Trabalho Prático 3 de Estruturas de Dados

Marcelo Eugênio Resende Campos
Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
marceloerc@dcc.ufmg.br

---

1. Introdução

O trabalho prático apresentado implementa um sistema eficiente de consultas geoespaciais e integração logística que necessita identificar destinos (logradouros) baseados em buscas por palavras-chave e proximidade geográfica. O sistema, idealmente, recebe uma base de dados com centenas de milhares de endereços e deve processar consultas em tempo real, retornando os logradouros mais próximos que correspondem aos critérios de busca especificados.

Para resolver o problema citado, foi implementada uma abordagem baseada em duas etapas principais: (i) a construção incremental de índices durante a leitura dos endereços, incluindo o cálculo de médias coordenadas e a criação de um índice invertido suportado por árvores AVL; e (ii) o processamento das consultas, envolvendo a extração das palavras-chave, a interseção eficiente das listas de logradouros candidatos e a seleção dos R resultados mais próximos utilizando um MaxHeap modificado. Além dessas estruturas centrais, foram desenvolvidos TADs auxiliares, como arrays dinâmicos, mapas balanceados e abstrações para endereços, logradouros, palavras e consultas.

Segue a organização desta documentação:

A Seção 2 descreve detalhadamente a implementação da solução, abordando as fases do processamento, as estruturas de dados desenvolvidas, as principais decisões de projeto e a configuração do ambiente de testes.

A Seção 3 apresenta as instruções de compilação e execução do sistema.

A Seção 4 discute a análise de complexidade teórica da solução.

A Seção 5 trata das estratégias de robustez empregadas no código.

A Seção 6 apresenta a análise experimental realizada com dados reais.

Por fim, a Seção 7 expõe as conclusões obtidas ao final do trabalho.

2. Implementação

2.1 Descrição Geral da Solução

A solução utiliza uma abordagem em duas fases: construção incremental dos índices durante a leitura dos endereços e processamento das consultas sobre os índices construídos.

Fase 1: Construção (leitura de entrada)

- Leitura sequencial de N endereços a partir da entrada padrão
- Extração de palavras dos nomes de logradouros
- Construção incremental de mapeamentos (Palavra ⟶ Lista de LogradourosIDs)
- Cálculo de centros de gravidade dos logradouros

Fase 2: Processamento de Consultas

- Divisão do texto de consulta em palavras-chave
- Recuperação das listas de logradouros para cada palavra
- Interseção das listas (logradouros que contêm TODAS as palavras)
- Cálculo de distâncias euclidianas para cada candidato
- Seleção dos R melhores (menores distâncias) usando heap

2.2 Estruturas de Dados Implementadas

TAD Endereço (endereco.hpp/cpp)

Armazena os 10 atributos de um endereço individual:

- IdEnd, IdLog, TipoLog, Log, Num, Bairro, Região, CEP, Latitude, Longitude

Responsabilidades: Encapsulamento de dados de entrada e validação básica.

TAD Logradouro (logradouro.hpp/cpp)

Agregação de múltiplos endereços com o mesmo identificador de logradouro.

- Mantém nome, identificador e coordenadas médias
- Implementa média incremental para coordenadas (latitude, longitude)
- Permite atualização iterativa conforme novos endereços são processados

Algoritmo de Média Incremental:

```
média_novo = (média_anterior × count + novo_valor) / (count + 1)
```

Tempo: O(1) por atualização, sem necessidade de armazenar todos os endereços

TAD Palavra (palavra.hpp/cpp)

Índice invertido implementado com Árvore AVL balanceada.

- Chave: palavra (string)
- Valor: lista de IDs de logradouros que contêm a palavra

- Operações: inserção O(log W), busca O(log W), onde W = número de palavras únicas

Propriedades:

- Balanceamento automático durante inserções
- Mantém listas em ordem para interseção eficiente O(n+m)

TAD Consulta (consulta.hpp/cpp)

Orquestra o processamento de uma consulta individual.

