AVL. Como a árvore é balanceada e possui complexidade $O(\log M)$ para cada busca individual, temos:

$$C_{busca} = O(K \cdot \log M) \tag{1}$$

Considerando um K pequeno, essa etapa é eficiente.

3.2 Interseção das Listas ($O(N^2)$ - O Gargalo)

O principal custo computacional ocorre na etapa de interseção. Quando as palavras são encontradas, o sistema retorna listas de IDs de logradouros. No pior caso (palavras muito populares), essas listas podm conter quase todos os N endereços da base de dados. Como a interseção é obtida ao comparar cada elemento de uma lista com todos os elementos da outra (loops aninhados), a complexidade tende a $O(N^2)$.

$$C_{intersecao} = O(N^2) \tag{2}$$

3.3 Recuperação dos Objetos (Busca Linear)

Após obter a lista de IDs filtrados, i.e, IDs que correspondem a palavra consultada, o algoritmo precisa recuperar o objeto `Logradouro` completo (com todos os atributos) na lista principal para calcular a distância. A função `buscaLinearID` percorre a lista principal (N) para cada resultado encontrado (R). No pior caso, onde $R \approx N$, temos novamente $O(N^2)$.

$$C_{recuperacao} = O(R \cdot N) \tag{3}$$

3.4 Ordenação (Bubble Sort)

Por fim, os R resultados são ordenados pela distância utilizando o algoritmo *Bubble Sort*. Esse algoritmo foi implementado pois é trivial e, como o objetivo deste trabalho não é a análise de algoritmos de ordenação, a implementação dele foi válida, uma vez que simplifica a implementação e análise do que realmente importa.

$$C_{ordenacao} = O(R^2) \tag{4}$$

3.5 Complexidade Total

A complexidade temporal total (C_{total}) do processamento de uma consulta é dada pelo somatório das complexidades individuais de cada etapa descrita anteriormente:

$$C_{total} = C_{busca} + C_{intersecao} + C_{recuperacao} + C_{ordenacao} \tag{5}$$

Substituindo pelos termos assintóticos analisados:

$$C_{total} = O(K \cdot \log M) + O(N^2) + O(R \cdot N) + O(R^2) \tag{6}$$

Para determinar o comportamento assintótico final, analisamos os cenários extremos:

- **Pior Caso (Termos Populares):** Neste cenário, o número de resultados é alto, tendendo a $R \approx N$. Assim, os termos $O(N^2)$, $O(R \cdot N)$ e $O(R^2)$ convergem todos para uma complexidade quadrática. Como $O(N^2)$ domina $O(K \cdot \log M)$, temos:

  $$C_{total\_pior} = O(N^2) \tag{7}$$

  Isso explica a perda de desempenho observada nos experimentos com consultas de alta frequência (muitos endereços).

- **Melhor Caso (Termos Raros):** Neste cenário, o termo é específico e retorna poucos resultados, acontece quando nao existem interseções, logo R tende a uma constante pequena ($R \approx 1$). As etapas de interseção e ordenação tornam-se desprezíveis ($O(1)$). Entretanto, a etapa de **Recuperação dos Objetos** ainda exige uma varredura linear na lista principal para encontrar o ponteiro do logradouro pelo ID. Portanto, o termo $O(R \cdot N)$ torna-se $O(1 \cdot N)$:

$$C_{total\_melhor} = \Omega(N) \tag{8}$$

Em suma, o algoritmo é limitado inferiormente pela busca linear de recuperação ($\Omega(N)$) e superiormente pela interseção quadrática ($O(N^2)$).

4 Estratégias de Robustez

Para garantir que o programa execute até o fim sem erros ou falhas, foram implementadas validações na leitura dos dados. O algoritmo ignora linhas em branco e remove caracteres incorretos de formatação, que costumam causar erros. Além disso, antes de acessar qualquer dado na memória, o código verifica se o ponteiro é válido (!=nullptr), prevenindo *Segmentation Fault*.

5 Análise Experimental

O objetivo desta análise experimental foi avaliar o desempenho do algoritmo de busca implementado. O foco principal foi testar a eficiência dos TADs: AVL, utilizada para organizar as palavras chave, e as Listas Encadeadas, responsáveis por armazenar os IDs dos logradouros onde cada palavra aparece.

Os experimentos foram realizados todos na mesma máquina, um seguido do outro, utilizando scripts em Python para a geração de casos de teste genéricos. O tempo de execução foi medido utilizando a biblioteca `<chrono>` do C++. Para isolar os custos e realizar uma análise mais precisa, o tempo foi dividido em duas etapas:

- **Tempo de Carga:** Leitura do arquivo, inserção na Lista de Logradouros e construção da Árvore AVL.
- **Tempo de Consulta:** Busca na AVL, interseção de listas, recuperação de objetos e ordenação dos resultados.

