UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Campus CAMPO MOURÃO

Relatório de Trabalho Final - ED2

Ordenação de Arquivos Grandes.

Aluno: Luís Felipe Galleguillos Campos RA:2304652

Aluno: Reginaldo Gregório de Souza Neto RA: 2252813

O código foi dividido em partes, assim como solicitado no enunciado do trabalho. Temos os buffers (entrada e saída), que são espaços de memória alocados dinamicamente, responsáveis pelo armazenamento temporário dos registros e seu transporte de um arquivo para o outro. O merge (K-way) é a parte responsável pela ordenação dos registros vindos de buffers de entrada, onde sempre é comparado o menor, e inserido no buffer de saída.

A maior dificuldade encontrada pela dupla se tratou do entendimento da proposta do trabalho, sendo melhor esclarecida em algumas horas de PAluno. Após isso, nós fizemos a implementação dos BUFFERS primeiramente, assim como sugerido, para que as implementações seguintes fossem mais fáceis. Uma vez que, para realizar as tarefas necessárias fora do escopo do buffers, nós utilizamos os próprios métodos disponíveis dentro deles para manipular os registros.

Utilizamos durante toda a execução do projeto a plataforma do GitHub para sincronizar os códigos e criar novas branchs a cada modificação e implementação.

Reginaldo: Buffer Saída, Merge (K-way) e o relatório.

Luís: Buffer Entrada, Split e correções necessárias no código. Vale ressaltar que a implementação foi feita concomitantemente e com muito diálogo entre a dupla, resultando em um trabalho em equipe sinérgico.

Por fim, a seguir se encontram as tabelas (em branco), pois não conseguimos executar os devidos testes sugeridos pelo professor. Fizemos nossos próprios testes de implementação, verificando cadas um dos passos entre o arquivo de entrada e de saída, porém acabamos demorando muito para conseguir resolver os últimos bugs encontrados e acabamos ainda com alguns probleminhas e não conseguindo entender muito bem o modo de executar o código com as especificações do trabalho, sendo necessário uma última conversa com um monitor ou com o próprio professor. Apesar destes problemas, ficamos muito satisfeitos com nosso desempenho e aprendizado.

Abaixo alguns dados do nossos testes:

Tamanho do arquivo de entrada: 1572865

Tamanho do arquivo de saída: 1572756

Lista dos 50 primeiros ids no arquivo final:

{

[id:0], [id:1], [id:2], [id:3], [id:4], [id:5], [id:6], [id:7], [id:8], [id:9], [id:10], [id:11], [id:12], [id:13],[id:14], [id:15], [id:16], [id:17], [id:18], [id:19], [id:20], [id:21], [id:22], [id:23], [id:24], [id:25], [id:26],[id:27], [id:28], [id:29], [id:30], [id:31], [id:32], [id:33], [id:34], [id:35], [id:36], [id:37], [id:38], [id:39],[id:40], [id:41], [id:42], [id:43], [id:44], [id:45], [id:46], [id:47], [id:48], [id:49]

}

		S		
		B/8	B/4	B/2
B	8388608 (8MB)			
	16777216 (16MB)			
	33554432 (32MB)			

Figure 3: Tempo De Execução Para Ordenar Arquivo com 256000 Registros

		S		
		B/8	B/4	B/2
B	16777216 (16MB)			
	33554432 (32MB)			
	67108864 (64MB)			

Figure 4: Tempo De Execução Para Ordenar Arquivo com 512000 Registros

		S		
		B/8	B/4	B/2
B	67108864 (64MB)			
	134217728 (128MB)			
	268435456 (256MB)			

Figure 5: Tempo De Execução Para Ordenar Arquivo com 921600 Registros

		S		
		B/8	B/4	B/2
B	67108864 (64MB)			
	134217728 (128MB)			
	268435456 (256MB)			

Figure 6: Tempo De Execução Para Ordenar Arquivo com 1572864 Registros

MODULARIZAÇÃO:

Buffers: Os buffers de entrada foram implementados com os métodos de criar um buffer, alocar espaço, ver qual é o próximo registro no “bico” do buffer, consumir esse registro, verificar se o buffer está vazio, assim como destruir a alocação do buffer.

Já o buffer de saída é apenas um, que possui os métodos de criação e alocação, inserção de registros no buffer que verifica se ele já está cheio para chamar o método de despejar esses registros dentro do arquivo final e possui também a destruição dos dados alocados na memória.

Kway: O k-way ou k-vias, foi implementado separadamente, sendo responsável por receber por parâmetro um vetor de buffers de entrada com seu conteúdo interno previamente ordenado e o local de memória do buffer de saída para que os registros possam ser acessados fora do escopo da função. Sua responsabilidade é verificar qual buffer de entrada possui o menor elemento em sua “ponta” para que esse seja acrescido no buffer de saída e posteriormente ser escrito no arquivo final.

Split: O split é responsável por separar o arquivo binário inicial em vários arquivos menores e ordenados internamente. Para isso ele utiliza o QuickSort, mecanismo que já foi explicado em aula anteriormente, que através de um partition organiza os itens maiores e menores que um determinado pivô dentro de um vetor. Fazendo uso da splitFile, o algoritmo lê um registro por vez do arquivo desejado, para que no final não haja “buffers quebrados”. Deste modo ele executa a cópia de todos os registros existentes no arquivo “grande” para os arquivos menores.