UNIVERSIDADE FEDERAL DE OURO PRETO – UFOP

Ciência da Computação

ESTRUTURA DE DADOS II

RELATÓRIO ORDENAÇÃO EXTERNA

Carlos Gabriel de Freitas

Daniel Monteiro Valério

Gabriel Mace do Santos Ferreira

Marcus Vinícius Souza Fernandes

Ouro Preto

2021

Introdução

O problema abordado consiste na implementação de métodos de ordenação externa, a fim de ordenar crescentemente um arquivo texto que corresponde a uma base de dados dos alunos que realizaram o “Provão” em 2003.

A fim de verificar a eficácia dos métodos foram realizados testes para a ordenação de 100, 1000, 10.000, 100.000 e 471.705 itens do arquivo. Além disso, os métodos foram executados em arquivos ordenados de forma crescente, decrescente e aleatória.

Os testes realizados nos métodos implementados foram transcritos para tabelas, as quais contêm as médias do tempo de execução, bem como as do número de comparações e do número de transferências de leitura (utilização de funções *fscanf* e *fread*) e de escrita (utilização de funções *fprintf* e *fwrite*).

É importante ressaltar que os resultados obtidos a partir dos métodos de ordenação externa serão gerados e armazenados no arquivo “resultado.txt”.

Vale lembrar que, ao adicionar o argumento opcional [-P] ao final da linha de execução, todos os dados presentes no arquivo serão impressos no formato “(número de inscrição do aluno) (nota do aluno) (sigla do estado do aluno) (cidade do aluno) (curso do aluno)”, antes e após a realização do método de ordenação externa escolhido.

Desenvolvimento

Método 1: Intercalação balanceada de vários caminhos (2f fitas), utilizando ordenação interna na etapa de geração de blocos

	Tempo de Execução		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	0,00 ms	15,625 ms	15,625 ms
1000	62,5 ms	78,125 ms	31,25 ms

10.000	109,375 ms	140,625 ms	96,75 ms
100.000	437,5 ms	375,0 ms	343,75 ms
471.705	2546,875 ms (2s)	2109,375 ms (2s)	3078,125 ms (3s)

	Comparações		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	905	930	1075
1000	25.053	25.160	26.484
10.000	430.527	430.895	444.055
100.000	5.665.613	5.666.175	5.808.351
471.705	31.478.535	31.478.010	32.124.723

	Transferências de Leitura / Escrita		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	200 / 200	200 / 200	200 / 200
1000	3000 / 3000	3000 / 3000	3000 / 3000
10.000	40.000 / 40.000	40.000 / 40.000	40.000 / 40.000
100.000	400.000 / 400.000	400.000 / 400.000	400.000 / 400.000
471.705	2.358.530 / 2.358.530	2.358.530 / 2.358.530	2.358.530 / 2.358.530

Durante a implementação da intercalação balanceada de vários caminhos (2f fitas), utilizando um método de ordenação interna na etapa de geração de blocos, foi necessário a criação das fitas utilizando arquivos binários, visto que o método de quicksort externo, que foi implementado anteriormente à esse, também faz uso de arquivos binários. Isto foi feito para manter uma certa padronização na leitura e escrita do arquivo durante a ordenação e na impressão da análise dos mesmos.

É possível observar que o número de transferências de leitura e de escrita são iguais, uma vez que cada aluno presente no arquivo será lido uma vez do

arquivo original “PROVAO.txt” ou das fitas de entrada e escrito uma vez nas fitas de saída ou no arquivo “resultado.txt” a cada passada realizada pelo algoritmo.

Segundo os dados da tabela, foi possível deduzir que esse método de ordenação externa é mais eficiente quando o arquivo original já se encontra ordenado da forma desejada (neste caso, como o método ordena os dados de forma crescente, ele será mais rápido quando o arquivo já se encontra ordenado crescentemente). Isso se deve ao fato de que não será necessário ordenar os registros, ocasionando um menor número de intercalações.

