Trabalho Prático 3 - Consultas ao Sistema de Despacho CabeAí

Igor Henrique Martins de Almeida

2023028536

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte - MG - Brasil

igorhma@ufmg.br

1. Introdução

Esta documentação apresenta a solução desenvolvida para a otimização logística da empresa “CabeAí”. O problema central abordado é a implementação de um sistema eficiente de consultas capaz de identificar destinos com base em nomes parciais de logradouros e na localização geográfica do usuário.

Dada uma base de dados real de endereços da cidade de Belo Horizonte, que apresenta uma distribuição enviesada de palavras, o desafio computacional consiste em processar esses dados para permitir buscas rápidas. O sistema deve receber como entrada a posição atual (coordenadas de GPS) e termos de busca textual, retornando como resposta os logradouros que contêm tais termos, ordenados pela proximidade em relação à origem informada.

Para resolver o problema, foi adotada uma abordagem modular utilizando a linguagem C, com o código organizado em Tipos Abstratos de Dados (TADs) específicos para garantir a eficiência e a organização lógica do sistema. A solução emprega uma Tabela Hash para o gerenciamento de logradouros únicos e o cálculo de seus centroides (médias de latitude e longitude), visando acesso rápido e tratamento de colisões. Simultaneamente, utiliza-se um Árvore Binária de Busca para a indexação das palavras, permitindo a recuperação eficiente dos identificadores de logradouros associados a cada termo da consulta. A classificação final dos resultados é realizada através do cálculo da distância Euclidiana combinado com o algoritmo de ordenação QuickSort.

A documentação está organizada da seguinte forma: a Seção 2 descreve detalhadamente a implementação, as estruturas de dados escolhidas e a interação entre os módulos. A Seção 3 apresenta a análise formal da complexidade de tempo e espaço dos algoritmos. A Seção 4 detalha as estratégias de robustez e programação defensiva. A seção 5 expõe a análise experimental de desempenho. Por fim, a Seção 6 apresenta as conclusões e a Seção 7, a bibliografia consultada.

2. Implementação

Para a implementação da solução, o código foi organizado de forma modular, seguindo a estrutura de diretórios exigida. A pasta *include* armazena os arquivos de cabeçalho (.h), que definem as interfaces dos TADs e os protótipos das funções, enquanto a pasta *src* contém os arquivos de código-fonte (.c), onde as funcionalidades são efetivamente implementadas. A compilação é gerenciada por um *Makefile*, que gera os arquivos objetos no diretório *obj* e o executável final, *tp3.out*, no diretório *bin*.

As principais estruturas de dados utilizadas foram *CadastroLogradouros*, *Logradouro*, *NoPalavra*, *IntVector*, projetadas para garantir eficiência tanto no armazenamento quanto na busca. O TAD *CadastroLogradouros* foi implementado utilizando uma Tabela Hash com endereçamento aberto e sondagem linear para o tratamento de colisões, permitindo o acesso rápido aos dados dos logradouros através de seus identificadores numéricos (*idLog*). Para otimizar a localidade de referência espacial durante o cálculo dos centroides, essa estrutura mantém também um vetor denso de ponteiros para os registros de *Logradouro*, onde são acumuladas as coordenadas de latitude e longitude e a contagem de ocorrências. O TAD *Logradouro* armazena as informações consolidadas de cada via, incluindo seu nome e as coordenadas médias calculadas após a leitura da base de dados.

Para o sistema de consultas textuais, empregou-se o TAD *Palavra*, estruturado como uma Árvore Binária de Busca não balanceada. Cada nó da árvore (*NoPalavra*) armazena um termo único identificado na base de dados e um ponteiro para uma estrutura *IntVector*. O *IntVector* é um vetor dinâmico de inteiros implementado manualmente para gerenciar as listas de identificadores de logradouros associados a cada palavra, permitindo o redimensionamento automático conforme a necessidade e evitando o desperdício de memória que ocorreria com vetores estáticos de tamanho fixo.