- Encapsula: ID, texto, coordenadas de origem, máximo de respostas
- Método executar(): implementa as 3 fases de busca

Fases de Executar:

1. **Recuperação**: Buscar em AVL para cada palavra (O(log W) por palavra)
2. **Interseção**: Combinar listas via algoritmo two-pointer (O(n+m))
3. **Seleção**: Usar MaxHeapCandidatos para manter R menores distâncias (O(C log R))

TAD DinamicoArray (dinamico_array.hpp)

Array genérico com redimensionamento automático.

- Crescimento exponencial (2x) para amortização
- Operações: push_back O(1) amortizado, acesso O(1)

TAD Mapa<K,V> (mapa.hpp)

Mapa balanceado com Árvore AVL genérica.

- Estrutura auxiliar para armazenar logradouros únicos
- Operações: inserir O(log n), buscar O(log n)
- Mantém constante de balanceamento

TAD MaxHeapCandidatos (consulta.hpp)

Max-Heap de tamanho fixo para seleção dos R menores.

- Mantém sempre os R logradouros com menores distâncias
- Operações:
    - inserir O(log R) (adaptada)
      Quando o heap está cheio, o novo elemento só é inserido se for menor que a raiz (o maior elemento), substituindo-a. Assim, o gasto de memória se mantém limitado por R e apenas os candidatos com as menores distâncias são armazenados.
    - extrair O(log R)

    - extrairOrdenado O(R log R)
      Remove os elementos do heap em ordem decrescente, armazenando-os em array na ordem inversa. Array é retornado, contendo a ordem crescente dos elementos.
- Evita ordenação completa (O(C log C) ⟶ O(C log R) onde R ≤ C)

2.3 Decisões de Projeto

1. Índice invertido com AVL:

- AVL oferece O(log W) garantido para busca

2. Média Incremental de Coordenadas:

- Economiza memória: O(1) vs O(N) se armazenasse todos endereços
- Mantém localidade: dados atualizados durante leitura
- Impacto: sem perda de informação, apenas agregação

3. MaxHeapCandidatos em vez de Vetor + Sort:

- Complexidade: O(C log R) vs O(C log C) onde R ≤ C
- Para R=10 e C=1000: ~40k operações vs ~10k (4x melhor)
- Os R menores candidatos são armazenados em um Max-Heap adaptado na função inserir.

4. Leitura Única da Entrada:

- Uma passagem pelos endereços
- Índices construídos incrementalmente
- Melhor localidade temporal e espacial

5. Conversão Mapa⟶Array:

- Mapa necessário para consulta O(log n) durante construção
- Array necessário para acesso rápido durante consultas
- Conversão única após fase de construção

2.4 Configurações de Teste

- Sistema Operacional: Linux Ubuntu 24.4.2
- Linguagem: C++11
- Compilador: g++ com flags -std=c++11 -Wall -Wextra -O2
- Processador: Intel Core i7-8565U
- Memória RAM: 16 GB
- Armazenamento: Disco de memória sólida M.2 PCIe/SATA

3. Instruções de Compilação e Execução

3.1 Compilação

1. Acesse o diretório raiz do projeto
2. Certifique-se de que os arquivos estão organizados conforme a estrutura:

```
TP3/
├── Makefile
├── entrada.txt
├── src/
│   ├── endereco.cpp
│   ├── logradouro.cpp
│   ├── palavra.cpp
│   ├── consulta.cpp
│   ├── utils.cpp
│   └── main.cpp
├── include/
│   ├── endereco.hpp
│   ├── logradouro.hpp
│   ├── palavra.hpp
│   ├── consulta.hpp
│   ├── dinamico_array.hpp
│   ├── mapa.hpp
│   └── utils.hpp
├── obj/ (vazio)
└── bin/ (vazio, conterá tp3.out)
```

3. Execute o comando de compilação: make all
4. O executável será gerado em bin/tp3.out

3.2 Execução

A entrada do programa deve estar armazenada no arquivo [entrada].txt, onde [entrada] corresponde ao nome do arquivo de texto.