Método 2: Intercalação balanceada de vários caminhos (2f fitas), utilizando seleção por substituição na etapa de geração de blocos

	Tempo de Execução		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	0,00 ms	0,00 ms	0,00 ms
1000	46,875 ms	78,125 ms	46,875 ms
10.000	171,875 ms	250 ms	265,625 ms
100.000	2000 ms (2s)	2078,125 ms (2s)	2453,125 ms (2s)
471.705	8937,5 ms (8s)	9578,125 ms (9s)	10640,625 ms (10s)

	Comparações		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	5102	10.702	11.592
1000	56.402	117.505	146.185
10.000	569.402	1.163.608	1.591.016
100.000	5.762.996	10.663.916	17.489.224
471.705	27.282.579	52.362.053	87.915.044

	Transferências de Leitura / Escrita		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**

100	100 / 100	200 / 200	200 / 200
1000	1000 / 1000	3000 / 3000	3000 / 3000
10.000	10.000 / 10.000	30.000 / 30.000	30.000 / 30.000
100.000	100.000 / 100.000	300.000 / 300.000	400.000 / 400.000
471.705	471.705 / 471.705	1.886.820 / 1.886.820	23.588.525 / 23.588.525

Assim como no método visto anteriormente, durante a implementação da intercalação balanceada de vários caminhos (2f fitas), utilizando seleção por substituição na etapa de geração de blocos, foi necessário a criação das fitas utilizando arquivos binários para manter uma certa padronização na leitura e escrita do arquivo durante a ordenação e na impressão da análise dos mesmos.

A partir dos dados coletados é possível observar que o número de operações (leitura e escrita) realizadas no arquivo aumentam de acordo com o modo como se encontra a ordenação desse, dado que um arquivo ordenado crescentemente possui um número de operações equivalente a quantidade de itens ordenados, enquanto os arquivos ordenados de forma decrescente e aleatória possuem um maior número de operações.

Esta observação se deve ao fato de que é necessária uma movimentação maior dos dados entre as fitas e a memória interna na situação em que o arquivo “PROVAO.TXT” se encontra em uma ordenação diferente daquela desejada (neste caso, as ordenações decrescente e aleatória).

Método 3: Quicksort Externo

	Tempo de Execução		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	0,00 ms	0,00 ms	0,00 ms
1000	15,625 ms	15,625 ms	93,75 ms
10.000	109,375 ms	171,875 ms	640,625 ms
100.000	1265,625 ms (1s)	1296,875 ms (1s)	11296,875 ms (11s)
471.705	7921,875 ms (7s)	6765,625 ms (6s)	64171,875 ms (64s)

	Comparações		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	5933	7341	8372
1000	64.883	76.483	116.555
10.000	654.383	757.969	1.331.869
100.000	7.543.131	7.554.219	15.309.961
471.705	35.607.570	35.612.842	77.519.650

	Transferências de Leitura / Escrita		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	100 / 100	100 / 100	197 / 197
1000	1000 / 1000	1000 / 1000	4637 / 4637
10.000	10.000 / 10.000	10.000 / 10.000	70.122 / 70.122
100.000	100.000 / 100.000	100.000 / 100.000	876.750 / 876.750
471.705	471.705 / 471.705	471.705 / 471.705	4.403.215 / 4.403.215

Durante a implementação do quicksort externo foi necessário a conversão do arquivo "PROVAO.TXT” para um arquivo binário, visto que o valor do deslocamento no arquivo texto (função *fseek*) variava a cada iteração da partição, produzindo resultados errôneos.

Podemos observar que, para o quicksort externo, o tempo de execução, número de comparações e os números de transferências de leitura e escrita são semelhantes quando o arquivo está ordenado de forma crescente ou decrescente, sendo estes valores muito maiores quando o arquivo está ordenado aleatoriamente.

Também podemos observar que os números de transferências de leitura e escrita são iguais à quantidade de alunos no arquivo que serão ordenados (correspondendo ao melhor caso do algoritmo), sendo a situação do arquivo estar ordenado aleatoriamente a exceção dessa observação.

Conclusão

Ao compararmos os resultados obtidos nos métodos 1 e 2 e supondo um número grande de registros a serem ordenados, podemos observar que o método 1 é melhor que o método 2 em todos os quesitos (tempo de execução, número de comparações e de transferências de leitura e escrita) em quase todas as situações, exceto em número de comparações e transferências quando o arquivo já está ordenado e no número de transferências quando o arquivo está ordenado de maneira contrária à desejada. Vale lembrar que o número de fitas utilizado foi relativamente alto (40 fitas totais, sendo 20 de entrada e 20 de saída), e deste modo, é seguro afirmar que o método 1 é mais eficiente que o método 2.