O fluxo de execução do programa inicia-se com a leitura do arquivo de entrada, onde os endereços são processados linha a linha para popular a Tabela Hash e acumular as coordenadas geográficas. Após a leitura, o sistema percorre o vetor denso de logradouros para calcular os centroides e constrói o índice invertido, inserindo cada palavra que compõe os nomes dos logradouros na Árvore Binária de Busca. Na etapa de consultas o programa lê os termos de busca e as coordenadas de origem, realiza a busca de cada termo na árvore e executa a interseção das listas de IDs resultadas para identificar os logradouros que contêm todas as palavras solicitadas.

Por fim, calcula-se a distância Euclidiana entre a origem e o centroide de cada logradouro encontrado, e os resultados são ordenados de forma crescente pela distância utilizando o algoritmo QuickSort, retornando ao usuário os destinos mais próximos até o limite requisitado.

Os testes do programa foram realizados em um ambiente com o sistema operacional Ubuntu 24.04.3, utilizando a linguagem C. A compilação do código foi feita com o compilador GCC em sua versão 13.3.0. O hardware utilizado para a execução dos testes foi um processador Intel Core i5-12400F e 32GB de memória RAM DDR4.

3. Instruções de compilação e execução

Para compilar e executar o programa, siga os passos descritos abaixo:

- Acesse o diretório raiz do projeto através do seu terminal;
- Execute o *Makefile* para compilar o projeto utilizando o seguinte comando: "*make all*";
- Com esse comando, os arquivos de código-fonte serão compilados, os arquivos objeto (.o) serão gerados na pasta *obj* e o arquivo executável, chamado *tp1.out*, será criado na pasta *bin*;
- Para executar o programa, utilize o seguinte comando, que redireciona um arquivo de entrada para o programa: "*./bin/tp1.out < caminho/para/seu/arquivo_de_entrada.txt*";
- A saída do programa, contendo as cenas geradas, é mostrada diretamente no terminal. Caso queira salvar a saída em um arquivo, proceda da seguinte maneira: "*./bin/tp1.out < caminho/para/seu/arquivo_de_entrada.txt > arquivo_de_saida.txt*".

4. Análise de complexidade

Nesta seção é apresentada a análise de complexidade de tempo e espaço para os os principais procedimentos implementados e para o sistema como um todo. Para essa análise considere as seguintes variáveis: N como o número total de endereços no arquivo de entrada, L como o número de logradouros únicos, P como o número de palavras únicas indexadas e S como o tamanho médio das listas de ocorrências (IDs) associados a cada palavra.

Função\Complexidade	Tempo	Espaço
ler_arquivo_entrada e registrar_logradouro	A função percorre as N linhas do arquivo de entrada uma única vez. Para cada linha, a inserção na tabela Hash (com tratamento de colisão por sondagem linear) tem o custo médio de *O(1)*. Assim, o tempo total é dominado pela leitura, resultando em $\theta(1)$.	O espaço utilizado cresce linearmente com o número de logradouros únicos armazenados no vetor denso e na Tabela Hash. Portanto, a complexidade é *O(L)*.
criar_indice_palavras e inserir_palavra	Esta função itera sobre os L logradouros únicos. Para cada palavra encontrada, realiza-se uma inserção na Árvore Binária de Busca. No caso médio, a inserção custa *O(log P)*, mas no pior caso (árvore degenerada) pode chegar a *O(P)*. O tempo total é *O(TotalPalavras * log P)* no caso médio.	A estrutura da árvore armazena cada palavra única e um vetor de IDs. O espaço é proporcional ao número de nós (P) mais o total de referências a logradouros armazenadas nos vetores dinâmicos, resultando em *O(P + Referencias)*.
processar_consulta	Para cada consulta com k termos, busca-se cada termo na árvore (*O(k * log P)*). A operação dominante é a interseção dos vetores de IDs, que exige a ordenação prévia via	O espaço auxiliar utilizado depende temporariamente dos vetores de interseçãoe da lista de resultados finais sendo *O(S)* no pior caso para armazenar os IDs

	QuickSort, custando *O(S logS*), onde S é o tamanho da lista de IDs. A ordenação final dos R resultados custa *O(R logR*).	recuperados.

