Relatório: Ordenação Externa com Merge-Sort para Arquivos Grandes

Neste trabalho a ordenação foi feita por ID , porem pode ser alterado para idade ou no casos dos nomes ordem alfabetica .

1️⃣ Introdução Este trabalho tem como objetivo aplicar a técnica de Ordenação Externa para ordenar arquivos de registros (em formato CSV) cujo tamanho excede a memória RAM disponível, tornando inviável a utilização de funções padrão como sort() em listas carregadas integralmente na memória principal.

A abordagem utilizada é baseada no algoritmo Merge-Sort, adaptado para trabalhar com dados armazenados em memória secundária (HD ou SSD). Esta adaptação segue o princípio da divisão e conquista, dividindo o problema em blocos gerenciáveis na RAM e depois mesclando-os em disco.

2️⃣ Descrição do Problema Dado um arquivo CSV muito grande, precisamos ordená-lo por uma chave de ordenação específica (ex: ID, CPF).

A memória RAM não é suficiente para carregar todas as linhas de uma vez.

É necessário usar uma técnica que trabalhe com blocos parciais, salvando resultados intermediários em disco e realizando a fusão (merge) incremental.

3️⃣ Solução Adotada A solução implementa o Merge-Sort Externo em duas fases principais:

📌 1) Fase de Divisão O arquivo CSV é lido em blocos (chamados runs) com tamanho limitado pela RAM.

Cada bloco é ordenado em memória usando um algoritmo de ordenação interna (ex: Merge-Sort interno ou .sort() do Python).

Cada bloco ordenado é gravado como um arquivo temporário (ex: run_0.csv, run_1.csv).

📌 2) Fase de Merge Todos os runs ordenados são mesclados usando uma estratégia de heap (min-heap), que garante eficiência na busca do menor (ou maior) registro entre todos os arquivos.

O resultado do merge é gravado em um arquivo final único e ordenado (sorted_output.csv).

Após a fusão final, os arquivos temporários são removidos, liberando espaço.

4️⃣ Descrição do Algoritmo 🔹 Merge-Sort Clássico O Merge-Sort tradicional é um algoritmo de divisão e conquista:

Divide o conjunto de dados em metades recursivamente.

Ordena cada metade.

Combina (merge) as metades ordenadas em uma única sequência.

Complexidade: O(n log n).

🔹 Merge-Sort Externo Adapta o mesmo conceito, mas considerando que os dados não cabem na RAM.

A divisão é feita criando runs do tamanho que cabem na memória.

O merge combina registros de vários arquivos em disco de forma eficiente, minimizando leituras e escritas.

5️⃣ Implementação no Python (Google Colab) A implementação foi desenvolvida no ambiente Google Colab, o que permite:

Criar e gerenciar arquivos CSV de teste.

Executar o código Python diretamente em nuvem.

Baixar o arquivo final ordenado facilmente.

Principais passos do código:

create_initial_runs: lê o arquivo CSV em blocos, ordena cada bloco e cria arquivos temporários.

merge_runs: abre todos os runs simultaneamente, usa uma min-heap para intercalar as linhas ordenadas e gera o arquivo final.

merge_sort_external: função principal que orquestra as fases de divisão, merge e limpeza.

6️⃣ Testes Realizados ✔ Arquivo de teste criado: alunos.csv Exemplo:

Copiar Editar ID,Nome,Idade 3,Ana,20 1,Bruno,22 2,Carlos,21 5,Diana,19 4,Eduardo,23 ✔ Configuração de memória simulada: MAX_LINES_IN_MEMORY = 2 Para forçar a divisão do arquivo em múltiplos runs, mesmo com arquivo pequeno.

✔ Execução no Colab:

Gerado o CSV de teste.

Ordenação executada com a função merge_sort_external.

Verificado o sorted_output.csv:

Copiar Editar ID,Nome,Idade 1,Bruno,22 2,Carlos,21 3,Ana,20 4,Eduardo,23 5,Diana,19 ✔ Download do resultado realizado com files.download() no Colab.

7️⃣ Validação Foi confirmada a ordenação correta da chave (ID) em ordem crescente.

Testes com diferentes tamanhos de MAX_LINES_IN_MEMORY validaram o particionamento e o merge.

O algoritmo pode ser aplicado a datasets de qualquer tamanho, limitado apenas pelo espaço em disco.

8️⃣ Conclusão Este trabalho demonstrou como aplicar Ordenação Externa usando Merge-Sort, uma técnica essencial para processamento de grandes volumes de dados em ambientes onde a RAM é limitada.

A abordagem é amplamente utilizada em bancos de dados, processamento de logs, big data, entre outros. Além disso, o experimento mostrou a importância de entender o funcionamento de algoritmos clássicos e suas adaptações para sistemas de arquivos reais.

9️⃣ Referências Algoritmo Merge-Sort: Wikipedia – https://pt.wikipedia.org/wiki/Merge_sort

Documentação Python: módulos csv, heapq, os.

Google Colab: ambiente de execução Python em nuvem.

Foram incluidos ao codigo alguns prints do dataframe para verificar se o ocdigo estava fazendo o que era esperado.

📌 Anexos Código-fonte: ordenacao_externa.py

Arquivo de teste: alunos.csv

Saída gerada: sorted_output.csv

Davidson Diógenes Vasconcelos da Silva Santos

DRE: 123531495

Mônica de Sousa Amaral

DRE: 119160444

Naya da Silva Nascimento

DRE: 119160428