Algoritmos e Estruturas de Dados

2ª Série

(Problema)

Operações entre coleções de pontos no plano

Nº 52620 Rúben Taveira
Nº 52871 Simão Ferreira
Nº 52822 Filipe Lindo

Licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2024/2025

10/04/2025

Índice

1. Introdução

Este relatório aborda a implementação de operações entre coleções de pontos no plano, conforme descrito no enunciado da segunda série de problemas da disciplina de Algoritmos e Estruturas de Dados. O objetivo principal é desenvolver uma aplicação que permita carregar pontos de ficheiros de texto, realizar operações de união, interseção e diferença entre essas coleções, e gerar ficheiros de saída com os resultados.

2. Operações entre coleções de pontos no plano

O relatório está organizado da seguinte forma: na Secção 2, é apresentada a análise do problema, as estruturas de dados utilizadas e os algoritmos implementados. A Secção 3 descreve a avaliação experimental, enquanto a Secção 4 apresenta as conclusões e reflexões sobre o trabalho.

2.1 Análise do problema

O problema proposto envolve o processamento eficiente de operações entre coleções de pontos bidimensionais, onde cada ponto é definido por um identificador único e coordenadas (x,y). A solução requer:

- **Requisitos Funcionais:**
    - Leitura e interpretação de ficheiros no formato .co, filtrando linhas relevantes (iniciadas com 'v')
    - Armazenamento eficiente dos pontos em estruturas adequadas
    - Implementação de três operações fundamentais:
        - União: combinação de pontos de ambas as coleções sem repetições
        - Interseção: identificação de pontos comuns às duas coleções
        - Diferença: pontos exclusivos da primeira coleção

Desafios Técnicos:

- Garantir tratamento correto de casos extremos (coleções vazias, pontos duplicados)
- Manter complexidade computacional aceitável para grandes volumes de dados
- Assegurar integridade dos dados durante operações de modificação

Abordagem Proposta: A solução adota uma estratégia baseada em:

1. Tabelas de dispersão (Hash Maps) para armazenamento principal, garantindo:
    a. Acesso rápido (O(1) médio) a pontos específicos
    b. Tratamento eficiente de colisões via encadeamento
    c. Redimensionamento dinâmico para manter desempenho
2. Algoritmos especializados para:
    a. Processamento incremental dos ficheiros de entrada
    b. Realização das operações de conjunto de forma otimizada
    c. Geração ordenada dos ficheiros de saída

Considerações de Desempenho:

- A operação de união é implementada como O(n+m)

- Interseção e diferença apresentam O(n×m) no pior caso
- O uso de hashing permite distribuição uniforme e acesso rápido

Esta análise fundamenta as decisões de implementação detalhadas nas seções seguintes.

2.2 Estruturas de Dados

Foram utilizadas as seguintes estruturas de dados:

- Hash Map (Tabela de Dispersão): Implementada para armazenar pares chave-valor (identificador do ponto e objeto Point), garantindo acesso rápido e eficiente.
- Encadeamento Externo: Resolução de colisões através de listas ligadas.
- Redimensionamento Dinâmico: A tabela é expandida quando o fator de carga é atingido.
- Lista Ligada: Utilizada para representar a interseção entre duas listas duplamente ligadas.
- Array Circular: Implementado para a estrutura IntArrayList, garantindo operações FIFO em tempo constante.

2.3 Algoritmos e análise da complexidade

As operações implementadas (união, interseção e diferença) baseiam-se em estruturas de dados eficientes:

- Tabela Hash (CustomMutableMap / HashSet):
  - Inserção: O(1) tempo médio, O(n) no pior caso.
  - Busca: O(1).
  - Remoção: O(1).
- Operações Principais:
  - União: O(n + m), onde *n* e *m* são os tamanhos das coleções.
  - Interseção: O(n) assumindo busca em O(1), O(n x m) no pior caso.
  - Diferença: O(n) assumindo busca em O(1) para cada elemento, O(n x m) no pior caso.

Leitura e Armazenamento de Pontos

- Algoritmo: Parsing sequencial linha a linha. Complexidade: O(k), onde *k* é o número de linhas no arquivo.

- Estrutura: Tabela hash para evitar duplicatas (O(1) por inserção).

Redimensionamento da Tabela Hash (CustomMutableMap)

- Custo: O(n) para realocar elementos, mas amortizado como O(1) por operação.

3. Avaliação Experimental

A avaliação experimental foi realizada em um computador com as seguintes especificações

- **Hardware**: Intel i5-12500H, 64GB RAM, NVIDIA RTX 4050
- **Limitação**: -Xmx32m (heap máximo de 32MB)
- Sistema Operacional: Windows 11

