BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

BCC4001 - Algoritmos E Estruturas De Dados 2 (IC4A_CM)

Aluno: Luiz Gustavo Takeda

RA: 2504510

Aluno: Matheus Angelo Farias de Souza

RA: 2515709

Introdução

Para realizar a ordenação de um arquivo contendo registros randomizados em formato binário, com limitação de memória, foi implementado um algoritmo que divide o arquivo em blocos do tamanho permitido pela memória disponível. Esses blocos são ordenados utilizando o algoritmo quicksort, e posteriormente, os blocos ordenados são processados com o uso de buffers para controle de memória. A ordenação final do arquivo é realizada através do algoritmo de intercalação k-way merge, que combina os blocos ordenados em um arquivo final.

Divisão de trabalho

Luiz G. Takeda

Desenvolveu buffer de entrada, funções.

Matheus Angelo

Desenvolveu buffer de saída, testes e relatório.

Implementação

A estrutura do software foi desenvolvida de forma modular, com os seguintes componentes principais:

Utilitários: As funções utilitárias estão localizadas na pasta lib/utils. Elas encapsulam funcionalidades essenciais, como:

- removeSegments: Remove arquivos temporários P*.bin.
- getIndexNextMin: Retorna o índice do buffer com o menor elemento e verifica se todos os buffers estão vazios.
- segmentAndSort: Realiza a partição e ordenação do arquivo de entrada em arquivos temporários P*.bin usando quicksort, de acordo com a memória disponível, e retorna o número de blocos gerados.

Quicksort: A implementação do algoritmo quicksort está localizada na pasta lib/quick-sort.

Gerador de Arquivos: O gerador de arquivos com registros aleatórios está na pasta lib/file-generator. A estrutura ITEM_VENDA foi modificada para permitir comparações necessárias durante a execução do código.

Buffers: A implementação dos buffers de entrada e saída está na pasta lib/file-buffer. Ambos os buffers foram desenvolvidos utilizando templates, permitindo o uso de diferentes tipos de dados. Os principais métodos incluem:

Buffer de Entrada (FileBufferIn):

- storeNextBlock: Método privado para consumir o próximo bloco de dados e alterar o estado para vazio caso não haja mais dados.
- viewNext:Retorna o próximo elemento mas não consome ele.
- getNext: Retorna o próximo elemento consumindo ele, caso seja o último elemento chama o storeNextBlock.
- isEmpty: Verifica se todos os dados foram consumidos.

- FileBufferIn: Aloca memória e abre arquivo, e consome os primeiros dados com storeNextBlock.
- ~FileBufferIn: Destroi o buffer, desalocando memória e fechando o arquivo.

Buffer de Saída (FileBufferOut):

- add: Adiciona itens ao buffer e, se o limite for atingido, grava os itens no arquivo.
- save: Grava quaisquer itens restantes no arquivo de saída.
- FileBufferOut: Aloca memória e abre/cria arquivo de saída.
- ~FileBufferOut: Destroi o buffer, desalocando memória e fechando o arquivo.

Comandos: As funções que encapsulam a lógica dos comandos disponíveis estão na pasta lib/commands e incluem:

- commandGenerateFile: Responsável pelo comando -g/generate, que cria um novo arquivo com registros aleatórios.

  Parâmetros:

  <caminho do arquivo+nome>: Especifica o caminho e o nome do arquivo que será gerado.

  <quantidade de registros>: Define o número de registros aleatórios que serão gerados.

  <seed>: Valor utilizado como semente para a geração dos registros aleatórios, garantindo a reprodutibilidade dos dados.

- commandConvertFileToTxt: Responsável pelo comando -r/read, que converte um arquivo de registros em formato binário para um formato legível em texto.

  Parâmetros:

  <caminho+nome do arquivo bin>: Especifica o caminho e o nome do arquivo binário de entrada que será convertido.

  <caminho+nome do arquivo de saída txt>: Define o caminho e o nome do arquivo de saída no formato de texto legível.

- commandSortFile: Responsável pelo comando -s/sort, que realiza a partição, ordenação e intercalação dos blocos, gerando um arquivo final ordenado.

  Parâmetros:

  <caminho+nome do arquivo de entrada>: Especifica o caminho e o nome do arquivo binário que será ordenado.

  <quantidade de bytes para usar>: Define a quantidade total de memória disponível..

  <quantidade de bytes para buffer de saída>: Especifica o tamanho do buffer de saída utilizado para gravação dos dados ordenados.

  <caminho+nome do arquivo de saída>: Define o caminho e o nome do arquivo de saída que será gerado após a ordenação.

Execução

Para compilar utilize a ferramenta makefile, executando o comando *make*, será gerado o arquivo executável em .\out\main.exe

Gerando arquivo exemplo

.\ out\main.exe -g .\out\data-256000.bin 256000 42

Ordenando arquivo exemplo

.\out\main.exe -s .\out\data-256000.bin 8388608 1048576 .\out\data-sorted-256000-B8MB-S8.bin

Visualizando dados exemplo

.\out\main.exe -r .\out\data-sorted-256000-B8MB-S8.bin \out\data-sorted-256000-B8MB-S8.txt

Resultados

A execução do software foi realizada em um ambiente Windows, utilizando uma máquina equipada com 16 GB de memória RAM, processador Intel i7 de 7ª geração, e um SSD. O uso do SSD impactou positivamente os resultados devido à sua alta velocidade de leitura e gravação. No entanto, possíveis variações no tempo de execução podem ter sido causadas por interferências do sistema operacional. A medição do tempo de execução foi realizada utilizando o script PowerShell "script.ps1".

Tempo De Execução Para Ordenar Arquivo com 256000 Registros

		S		
		B/8	**B/4**	**B/2**
B	**8388608 (8MB)**	1.889	1.574	1.580
	16777216 (16MB)	1.297	1.293	1.306
	33554432 (32MB)	1.274	1.324	1.336

Tempo De Execução Para Ordenar Arquivo com 512000 Registros

		S		
		B/8	**B/4**	**B/2**
B	**16777216 (16MB)**	3.644	3.173	3.189
	33554432 (32MB)	2.751	3.174	3.250
	67108864 (64MB)	2.938	3.213	3.199

Tempo De Execução Para Ordenar Arquivo com 921600 Registros

		S		
		B/8	**B/4**	**B/2**
B	**67108864 (64MB)**	6.256	5.754	7.095
	134217728 (128MB)	6.295	7.250	7.262
	268435456 (256MB)	6.502	7.250	7.760

Tempo De Execução Para Ordenar Arquivo com 1572864 Registros

		S		
		B/8	**B/4**	**B/2**
B	**67108864 (64MB)**	12.152	11.868	14.575
	134217728 (128MB)	11.620	13.911	14.391
	268435456 (256MB)	12.495	12.770	13.417

Análise dos Resultados

Embora algumas inconsistências nos dados possam ser atribuídas à interferência do sistema operacional, é possível identificar tendências gerais. À medida que o tamanho do arquivo aumenta, o tempo de execução também aumenta, embora não de forma linear. Em relação ao tamanho do buffer, a configuração S = B/8 tende a ser mais eficiente do que S = B/4 e S = B/2, conforme esperado.

É importante notar que, em geral, quanto maior o tamanho do buffer (B), mais rápida é a execução. No entanto, em alguns casos específicos, como para os conjuntos de dados de 921.600 e 1.572.864 registros, um buffer de 256 MB apresentou desempenho ligeiramente inferior ao de 128 MB na configuração S = B/8. Uma possível explicação para isso é a sobrecarga adicional na alocação de memória maior, onde o sistema operacional pode ter se tornado um gargalo.