Em relação ao programa como um todo, a complexidade de tempo é dominada pela fase de pré-processamento (leitura e indexação) ou pela quantidade de consultas, dependendo da carga de trabalho. A leitura e construção das estruturas iniciais possuem custo linear em relação à entrada, *O(N)*, garantindo que o sistema seja capaz de carregar grandes bases de endereços eficientemente. Durante as consultas, a eficiência depende do balanceamento da árvore BST e do tamanho das listas de logradouros associados às palavras comuns; contudo, o uso de interseção otimizada e hash para recuperação de coordenadas mantém o tempo de resposta adequado.

Para a complexidade de espaço, o custo geral é *O(L + P + R)*, onde o sistema armazena apenas os logradouros únicos (L) e as palavras únicas (P), além das referências cruzadas (R). Como L é significativamente menor que N (vários endereços compartilham o mesmo logradouro) e o uso de vetores dinâmicos evita o desperdício de memória, a solução apresenta uma eficiência espacial robusta, alocando memória proporcional apenas à diversidade da base de dados, e não ao seu tamanho bruto total.

5. Estratégias de Robustez

Para garantir a estabilidade do sistema e prevenir falhas durante a execução, especialmente ao lidar com grandes volumes de dados ou entradas malformadas, foram adotadas diversas estratégias de programação defensiva. O foco principal foi a integridade da memória e a manutenção da eficiência das estruturas de dados sob carga.

Gerenciamento dinâmico e redimensionamento de estruturas: uma das principais vulnerabilidades em sistemas baseados em hashing é a degradação de desempenho ou falha por falta de espaço quando a entrada supera a capacidade estimada. Para mitigar isso, o TAD *Logradouro* implementa uma estratégia de redimensionamento: ao atingir o limite de capacidade do vetor de logradouros, tanto o armazenamento denso quanto a Tabela Hash são expandidos automaticamente. A tabela é recriada com o dobro do tamanho e todos os registros existentes são

remapeados. Isso garante que o fator de carga permaneça baixo, mantendo a complexidade de busco próxima de *O(1)* e evitando loops infinitos na sondagem linear.

Validação de alocação de memória: todas as operações que envolvem alocação dinâmica são seguidas por verificações imediatas. Caso o sistema operacional não consiga fornecer a memória solicitada o programa encerra a execução de forma controlada ou retorna códigos de erro apropriados, prevenindo acessos a ponteiros nulos que causariam falhas de segmentação.

Sanitização e validação da entrada: o processamento de arquivos de texto lida com a variabilidade de formatação. O uso da função *trim* remove espaços em branco no início e fim das strings antes do processamento, e a função *split* verifica a quantidade de tokens obtidos antes de tentar acessá-los. Na leitura dos endereços, o retorno de *fgets* é validado para garantir que o fluxo de leitura não foi interrompido, e linhas que não possuem os campos mínimos necessários são descartadas silenciosamente, garantindo que apenas dados válidos preencham as estruturas.

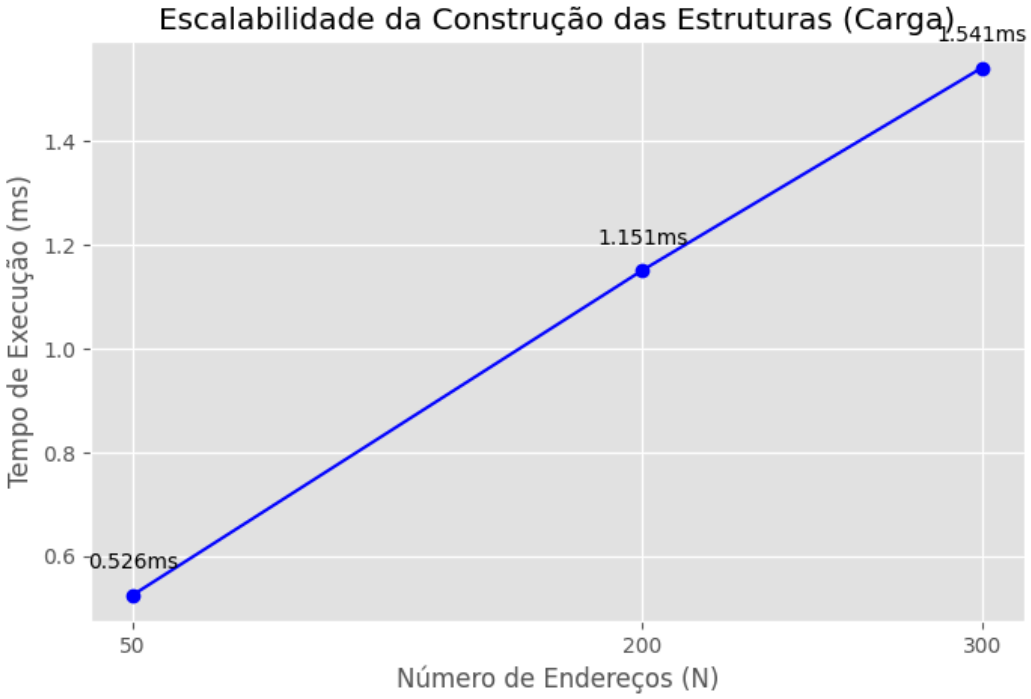
Segurança em estruturas recursivas: no TAD *Palavra*, as operações na Árvore Binária de Busca implementam verificações de caso base para nós nulos. Isso impede erros de acesso indevidos ao percorrer a árvore, garantindo que buscas por palavras inexistentes retornem graciosamente sem derrubar o sistema.