5.1 Escalabilidade e Tamanho da Entrada (N)

O primeiro experimento avaliou o comportamento do sistema variando o número de endereços (N) de 1.000 a 200.000. O objetivo foi verificar, a medida que N cresce, o gasto computacional da construção do índice e o impacto no tempo de consulta.

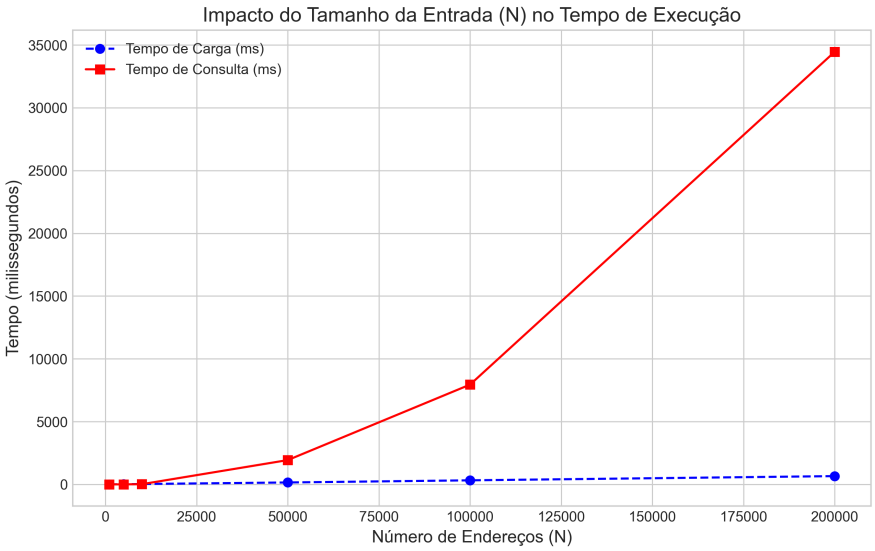

Figura 1: Tempo de Execução (Carga vs. Consulta) em função de N.

Como observado na Figura 1:

- **Carga (Linha Azul):** Apresentou um crescimento suave, próximo de um crescimento linear. Isso evidencia a eficiência da estrutura de indexação escolhida (Árvore AVL), onde as inserções possuem complexidade média logarítmica ($O(\log N)$). Observando esse comportamrnto, é razoável concluir que a etapa de construção do índice é escalável e não representa o gargalo do sistema, mesmo para grandes volumes de dados.

- **Consulta (Linha Vermelha):** Apresentou crescimento acentuado (tendência quadrática/ exponencial). Isso ocorre porque, nos testes de escalabilidade, o vocabulário utilizado gerou muitas colisões (termos comuns). O gargalo não reside na busca na árvore ($O(\log M)$), mas sim na manipulação das listas de Palvras associadas a Ids resultantes. A interseção de listas longas e a ordenação dos resultados (Bubble Sort) dominam o gasto computacional quando N cresce.

5.2 Impacto da Natureza da Consulta (Popularidade)

Para investigar o gargalo observado na consulta, realizou-se um experimento fixando $N = 20.000$ e variando a exclusividade do termo buscado. Comparou-se a busca por um termo "Popular"(presente em todos os registros) contra um termo "Raro"(presente em apenas um registro).

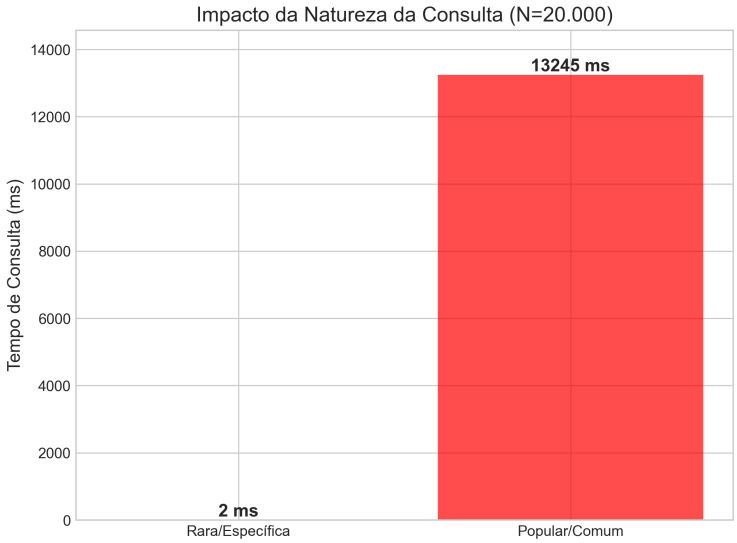

Figura 2: Comparação de tempo entre consultas de alta e baixa frequência.

A Figura 2 demonstra uma disparidade de desempenho de mais de 6.000 vezes.