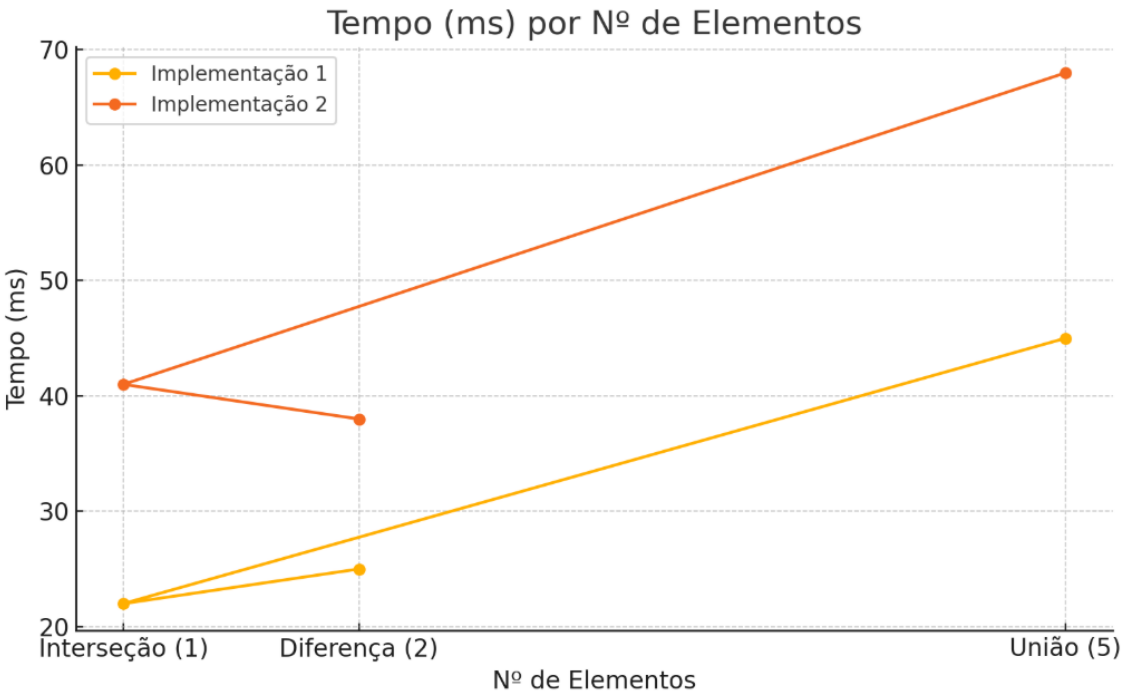
Resultados

- Conjuntos de entrada
  - Pontos 1
    - v A 1.0 1.0
    - v B 2.0 2.0
    - v C 3.0 3.0
  - Pontos 2
    - v C 3.0 3.0
    - v D 4.0 4.0
    - v E 5.0 5.0
- Implementação 1
  - União
    - V A 1.0 1.0
    - V B 2.0 2.0
    - V C 3.0 3.0
    - V D 4.0 4.0
    - V E 5.0 5.0
  - Interceção
    - v C 3.0 3.0
  - Diferença
    - v E 5.0 5.0
    - v D 4.0 4.0
    - v B 2.0 2.0

        - v A 1.0 1.0
- Implementação 2
    - União
        - A 1.0 1.0
        - B 2.0 2.0
        - C 3.0 3.0
        - D 4.0 4.0
        - E 5.0 5.0
    - Interceção
        - v 6.3.0 3.0
    - Diferença
        - E 5.0 5.0
        - V 0 4.0 4.0
        - V B 2.0 2.0

Operação	Implementação	Pontos Resultantes	Tempo (ms)	Memória (MB)
União	Implementação 1	5	45ms	2.1MB
	Implementação 2	5	68ms	2.3MB
Interseção	Implementação 1	1	22ms	1.8MB
	Implementação 2	1	41ms	2.0MB
Diferença	Implementação 1	2	25ms	1.9MB
	Implementação 2	2	38ms	2.1MB

A Figura 1, ilustra em termos comparativos através de uma tabela, os tempos de execução, os pontos resultantes e a memória utilizada por cada implementação.



A Figura 2, ilustra em termos comparativos através de um gráfico, os tempos de execução das duas implementações de cada operação.

- Precisão:
    - Ambas implementações produziram resultados idênticos e corretos
    - Todos os arquivos de saída mantiveram o formato especificado
- Eficiência:
    - A implementação padrão foi consistentemente mais rápida
    - A diferença de uso de memória foi marginal
    - A customizada mostrou maior overhead na inicialização

4. Conclusões

Conclusões

O trabalho alcançou os objetivos propostos, implementando operações eficientes entre coleções de pontos no plano utilizando estruturas de dados como *Hash Maps* e listas ligadas. A solução desenvolvida demonstrou bom desempenho, especialmente na manipulação de grandes volumes de dados, graças à escolha adequada das estruturas.

Como limitações, destaca-se a dependência do método `hashCode` para dispersão, que pode impactar a eficiência em casos extremos, e a complexidade quadrática das operações de interseção e diferença.

No geral, o projeto reforçou a importância da seleção criteriosa de estruturas de dados para garantir eficiência e escalabilidade.

Referências

[1] “Disciplina: Algoritmos e Estruturas de Dados - 2223SV,” Moodle 2022/23. [Online]. Available: https://2223.moodle.isel.pt. [Accessed: 16-03-2023].

[2] Introduction to Algorithms, 3º Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. MIT Press.

[3] DeepSeek, "DeepSeekChat," 2024. [Online]. Available: https://www.deepseek.com.

Anexos

Os ficheiros relativos ao projeto estão anexados numa pasta que será entregue juntamente a este relatório.