De posse dos resultados obtidos nos métodos 1 e 3, podemos observar que o método 1 também é melhor que o método 3 em todos os quesitos, exceto em número de transferências de leitura e de escrita quando o arquivo a ser ordenado já se encontra ordenado de modo crescente ou decrescente. Isto ocorre pois as operações realizadas no método 1 ocorrem entre múltiplos arquivos (fitas de entrada e fitas de saída), enquanto no método 3 ocorrem *in situ,* ou seja, no mesmo arquivo. Além disso, no método 3, é necessário fazer a ordenação dos sub-arquivos pelas chamadas recursivas presentes no algoritmo.

A partir dos dados coletados no método 2 e no método 3, é possível observar que, na situação onde o arquivo se encontra previamente ordenado:

- crescentemente, o método 3 possui um menor tempo de execução que o método 2, mas um maior número de comparações, enquanto ambos possuem o mesmo número de transferências de leitura e de escrita;
- decrescentemente, o método 3 possui um menor tempo de execução, um menor número de comparações, de transferências de leitura e de escrita que o método 2;
- aleatoriamente, o método 3 possui um maior tempo de execução, mas um menor número de comparações, de transferências de leitura e de escrita que o método 2;

Apesar da complexidade de um método de ordenação ser medido através do número de transferências realizado nele, o tempo de execução do método 2 é drasticamente menor (cerca de 6 vezes menor, considerando 471 mil registros) que o do método 3 na situação em que o arquivo se encontra ordenado aleatoriamente. Vale lembrar que, no funcionamento do método 2, os elementos são ordenados na memória interna pela estrutura heap. Dessa forma, ao serem transferidos para as fitas, os registros se encontram ordenados e, portanto, reduzem o tempo necessário para as demais etapas do algoritmo, consequentemente levando a um menor tempo de execução.

Em uma situação real onde a ordenação externa se torne necessária, um número muito grande de registros em um arquivo desordenado (ordenado aleatoriamente) implicaria em uma disparidade ainda mais drástica no tempo de execução, tornando assim o método 2 mais favorável que o 3.

De posse das comparações efetuadas anteriormente, podemos afirmar então que, em uma situação real, o método 1 é o mais eficiente dentre todos eles, e o método 3 é o menos eficiente.

Durante a implementação do primeiro método, houve dificuldade em relação a determinar o número de registros contidos em cada bloco. A fim de solucionar tal problema, foi adicionado um ponteiro a struct fita para receber um vetor que representa a quantidade de registros em um bloco.

Durante a implementação do segundo método, ocorreram dificuldades quanto à escrita em um arquivo, após a ordenação do vetor pelo heap. Com o intuito de identificar uma solução para o problema foi contatada a tutora, e decidido que deveria ser adicionada uma condição para a situação em que o ponteiro do arquivo fosse nulo (NULL).

Durante a implementação do terceiro método, o grupo enfrentou diversas dificuldades. Primeiramente foi necessário manter a formatação da inscrição de um aluno, bem como a de sua nota (ex: 00465579 e 075.9, respectivamente, ao invés de 465579 e 75.9). Isso foi remediado através das funções *completaInscricao* e *completaNota*, que fazem uso das funções *contaDigitosLong* e *contaDigitosDouble* (funções que contam quantos dígitos o número n passado por parâmetro possui),

respectivamente, para então imprimir adequadamente zeros à esquerda e padronizar a saída.

Um erro de lógica na implementação da função *RetiraUltimo* acarretou diversas falhas no output do método. Este erro foi solucionado ao decrementar o contador após o laço de repetição *while*, o qual passou a indicar o último aluno presente na área, em vez da primeira posição vazia (aluno fictício com inscrição -1) presente nela.

Além disso, durante a ordenação do arquivo texto a função que percorria o arquivo (*fseek*) apresentava valores variados a cada chamada recursiva da função *QuicksortExterno*, portanto o grupo decidiu converter o arquivo “PROVAO.TXT” para binário, de forma a tornar mais simples o caminhamento dos ponteiros no arquivo. Ao final do método, o arquivo binário é convertido de volta para um arquivo texto, denominado “resultado.txt”.