Universidade Federal do Rio de Janeiro

Implementação de Ordenação Externa via Merge-Sort

Charles Ribeiro Chaves - 122086950
Filipe Viana da Silva - 121050053
Vinícius Brasil de Oliveira Barreto - 120029237

6 de julho de 2025

Sumário

Resumo

Este documento apresenta uma análise técnica da implementação de um algoritmo de ordenação externa baseado no paradigma *merge-sort*, desenvolvido para ordenar arquivos CSV maiores que a memória RAM disponível. São descritas as estruturas de dados utilizadas, a divisão modular do código, as funções principais, complexidades computacionais, além de problemas e observações durante o desenvolvimento. Por fim, são discutidos os resultados obtidos.

1 Introdução

A ordenação externa é necessária quando o volume de dados a ser ordenado excede a capacidade da memória principal, exigindo o uso de memória secundária (disco). O algoritmo implementado divide o arquivo em blocos (chunks), ordena cada bloco na memória e realiza uma intercalação eficiente dos blocos ordenados até obter o arquivo final ordenado.

2 Estruturas de Dados Utilizadas

- **Listas**: usadas para armazenar temporariamente os registros (linhas do CSV) durante a ordenação interna dos chunks.
- **Heap (fila de prioridade)**: implementado via módulo `heapq` do Python, para realizar o merge multi-way eficiente dos arquivos temporários ordenados.
- **Arquivos temporários**: armazenam os chunks ordenados para processamento incremental e controle do uso da memória.

3 Divisão Modular e Descrição das Rotinas

O código está organizado em funções modulares com responsabilidades claras:

- `estimate_row_size`: estima o tamanho médio de uma linha para definir o tamanho do buffer.
- `merge_sort_internal` e `merge`: implementam o algoritmo *merge-sort* interno para ordenar os chunks na memória.
- `write_sorted_chunk`: ordena um chunk usando o merge-sort interno e grava em arquivo temporário.
- `merge_files_heap`: realiza o merge k-way de arquivos ordenados usando heap.
- `merge_chunks_multi_pass`: gerencia o merge multi-pass para escalabilidade, mesclando grupos de arquivos até restar um único arquivo final.
- `external_merge_sort_optimized`: rotina principal que orquestra o processo, gerenciando leitura, divisão, ordenação dos chunks e merge multi-pass.

4 Complexidade de Tempo e Espaço

- **Tempo**: A ordenação interna dos chunks tem complexidade $O(n \log n)$ para cada bloco, onde n é o número de linhas no chunk. O merge multi-way usando heap tem complexidade $O(N \log k)$, onde N é o número total de registros e k o número de arquivos sendo mesclados simultaneamente. O merge multi-pass reduz o número de arquivos progressivamente, mantendo a eficiência.

- **Espaço**: O uso de memória é limitado pelo tamanho do buffer configurado, garantindo que apenas um chunk caiba na memória por vez. O uso de arquivos temporários permite processar arquivos muito maiores que a RAM disponível.

5 Problemas e Observações

Durante o desenvolvimento, alguns desafios foram identificados:

- **Tratamento do cabeçalho**: foi necessário garantir que o cabeçalho do CSV fosse escrito apenas no arquivo final para evitar interferência na ordenação dos dados.

- **Conversão de tipos**: para ordenar corretamente chaves numéricas e textuais, implementou-se uma função de conversão que tenta transformar strings em números, melhorando a precisão da ordenação.

- **Gerenciamento de arquivos temporários**: a criação e remoção adequada dos arquivos temporários foi fundamental para evitar consumo excessivo de disco e garantir limpeza após a execução.

- **Escalabilidade**: a implementação do merge multi-pass permitiu que o algoritmo escalasse para grandes volumes de arquivos temporários, melhorando a eficiência do merge.

6 Conclusão

A implementação do algoritmo de ordenação externa via merge-sort demonstrou ser eficaz para ordenar arquivos CSV maiores que a memória RAM disponível, respeitando limitações de espaço e mantendo boa performance. A modularização do código facilitou a manutenção e possíveis extensões. O uso do merge multi-pass e da ordenação interna baseada em merge-sort garantiu escalabilidade e precisão. Os resultados obtidos mostram que a abordagem é viável para aplicações práticas que lidam com grandes volumes de dados.