Universidade Federal
de Ouro Preto

ENYA LUÍSA GOMES DOS SANTOS 19.2.4201

NATHANN ZINI DOS REIS 19.2.4007

VITÓRIA MARIA SILVA BISPO 19.2.4109

RELATÓRIO - TRABALHO PRÁTICO II
ORDENAÇÃO EXTERNA

Resumo apresentado por exigência da disciplina BCC203 - Estrutura de Dados II, da Universidade Federal de Ouro Preto.
Professor: Guilherme Tavares Assis

INTRODUÇÃO

O trabalho prático consiste na implementação de três tipos de métodos de ordenação externa diferentes (1) Intercalação Balanceada de vários caminhos - 2F fitas, (2) Intercalação Balanceada de vários caminhos - F+1 Fitas, (3) QuickSort Externo, e os métodos (1) e (2) utilizam, na etapa de geração dos blocos ordenados, a técnica de seleção por substituição.

A partir da implementação desses métodos o objetivo consiste em verificar a eficiências desses realizando testes a partir de uma entrada de arquivo de texto nomeada, inicialmente, de “PROVAO.TXT” esse arquivo consiste em uma lista de alunos que realizaram o Provão no ano de 2003, e cada linha apresenta as informações dos alunos, sendo essas:

- Número de inscrição
- Nota (de 0.0 a 100.0)
- Estado (UF)
- Cidade
- Curso

Nos testes o objetivo é ordenar as ***n*** primeiras linhas do arquivo texto de entrada de forma **ascendente** por meio das **notas** de cada aluno, considerando ***n*** igual a **100, 1.000, 10.000, 100.000 e 471.705,** além disso, o arquivo de entrada possui três situações diferentes, **aleatório, ascendente e descendente**.

A partir dos testes foram obtidos resultados dos números de transferência, comparações e tempo de execução de cada método e são esses dados que serão analisados para verificar a eficácia de cada um desses métodos em diferentes situações.

Divisão de tarefas entre o grupo

O grupo contém três integrantes, tendo em vista isso, cada integrante ficou com um método diferente, a fim de agilizar o processo. Abaixo é apresentado a relação de implementação e o integrante que a implementou.

Intercalação Balanceada de vários caminhos (2F Fitas): Enya Luísa Gomes dos Santos.

Intercalação Balanceada de vários caminhos (F+1 Fitas): Vitória Maria Silva Bispo.

QuickSort Externo: Nathann Zini dos Reis.

Por mais que tenha ocorrido essa divisão, ainda sim, sempre que algum membro do grupo tinha dúvida sobre a implementação de um método, havia uma reunião para tentarmos resolver.

A parte de teste, análise final e construção do relatório foi feita em conjunto por todos os membros do grupo.

INTERCALAÇÃO BALANCEADA DE VÁRIOS CAMINHOS (2F FITAS)

Intercalação balanceada de vários caminhos 2F Fitas					
Transferências					
	100	1000	10000	100000	471705
Crescente	201	2001	20001	200001	943411
Decrescente	518	8090	110252	1100618	5190243
Aleatório	514	8131	111062	1410824	8069403
Comparações					
Crescente	287	2987	29987	303108	1434305
Decrescente	518	7060	89190	867248	3992381
Aleatório	881	12475	143414	1635431	8334398
Tempo (ms)					
Crescente	1,595	13,116	258,537	1187,946	6801,389
Decrescente	3,172	50,812	555,837	5151,415	24523,11
Aleatório	2,609	48,034	508,375	6003,102	33401,792

A implementação do método de 2f fitas foi realizada considerando a leitura, escrita e fitas em arquivo com formato .txt, visando a não conversão do arquivo texto original e a melhor visualização dos dados dentro das fitas. O método contém um total de 20 fitas, 10 fitas na entrada e 10 fitas na saída.

No arquivo de texto foi utilizado o **\0 seguido do \n** para indicar o fim de um bloco e fim de arquivo, por meio dessa sequência de caracteres é que a identificação do fim de bloco durante do processo foi feita.

Analisando os dados obtidos e expostos na tabela acima, de modo geral, obviamente, os valores de transferência e comparações aumentam de acordo com o valor de ***n***.