- **Consulta Rara (2 ms):** A AVL localiza o termo rapidamente e retorna uma lista de tamanho 1. O custo de pós-processamento é desprezível.
- **Consulta Popular ($>$ 13 s):** A AVL retorna uma lista contendo 20.000 IDs. O sistema precisa realizar a interseção e, principalmente, a ordenação desses 20.000 itens. Este resultado comprova que o desempenho do algoritmo de busca é muito dependente da popularidade dos termos.

5.3 Impacto da Densidade de Logradouros

Por fim, avaliou-se como a diversidade do vocabulário afeta o desempenho. Fixou-se $N = 50.000$ em dois cenários:

1. **Alta Densidade:** Apenas 50 logradouros únicos (muitos endereços repetindo o mesmo nome).
2. **Baixa Densidade:** 50.000 logradouros únicos (cada endereço tem um nome distinto).

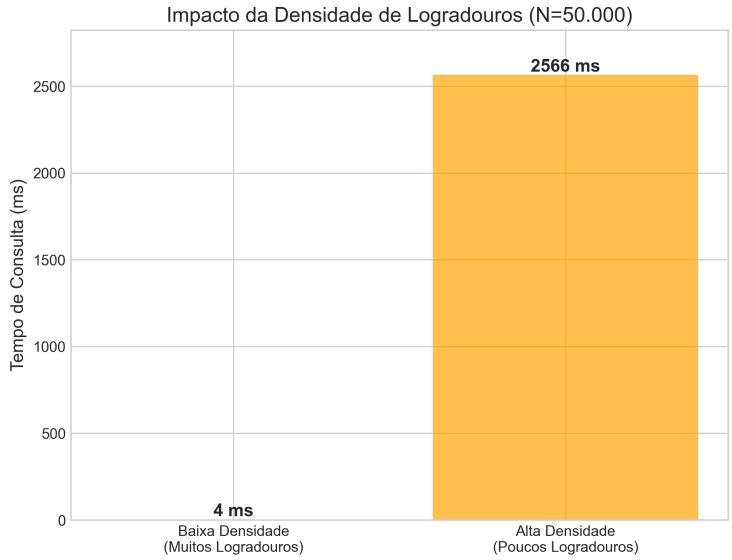

Figura 3: Impacto da repetição de logradouros no tempo de consulta.

Conforme a Figura 3, o cenário de Alta Densidade resultou em um tempo de consulta MUITO maior. Embora a árvore AVL seja menor (apenas 50 nós), cada nó carrega uma de lista palavras com média de 1.000 IDs associados a cada uma delas. No cenário de Baixa Densidade, a árvore é maior (50.000 nós), mas cada palavra é associada a apeanas um ID. Isso confirma que o tempo gasto descendo a árvore AVL é insignificante perto do trabalho pesado de processar as listas de ocorrências quando elas ficam muito grandes.

5.4 Conclusão dos Experimentos

Os resultados validam a implementação da Árvore AVL como uma estrutura eficiente para indexação em memória principal, apresentando um excelente desempenho. Entretanto, a análise revelou que algoritmos custosos de ordenação e interseção em listas encadeadas tornam-se um peso para consultas com baixa exclusividade (termos muito frequentes), sugerindo que possíveis otimizações deveriam focar no tratamento de listas de palavra→IDs.

6 Conclusão

O trabalho validou a eficiência da arquitetura baseada em Árvore AVL e Listas Encadeadas para indexação. A análise experimental confirmou a escalabilidade linear ($O(N)$) da etapa de carga, garantida pelo balanceamento da árvore. Por outro lado, o tempo de consulta demonstrou-se dependente da *exclusividade* dos termos, perdendo desempenho em cenários de alta frequência devido ao custo quadrático de manipulação de listas densas. Adicionalmente, a estratégia de converter as listas filtradas para vetores antes da ordenação otimizou a localidade espacial de referência (cache). Conclui-se que o sistema é robusto e altamente eficiente para buscas específicas,

7 Bibliografia

Referências

[1] Anisio, Marcio, Wagner, Washington. (2025). *DCC205/DCC221 Estruturas de Dados: Trabalho Prático 3 - Consultas ao Sistema de Despacho CabeAí*. DCC/ICEx/UFMG.

[2] Anisio, Marcio, Wagner, Washington, (2025). *Slides da Disciplina Estrutura de Dados* [Material Didático disponível via moodle]. DCC/ICEx/UFMG.

[3] Chamowicz, L. (2025). *Slides Disciplina Programação e Desenvolvimento de Software II* [Material Didático disponível via moodle]. DCC/ICEx/UFMG.

[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[5] GeeksforGeeks. *AVL Tree Data Structure*. GeeksforGeeks. Acessado em 02 de dezembro de 2025. Disponível em: https://www.geeksforgeeks.org/dsa/introduction-to-avl-tree/