Análise de Complexidade - USEI08: Pesquisa por Área Geográfica

Logistics on Rails - Grupo 023

Resumo

Este documento apresenta a análise detalhada da complexidade algorítmica da implementação da user story USEI08 - Pesquisa por Área Geográfica. A solução implementa uma KD-Tree balanceada para pesquisas espaciais eficientes em estações ferroviárias europeias, demonstrando complexidades temporais e espaciais ótimas para operações de pesquisa por intervalo em grandes conjuntos de dados.

Conteúdo

1 Introdução

A user story USEI08 requer a implementação de um sistema de pesquisa espacial por área geográfica para estações ferroviárias europeias. A solução desenvolvida utiliza uma KD-Tree balanceada que permite pesquisas eficientes com filtros opcionais por país, tipo de estação (cidade/principal).

1.1 Características do Conjunto de Dados

- Aproximadamente 64.000 estações europeias
- Coordenadas geográficas (latitude, longitude)
- Metadados: país, fuso horário, flags (isCity, isMainStation)

2 Arquitetura da Solução

2.1 KD-Tree Balanceada

A estrutura central é uma KD-Tree 2D construída de forma balanceada utilizando as coordenadas (latitude, longitude) das estações.

```
public class KDTree {
    public static class Node {
        private final List<EuropeanStation> stations;
        private Node left, right;
        private final double latitude, longitude;
        private final int depth;
    }
}
```

Listing 1: Estrutura da KD-Tree

2.2 Mecanismo de Pesquisa Espacial

```
public record SpatialSearch(KDTree kdTree) {
    public List<EuropeanStation> searchByGeographicalArea(
        double latMin, double latMax,
        double lonMin, double lonMax,
        String countryFilter,
        Boolean isCityFilter,
        Boolean isMainStationFilter
    );
}
```

Listing 2: Interface do SpatialSearch

3 Análise de Complexidade Temporal

3.1 Construção da KD-Tree

Complexidade: $\mathbf{O(N \log N)}$

Justificação:

- Ordenação inicial por latitude e longitude: $O(N \log N)$
- Construção recursiva com partição por medianas: $O(N \log N)$
- Processamento de $O(N)$ elementos em cada nível de recursão

```
public void buildBalanced(List<EuropeanStation> stationsByLat,
                          List<EuropeanStation> stationsByLon) {
    // Ordena  o inicial: O(N log N)
    // Constru  o recursiva: O(N log N)
}
```

Listing 3: Algoritmo de Construção Balanceada

3.2 Pesquisa por Área Geográfica (Range Search)

Cenário	Complexidade	Justificativa
Melhor Caso	$O(\log N)$	Poda eficiente, região seletiva
Caso Médio	$O(\sqrt{N})$	KD-Tree balanceada em 2D
Pior Caso	$O(N)$	Área de pesquisa muito ampla

Tabela 1: Complexidade da Pesquisa Espacial

3.2.1 Análise Detalhada do Algoritmo de Pesquisa

```
private void searchInRangeRecursive(KDTree.Node node,
                                    double latMin, double latMax,
                                    double lonMin, double lonMax,
                                    String countryFilter,
                                    Boolean isCityFilter,
                                    Boolean isMainStationFilter,
                                    int depth,
                                    List<EuropeanStation> results) {

    if (node == null) return; // O(1)

    // Verifica  o do n  atual: O(1)
    boolean inLatRange = (currentLat >= latMin && currentLat <= latMax);
    boolean inLonRange = (currentLon >= lonMin && currentLon <= lonMax);

    if (inLatRange && inLonRange) {
        // Filtragem das esta  es no n  : O(K)
        for (EuropeanStation station : node.getStations()) {
            if (matchesFilters(station, countryFilter,
                isCityFilter, isMainStationFilter)) {
                results.add(station); // O(1)
            }
        }
    }

    // Poda baseada na dimens o atual
    int currentDimension = depth % 2;
    if (currentDimension == 0) {
        if (latMin <= currentLat) {
```

```
            searchInRangeRecursive(node.getLeft(), ...); // Recurs o
        }
        if (latMax >= currentLat) {
            searchInRangeRecursive(node.getRight(), ...); // Recurs o
        }
    }
    // L gica similar para longitude...
}
```