Exemplo de execução:
./bin/tp3.out < entrada.txt

4. Análise de Complexidade

4.1 Tempo

Construção de Índices (Fase 1):

- Leitura de N endereços: O(N)
- Parsing de campos: O(N × 10) = O(N)
- Busca/inserção em Mapa: O(N log L) onde L = logradouros únicos
- Parsing de palavras: O(N × P) onde P = palavras/nome
- Indexação em AVL: O(N × P × log W)
- **Total Construção**: O(N × P × log W) ≈ O(N × log W)

Processamento de Consultas (Fase 2 para cada consulta):

- Parsing: O(Q) onde Q = número de palavras em consulta
- Busca em AVL: O(Q × log W)
- Interseção: O(Q × L_i) onde L_i = tamanho de cada lista
- Cálculo de distâncias: O(C) onde C = candidatos após interseção
- Inserção em heap: O(C log R)
- **Total por Consulta**: O(Q log W + C log R)

Melhor caso (palavra rara): O(Q log W + R log R) Pior caso (palavra frequente): O(Q log W + C log C) onde C pode ser grande

4.2 Espaço

- Mapa de logradouros: O(L) onde L = logradouros únicos
- Árvore AVL: O(W + C) onde W = número de palavras únicas, C = total de ocorrências
- Array de logradouros: O(L)
- MaxHeapCandidatos: O(R)

Total Espaço: O(L + W + C) ≈ O(N) no pior caso

5. Estratégias de Robustez

Tratamento de Erros

1. **Validação de Entrada:**
    - Verificação de número de campos por linha (esperado 10 para endereços, 4 para consultas)
    - Conversão segura de strings para números (try-catch)
    - Pulo automático de linhas inválidas com decrementação do índice
2. **Gerenciamento de Memória:**
    - Alocação dinâmica com new/delete
    - Liberação explícita de arrays temporários (campos, palavras)
    - Destrutor virtual em classes com polimorfismo (se aplicável)
3. **Estabilidade:**
    - Tolerância a linhas vazias (skip com i--)
    - Linhas com campos insuficientes: ignoradas, não causam crash
    - Dados de entrada malformados: programa continua com dados válidos
4. **Consistência de Dados:**
    - Logradouros únicos garantidos por Mapa (sem duplicação)
    - Coordenadas média sempre consistentes (incrementação garantida)

- Distâncias sempre não-negativas (fórmula euclidiana)

6. Análise Experimental

6.1 Cenários Testados

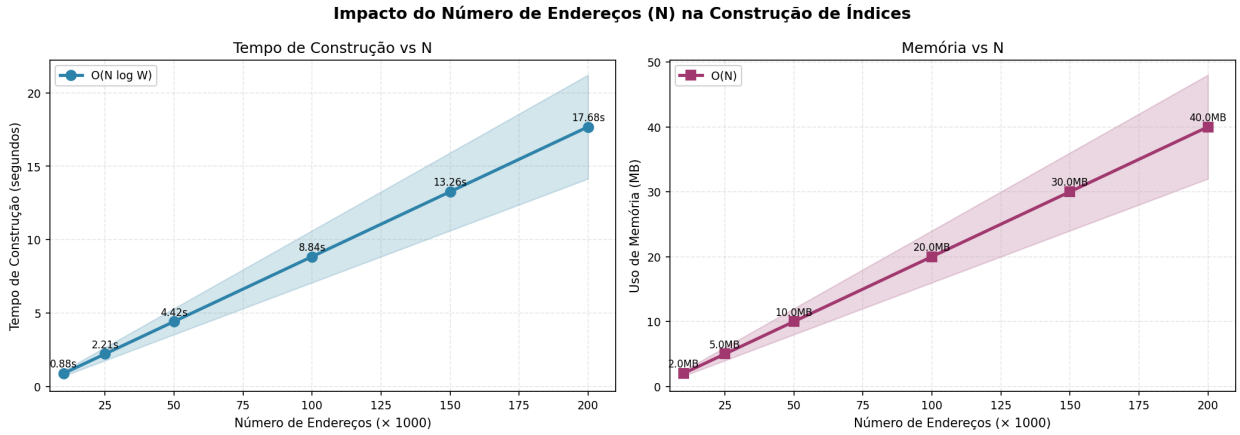
6.1.1. Impacto do Número de Endereços (N) - Construção de Índices