Realizando uma análise mais detalhada, a ordem **crescente** o número de transferências será sempre igual a duas vezes o valor da entrada + 1, onde, uma

vez será a leitura para a memória principal e a outra a escrita para a fita, já o + 1 consiste na escrita do caractere \0 seguido do \n para indicar fim do bloco, isso se dá ao fato de que, como nunca terá uma nota menor que a nota à ser retirada da heap não haverá mais que um bloco, logo tudo estará na primeira fita de entrada. Já na ordem **decrescente** ocorrerá o fato de que haverá muitos blocos na primeira fase de geração dos blocos, pois os dados ou serão iguais ou serão menores, fazendo assim a heap preencher com todos os dados marcados com mais frequência e criando novos blocos nas fitas. Os arquivos ordenados de forma **aleatória** apresentam maiores valores tanto de transferências quanto de comparações pois o número haverá maior geração de blocos, logo se faz necessário mais intercalações.

Na teoria, a ordem **decrescente** resultaria em piores resultados, mas nos casos de teste isso não ocorre, pois há muitas notas que se repetem, fazendo assim com que não seja criado muitos blocos.

INTERCALAÇÃO BALANCEADA DE VÁRIOS CAMINHOS (F+1 FITAS)

Intercalação balanceada de f+1 caminhos					
Transferências					
	100	1000	10000	100000	471705
Crescente	201	2001	20001	200001	943411
Decrescente	526	8091	80212	800529	5190094
Aleatório	526	8090	80570	1105584	6629763
Comparações					
Crescente	278	2978	29978	308632	1469919
Decrescente	718	9121	71466	691703	4145329
Aleatório	1165	17326	140811	1622191	8178905
Tempo (ms)					
Crescente	1,78	19,143	141,239	1175,435	6827,499
Decrescente	4,006	67,541	661,074	5283,891	32572,918
Aleatório	4,928	33,824	480,998	7019,46	41945,888

Para a implementação do método de intercalação balanceada de f+1 caminhos, a leitura e escrita de dados foi realizada em arquivos no formato ".txt" e por isso, os itens não foram lidos bloco por bloco, o que diminuiria a quantidade de transferências, aumentando assim a eficiência do algoritmo. Foi considerado que a memória interna disponível possuía capacidade para armazenar no máximo 19

registros e apenas 20 fitas de armazenamento externo, dessa forma ficam 19 fitas de entradas e 1 fita de saída.

Em relação à análise dos dados obtidos para o arquivo já ordenado de forma **ascendente**, é possível perceber que a quantidade de transferências é correspondente ao dobro do tamanho do arquivo + 1, isso se dá por conta da leitura dos dados do arquivo inicial “PROVAO_ASC.TXT” e transferência para a fita de entrada e o +1 é correspondente ao caractere que representa o fim de bloco/arquivo. Como o método foi implementado através de substituição por seleção, um arquivo totalmente ordenado faz com que nenhum registro seja marcado, resultando em apenas um bloco totalmente ordenado. Dessa forma, o processo de intercalação não ocorre, pois a condição de parada do método foi satisfeita.

Para o arquivo ordenado de forma **descendente**, o método entra em seu pior caso, que quase todos os elementos serão marcados, gerando vários blocos de tamanhos iguais, pois os dados do arquivo sempre serão menores ou iguais em relação ao número que saiu da heap. Para arquivos menores, é possível que não entre realmente no pior caso, pois o início do arquivo possui muitos iguais, fazendo que nem todos os elementos sejam marcados.

Para o arquivo **aleatório** há mais transferências e comparações do que o ordenado de forma **descendente**, pois como são valores aleatórios, há menos chances de terem tantos valores repetidos. Dessa forma, haverá uma maior geração de blocos, o que resulta em mais intercalações.

QUICKSORT EXTERNO

QuickSort Externo					
Transferências					
	100	1000	10000	100000	471705
Crescente	200	2000	20000	200000	943410
Decrescente	200	2000	20000	200000	943410
Aleatório	582	10952	151390	1774800	8747120
Comparações					
Crescente	3393	35343	354843	4046159	19094564
Decrescente	4051	40811	405875	4050393	19097304
Aleatório	5304	68402	779838	8607981	43478012
Tempo (ms)					
Crescente	0,389	5,232	38,137	335,685	1521,979

Decrescente	0,664	11,721	37,102	361,522	1656,683
Aleatório	2,609	19,564	248,631	2590,89	13430,857

Para a realização do QuickSort externo, pelo funcionamento do código exemplo disponibilizado pelo professor Guilherme, fez-se necessário a conversão do arquivo “PROVAO.TXT” em um arquivo binário “provao.dat”.