6. Análise Experimental

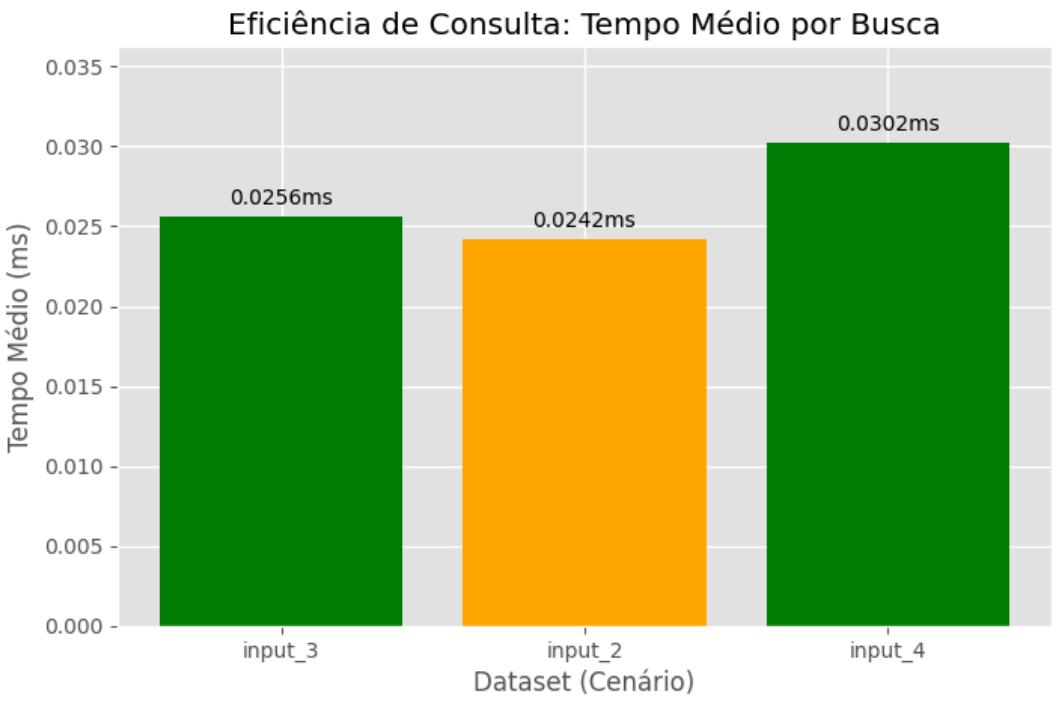
Esta seção apresenta a análise de desempenho da solução implementada para o sistema de consultas “CabeAí”, avaliando a eficiência das estruturas de dados sob diferentes volumes de dados. Para garantir a precisão das medições, o tempo de execução foi segmentado em duas etapas distintas: o tempo de carga, que engloba a leitura do arquivo e construção das estruturas de dados, e o tempo de consulta, referente ao processamento das buscas e ordenação dos resultados. Foram utilizados 3 arquivos para teste, sendo dois deles disponibilizados via moodle: os arquivos *input_2*, com 200 endereços e 10 consultas, *input_3*, com 50 endereços e 5 consultas, e *input_4*, criado a partir da junção dos 3 arquivos de teste disponibilizados via moodle (*input_1*, *input_2* e *input_3*) contendo 300 endereços e 5 consultas.