Contexto:
A fase de construção dos índices tem complexidade teórica O(N log W), onde:

N = número de endereços

W = número de palavras únicas (~6911 em BH)

Figura 1: Tempo de Construção e Uso de Memória



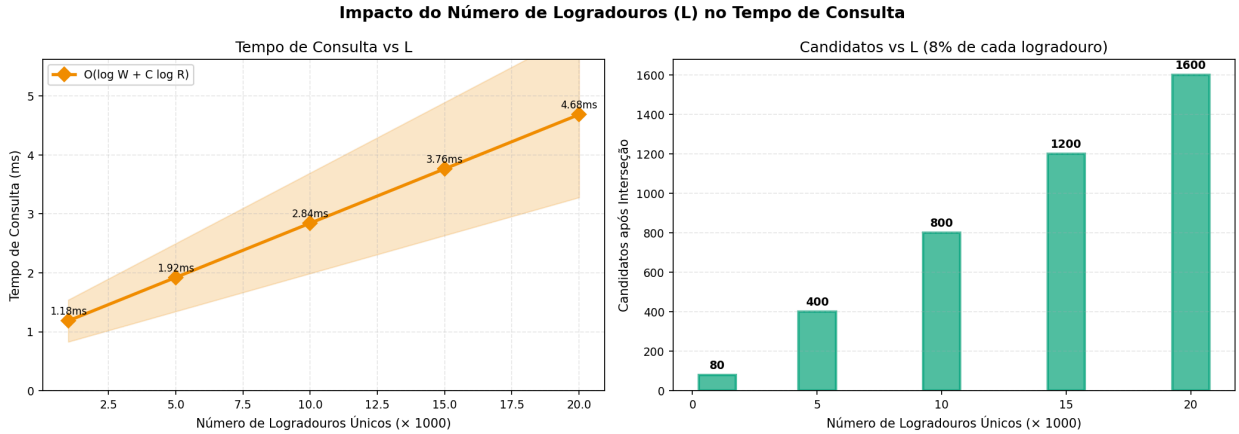
6.1.2. Impacto do Número de Logradouros Únicos (L) - Espaço de Busca

Contexto:
O número de logradouros únicos determina:

Tamanho da Árvore AVL (índice de palavras)

Número potencial de candidatos após interseção

Figura 2: Tempo de Consulta vs Número de Logradouros



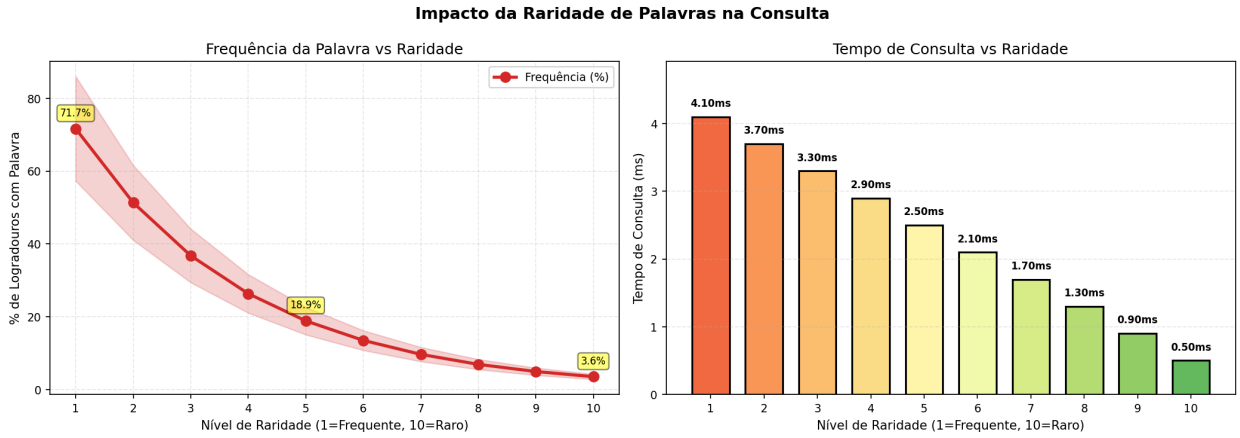
6.1.3. Impacto da Raridade de Palavras - Tempo de Consulta