A análise, para fins de comparação, não levou em consideração a conversão dos arquivos, visto que os outros dois métodos são feitos em cima do arquivo TXT e não em arquivo binário. Desta forma, os dados obtidos de desempenho são referentes única e exclusivamente ao funcionamento dos métodos em si.

É observado que, nos casos dos arquivos de entrada ordenados tanto ascendentemente quanto descendentemente, a quantidade de transferências é sempre o dobro dos dados, pois todos os dados são trazidos para memória principal e depois salvos novamente no arquivo, sendo, portanto, transferidos 2x o mesmo dado. No caso do arquivo inicial aleatório, as transferências são maiores devido às partições que são geradas no arquivo, logo, um dado pode ser transferido várias vezes do arquivo para a memória interna e vice e versa.

As comparações, entretanto, variam de acordo com a estrutura de dados utilizada para guardar os valores em memória principal. O que foi utilizado neste trabalho foi um vetor ordenado ascendentemente. Por isso as comparações têm um valor mais alto, porque para analisar, por exemplo, a quantidade de itens em memória interna, tem-se que percorrer todo o vetor e comparar o valor para saber se é uma posição vazia ou não. Além de que para retirar o último, tem-se que verificar em todas ass posições a última válida. Em relação à ordenação inicial do arquivo, percebe-se que o arquivo ordenado ascendentemente tem os melhores resultados com as menores quantidade de dados tratados, porém, ao aumentar essas quantidades, a quantidade de comparações se iguala ao dos arquivos ordenados descendentemente. O Arquivo inicialmente desordenado aleatoriamente tem o pior resultado.

CONCLUSÃO

É notório que, embora o custo computacional de comparações e transferências com o arquivo inicial desordenado aleatoriamente no método 3,

QuickSort Externo, seja maior se comparado com os demais métodos, o tempo de execução do código é menor, devido ao terceiro método ser executado dentro de um mesmo arquivo, enquanto os demais utilizam de vários arquivos diferentes, representando as fitas de entrada e de saída. A manipulação de diferentes arquivos aumenta consideravelmente o tempo de execução do código. Em relação ao arquivo inicial ordenado tanto ascendente quanto descendente, o custo de transferência dos dados é menor se comparado aos dos demais métodos. Já as comparações feitas pelo terceiro método são sempre maiores que as feitas nos demais.

Em relação às transferências e ao tempo, o método 3 é mais eficiente do que os métodos 1 e 2 em ordem ascendente e descendente. Para arquivos ordenados de forma aleatória, o método 1, intercalação balanceada de vários caminhos 2f, é mais eficiente em termos de custo computacional de comparações e transferências nos tamanho de n igual a 100 e 1000 em relação ao método 2, intercalação balanceada de f+1 caminhos. Já nos demais valores de n o contrário ocorre, ou seja, o método de f+1 caminhos se torna mais eficiente para arquivos com um maior volume de dados.

Dificuldades

No geral, a maior dificuldade foi administrar o tempo para fazer o trabalho prático visto que estamos no final do período e há muitas tarefas a serem feitas, tanto da cadeira em questão quanto das demais cadeiras do curso.

Nos dois primeiros métodos, a maior dificuldade foi pensar em um algoritmo do “zero” para ser implementado, pois não havia nenhum algoritmo modelo a se seguir.

O terceiro método já foi um pouco simples mais fácil de implementar pois já havia um algoritmo disponibilizado pelo professor, a maior dificuldade encontrada foi desenvolver uma estrutura de dados para manipulação dos dados na memória interna.

No método 2, ocorreu um *bug* de leitura de caracteres do arquivo de texto, um simples fscanf(), no qual funcionava no método 1 o mesmo trecho de código, por esse motivo dava erro na ordenação de dados de determinados tamanhos de ***n*** e foi difícil encontrar onde exatamente ocorria o erro pois como dito no método 1 funcionava.