Relatório de Algoritmos e estrutura de dados 2

Alunos:

Ronaldo Pereira da Costa 20240514

João Gabriel Pepa Pereira 20240915

Introdução

Neste trabalho, utilizando a entrada fornecida pela professora, desenvolvemos um programa em C para realizar a ordenação externa de grandes volumes de dados. O processo foi dividido em duas etapas principais: a geração de runs (blocos ordenados menores) e a intercalação final desses blocos.

Divisão e ordenação dos dados

No arquivo main.c, chamamos inicialmente a função gerar_runs(). Essa função:

- Cria a pasta runs/, onde serão armazenados os arquivos runN.txt;
- Lê o arquivo de entrada, depois aloca memória para armazenar os blocos;
- Lê blocos de até 10.000 números por vez do arquivo entrada.txt;
- Ordena cada bloco na memória utilizando o algoritmo QuickSort;
- Salva cada bloco ordenado como um arquivo runN.txt separado.

Este método divide o arquivo de entrada em partes menores que cabem na memória, permitindo o ordenamento eficiente de arquivos muito grandes.

Intercalação em Fases

Devido ao limite do sistema operacional em relação ao número máximo de arquivos abertos simultaneamente, não foi possível abrir todas as runs de uma vez para realizar a intercalação final.

Para contornar esse problema, adaptamos a solução implementando uma intercalação em fases:

1. A função intercala_runs_em_fases() divide as runs geradas em grupos de até 1000 arquivos por fase.
2. Cada grupo é intercalado separadamente, e os resultados são armazenados em arquivos intermediários, salvos na nova pasta intermediarios/.
3. Por fim, todos os arquivos intermediários são abertos juntos para realizar a última fase da intercalação, gerando o arquivo final ordenado saida.txt.

Durante essa etapa, os primeiros números de cada arquivo são lidos e armazenados em vetores auxiliares. Esses valores são usados para construir uma **min-heap** com a função construir_heap(), garantindo que o menor valor fique na raiz. A cada iteração:

- O menor valor é retirado da heap e escrito no arquivo de saída;
- Um novo valor é lido da mesma run de onde veio o valor removido;
- A estrutura da heap é ajustada com a função heapify().

Esse processo se repete até que todos os arquivos tenham sido completamente processados.

Resultados e Análise

Realizamos testes com diferentes tamanhos de entrada. Ao gerar arquivos com até 10.000 números, o programa levou cerca de 3 minutos para criar e ordenar todas as runs e concluir a intercalação.

Em outro teste, utilizamos blocos menores de 1.000 números. A leitura e escrita dos blocos foi mais rápida, mas como a quantidade de runs geradas aumentou consideravelmente (cerca de 100.000 arquivos), o tempo total do processo também foi impactado. Mesmo sem executar a intercalação, a simples geração das runs já levou cerca de 5:30 minutos tendo um tempo total de aproximadamente 9:30.

Portanto, quanto menos runs são geradas, mais eficiente tende a ser o programa, devido à redução na carga de gerenciamento de arquivos.

Dificuldades Encontradas

Durante o desenvolvimento, enfrentamos diversas dificuldades, principalmente relacionadas à manipulação de arquivos em C, como criar diretórios, abrir e fechar arquivos corretamente, e lidar com o limite de arquivos abertos. Contornamos essas situações pedindo ajuda a alguns veteranos e por meio de outros estudos por conta de não termos visto este conteúdo ainda (conteúdo de prog 2).

Além disso, um dos maiores desafios foi visualizar a arquitetura geral do algoritmo. Embora tivéssemos noção sobre partes específicas, como ordenação e uso de heap, a integração de todas essas partes em uma solução aceitável e funcional exigiu tempo.

TESTES:

10.000 Números por aquivo
Criar Runs e ordenar: 3:00

1.000 Números por arquivo
Criar Runs e ordenar: 9:30