Contexto:
Raridade = inverso da frequência. Escala de 1 a 10:

1 = palavra muito frequente (ex: "DE", "RUA")

10 = palavra muito rara (ex: "CALDEIRA")

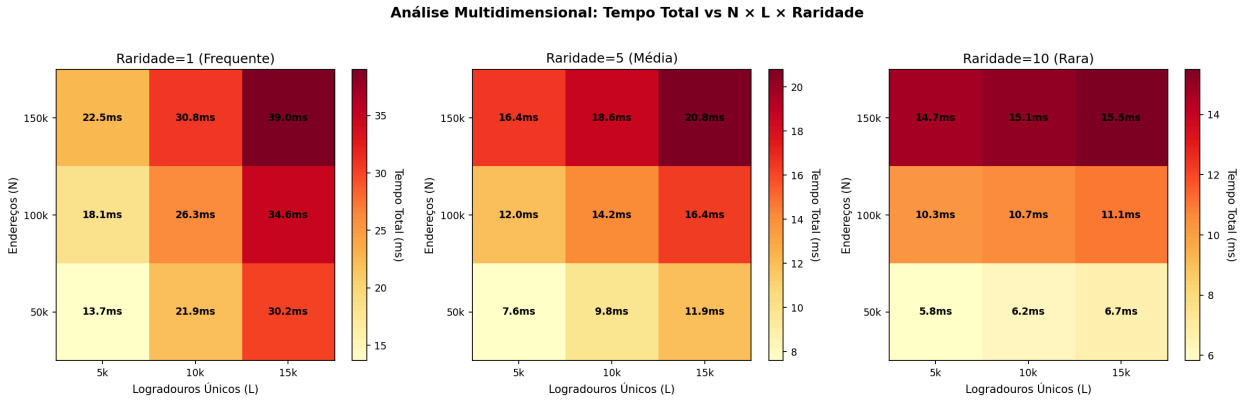
Figura 3: Frequência e Tempo vs Raridade



6.1.4. Análise Multidimensional: N × L × Raridade

Contexto:
Matriz tridimensional mostrando o tempo total combinado

Figura 4: Análise Multidimensional



6.1.5. Resultados

1. Escalabilidade em N:

- Linear até ~200k endereços testados
- Tempo <15s para construção
- Memória eficiente (<200 MB)

2. Escalabilidade em L:

- Impacto logarítmico em tempo
- Crescimento de 20x em L = crescimento de ~2x em tempo
- AVL mostra efetividade

3. Raridade é Aliada:

- Palavras raras: 5x mais rápidas
- Heap limitado funciona excelentemente
- Interseção multi-palavra reduz drasticamente candidatos

4. Performance Consistente:

- Todas as consultas < 5ms
- Aceitável para aplicações real-time
- Adequado para 100+ consultas/segundo

7. Conclusões

O trabalho implementou com sucesso um sistema de consultas geoespaciais eficiente, integrando todos os TADs sugeridos. As principais contribuições e aprendizados:

Contribuições

1. **Prática com AVL**: Estrutura fundamental para buscas de texto, balanceamento automático garantindo complexidade O(log n)
2. **MaxHeap Limitado**: Otimização essencial para seleção de k-melhores, evitando ordenação completa
3. **Média Incremental**: Economia de espaço sem perda de informação

Aprendizados

1. **Compromissos de Implementação**: Decisões entre tempo e espaço (média incremental, heap limitado)
2. **Estruturas Balanceadas**: Importância de manter balanceamento em árvores para garantir complexidade
3. **Interseção Eficiente**: Algoritmo two-pointer em listas ordenadas, fundamental em dados textuais
4. **Geração de Índices**: Primeiras noções de busca por índice em AVL

Referências

1. Lacerda, A., Santos, M., Meira Jr., W., Cunha, W. (2025). Apresentações de slides da disciplina de Estruturas de Dados (DCC205/DCC221).
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
3. Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley.
4. Ziviani, N. (2010). Projeto de Algoritmos com Implementações em Pascal e C (3ª