O primeiro cenário avaliado focou no impacto do número de endereços no tempo de carga e construção dos TADs. Para esse experimento foram utilizados os 3 arquivos descritos acima. Os resultados obtidos demonstraram um comportamento coerente com a complexidade assintótica linear *O(N)* esperada para a inserção na

Tabela Hash e na Árvore. Observa-se que, ao aumentar a entrada em 505 (de 200 para 300 endereços), o tempo de processamento cresceu aproximadamente 34%, indicando uma escalabilidade eficiente e próxima da linearidade, onde os custos fixos de I/O e alocação de memória se tornam menos representativos conforme a entrada cresce. Os resultados desse experimento são apresentados no gráfico abaixo.



O segundo cenário analisou a eficiência do processamento das consultas, isolando o impacto do tamanho da base de dados no tempo de resposta. Para isso, comparou-se o desempenho entre o *input_3* e o *input_4*. Ambos executam o mesmo conjunto de 5 consultas textuais idênticas, porém sobre bases de dados de tamanhos distintos. O tempo médio por consulta sofreu uma variação mínima, como podemos ver pelo gráfico abaixo. Esse aumento marginal de apenas 18% no tempo de resposta mesmo diante de uma base de dados 600% maior, evidencia a robustez da Tabela Hash e da Árvore Binária, confirmando que a solução é capaz de manter tempos de respostas rápidos mesmo com o crescimento do volume de dados armazenado.



Por fim, foi avaliada a consistência do sistema frente a consultas variadas utilizando o arquivo *input_2*. Com uma base de tamanho intermediário e um conjunto maior de 10 consultas distintas, o tempo médio por consulta se manteve próximo aos demais cenários. Isso demonstra que o desempenho do sistema é estável e não apresenta degradação significativa dependendo dos termos buscados. Em conclusão, os experimentos validam as escolhas do projeto, demonstrando que a combinação de indexação por árvores para palavras e hashing para logradouros oferece uma solução escalável, com custo de construção linear e tempos de busca extremamente eficientes.

7. Conclusão

Este trabalho abordou o desenvolvimento de um módulo de consultas logísticas para o sistema “CabeAí”, implementando uma solução em linguagem C capaz de processar bases de dados de endereços reais e realizar buscas eficientes baseadas em texto e proximidade geográfica. Através da combinação estratégica de TADs, especificamente um Tabela Hash com endereçamento aberto para o gerenciamento de logradouros e uma Árvore Binária de Busca para a indexação de palavras, foi possível atingir os objetivos de desempenho e escalabilidade propostos, garantindo respostas rápidas mesmo diante de grandes volumes de dados.

Com a realização deste projeto, consolidaram-se conceitos fundamentais da disciplina. A implementação exigiu um entendimento prático sobre o compromisso

entre tempo e espaço, evidenciando como a escolha adequada de estruturas é crucial para lidar com dados de distribuição enviesada. O trabalho permitiu aprofundar o conhecimento em técnicas de hashing e tratamento de colisões, manipulação de vetores dinâmicos para otimização de memória e a aplicação de algoritmos de ordenação eficientes como o QuickSort.

Além dos aspectos algorítmicos, o projeto reforçou a importância da robustez no desenvolvimento de software. A implementação de estratégias como o redimensionamento dinâmico da Tabela Hash e a sanitização de entradas mostrou-se essencial para garantir a estabilidade do sistema frente a diferentes cargas de trabalho. Por fim, a condução da análise experimental validou as hipóteses teóricas de complexidade, demonstrando na prática que a solução escala linearmente durante a carga de dados e mantém uma eficiência elevada no processamento das consultas, cumprindo com êxito os requisitos dos sistema de despacho.

8. Bibliografia

Cunha, W., Lacerda, A., Meira Jr., W., Santos, M. (2025). Slides virtuais da disciplina de estrutura de dados.

Disponibilizados via moodle. Departamento de Ciência da Computação.

Universidade Federal de Minas Gerais. Belo Horizonte.