Listing 4: Algoritmo de Pesquisa Recursiva

3.3 Complexidade dos Filtros

```
private boolean matchesFilters(EuropeanStation station,
                               String countryFilter,
                               Boolean isCityFilter,
                               Boolean isMainStationFilter) {
    // Todas as opera  es: O(1)
    // Filtros nulos s o ignorados
    // Compara  o case-insensitive para pa s
}
```

Listing 5: Aplicação de Filtros

Complexidade: $\mathbf{O(1)}$ por estação

4 Análise de Complexidade Espacial

Componente	Complexidade
Armazenamento KD-Tree	$O(N)$
Stack de Recursão (caso médio)	$O(\log N)$
Resultados da Pesquisa	$O(K)$
Estruturas Auxiliares	$O(1)$

Tabela 2: Complexidade Espacial

5 Fatores que Influenciam o Desempenho

5.1 Balanceamento da Árvore

- **Altura ótima:** $O(\log N)$
- **Construção balanceada** garante desempenho consistente
- **Métrica de balanceamento:** $h \leq 2 \times \log_2(N)$

5.2 Eficiência da Poda

- **Redução do espaço de pesquisa** através de condições geométricas
- **Evita subárvores** quando a região de pesquisa não as interseta
- **Decisões baseadas** na dimensão atual (latitude/longitude)

5.3 Otimização de Buckets

- **Agrupamento** de estações com coordenadas idênticas
- **Redução** do número total de nós
- **Eficiência** em operações de filtragem

6 Validação Experimental

6.1 Métricas da Implementação

```
public String getComplexityAnalysis() {
    return String.format("""
        Propriedades da KD-Tree:
        - Altura: %d
        - N  s : %d
        - Balanceamento: %s

        Complexidade Temporal:
        - Melhor caso: O(log n)
        - Caso  m  d i o : O(  n  )
        - Pior caso: O(n)

        Complexidade Espacial:
        - Auxiliar: O(1)
        - Stack de recurs o: O(log n)
        """,
        kdTree.height(),
        kdTree.size(),
        kdTree.height() <= 2 * Math.log(kdTree.size()) / Math.log(2)
            ? "Bom" : "Pode ser melhorado");
}
```

Listing 6: Métricas de Desempenho

6.2 Resultados Esperados para 64k Estações

Operação	Complexidade	Tempo Esperado
Construção KD-Tree	$O(N \log N)$	$\sim$ 100-200ms
Pesquisa Seletiva	$O(\log N)$	$\sim$ 0.1-1ms
Pesquisa Média	$O(\sqrt{N})$	$\sim$ 1-10ms
Pesquisa Completa	$O(N)$	$\sim$ 10-100ms

Tabela 3: Desempenho Esperado (N = 64.000)

7 Conclusão

A implementação da USEI08 demonstra uma abordagem eficiente e escalável para pesquisas espaciais em grandes conjuntos de dados:

- **Desempenho Ótimo:** Complexidade sub-linear no caso médio ($O(\sqrt{N})$)

- **Escalabilidade:** Adequada para conjuntos de dados de grande dimensão
- **Flexibilidade:** Suporte a múltiplos filtros sem impacto significativo
- **Robustez:** Balanceamento automático garante desempenho consistente

A solução atende plenamente aos requisitos da USEI08, oferecendo tempos de resposta rápidos mesmo para o conjunto de dados completo de 64.000 estações europeias.

Anexos

A. Notação Assintótica Utilizada

- $O(N)$: Complexidade linear
- $O(\log N)$: Complexidade logarítmica
- $O(\sqrt{N})$: Complexidade sub-linear típica de KD-Trees 2D
- $O(N \log N)$: Complexidade log-linear