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

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 quatro tipos de métodos de pesquisa externa diferentes (1) acesso sequencial indexado, (2) árvore binária de pesquisa adequada à memória externa, (3) árvore B e (4) árvore B*, e comparar dados como, número de transferências, número de comparações e tempo de execução, dos diferentes métodos e com diferentes tamanhos de arquivos.

Divisão de tarefas entre o grupo

Inicialmente, para o primeiro método - acesso sequencial indexado - foi feito em conjunto, ao mesmo tempo, usando a ferramenta do VS Code, Live Share. Já para os outros métodos ocorreu a divisão, cada um ficou responsável por desenvolver um método. Abaixo há a relação entre os métodos e quem implementou.

Árvore binária de pesquisa adequada à memória externa: Enya L. G. dos Santos

Árvore B: Nathann Zini dos Reis

Árvore B*: Vitória Maria Silva Bispo

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

Resultados obtidos

Acesso sequencial indexado

Pré-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	8	8	8
2.000	77	77	77
20.000	770	770	770
200.000	7693	7693	7693
2.000.000	76924	76924	76924

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0	0	0
2.000	0	0	0
20.000	0	0	0
200.000	0	0	0
2.000.000	0	0	0

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	0.0000ms
2.000	0.0000ms	0.0000ms	0.0000ms
20.000	15.650ms	46.8750ms	31.2500ms
200.000	234.3750ms	187.5000ms	234.3750ms
2.000.000	1968.7500ms	1859.3750ms	2015.6250ms

Pós-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	1	1	0
2.000	1	1	0
20.000	1	1	0
200.000	1	1	0
2.000.000	1	1	0

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	22	25	0
2.000	51	40	0
20.000	610	201	0
200.000	5927	1800	0
2.000.000	605	72206	0

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	0.0000ms
2.000	0.0000ms	0.0000ms	0.0000ms
20.000	0.0000ms	0.0000ms	0.0000ms
200.000	0.0000ms	0.0000ms	0.0000ms
2.000.000	0.0000ms	0.0000ms	0.0000ms

Esse método tem o pré-processamento não muito demorado, e a pesquisa muito rápida. A desvantagem desse método é que, os dados precisam estar ordenados para funcionar de forma satisfatória, pois caso contrário o dado não será encontrado na pesquisa.

Árvore binária de pesquisa externa

Pré-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	643	643	2359
2.000	6163	6163	47347
20.000	61363	61363	874513
200.000	613363	613363	22700339
2.000.000	6133363	6133363	651036945

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	417	210	3738
2.000	4014	2010	81457
20.000	40017	20010	1583281
200.000	400017	200010	43415897
2.000.000	4000017	2000010	1279031577

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	15.6250ms
2.000	62.5000ms	62.5000ms	343.7500ms
20.000	650.0000ms	609.375ms	7656.2500ms
200.000	6203.1250ms	5968.7500ms	215796.8750ms
2.000.000	62656.2500ms	63375.0000ms	5676015.6250ms

Pós-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	147	51	11
2.000	1050	948	7
20.000	15337	4661	13
200.000	153637	46361	22
2.000.000	15298	1876757	27

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	296	53	20
2.000	2102	950	12
20.000	30676	4663	21
200.000	307276	46363	36
2.000.000	30598	1876759	45

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	0.0000ms
2.000	0.0000ms	15.6250ms	0.0000ms
20.000	62.5000ms	15.250ms	0.0000ms
200.000	562.5000ms	171.8750ms	0.0000ms
2.000.000	46.8758ms	7031.2500ms	0.0000ms

Por meio dos dados obtidos, pode-se perceber que o maior gasto de tempo está no pré-processamento, isso devido a quantidade de transferência que ocorre

entre dois arquivos, um sendo o de dados e o outro da própria árvore binária externa, e também entre a memória principal.

Outro ponto a se destacar é que, os arquivos com os dados já ordenados, crescente e decrescente, demoram menos, em questão de tempo, do que os aleatórios, esse fato se dá pela implementação do código, pois quando os dados já estão ordenados e inserção do item na árvore é mais simples, o pai não precisa ser encontrado percorrendo a árvore, pois ele sempre será o item acima do filho, basta inserir na esquerda ou na direita, o que resulta em uma complexidade constante. Já no método aleatório a cada inserção de um novo item, a árvore é percorrida desde o seu início até encontrar o pai do novo item, tendo complexidade logarítmica no pior caso.

Já na pesquisa o aleatório se sobressai no tempo, pois sua complexidade no pior caso é logarítmica e para crescente e decrescente a complexidade é linear.

Por fim, como esperado, o número de comparações, transferências e tempo aumenta de acordo com a quantidade de dados do arquivo e também depende do item a ser pesquisado, pois ele pode estar mais no início do arquivo ou mais para o final, tal fator também irá influenciar nos resultados das medidas.

Árvore B

Pré-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	15	15	15
2.000	135	135	135
20.000	1335	1335	1335
200.000	13335	13335	13335
2.000.000	133335	133335	133335

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	3792	2431	2782
2.000	57292	35665	41487
20.000	781040	481243	560376
200.000	9888730	6068917	7002682
2.000.000	119835486	73217455	85254358

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	0.0000ms
2.000	15.6250ms	15.6250ms	0.0000ms
20.000	171.8750ms	156.250ms	125.000ms
200.000	1781.2500ms	1593.7500ms	1375.0000ms
2.000.000	26343.7500ms	16484.3750ms	17062.5000ms

Pós-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0	0	0
2.000	0	0	0
20.000	0	0	0
200.000	0	0	0
2.000.000	0	0	0

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	9	7	6
2.000	11	13	11
20.000	13	12	13
200.000	30	18	18
2.000.000	21	19	20

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	0.0000ms
2.000	0.0000ms	0.0000ms	0.0000ms
20.000	0.0000ms	0.0000ms	0.0000ms
200.000	0.0000ms	0.0000ms	0.0000ms
2.000.000	0.0000ms	0.0000ms	0.0000ms

Com base nos dados obtidos, é observado que o maior tempo gasto e, logo, custo operacional gasto na execução do método da Árvore B se encontra no pré-processamento, uma vez que, na proposta do nosso trabalho, a árvore é toda montada em memória principal para então pesquisar um item dela. Essa pesquisa, por sua vez, que é evidenciada nos dados de pós-processamento, é feita de maneira extremamente rápida e com pouquíssimo custo operacional.

Devido à configuração do computador no qual foram realizados os testes, todos os dados dos arquivos couberam em memória principal, porém, dependendo da limitação do computador em que esse código for executado, não será possível carregar todos os arquivos.

Árvore B*

Pré-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	14	14	14
2.000	134	134	134
20.000	1334	1334	1334
200.000	13334	13334	13334
2.000.000	133334	133334	133334

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	4399	3598	2782
2.000	63584	47907	48490
20.000	844317	604457	627562
200.000	10521955	7302107	7795986
2.000.000	126168730	85550611	92444742

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	0.0000ms
2.000	15.6250ms	15.6250ms	0.0000ms
20.000	187.5000ms	187.500ms	156.2500ms
200.000	2109.3750ms	1796.8750ms	1578.1250ms
2.000.000	23984.3758ms	21453.1250ms	17687.5000ms

Pós-processamento:

Transferência			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0	0	0
2.000	0	0	0
20.000	0	0	0
200.000	0	0	0
2.000.000	0	0	0

Comparações			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	13	13	11
2.000	18	21	17
20.000	22	23	25
200.000	30	30	31
2.000.000	34	33	29

Tempo			
Quantidade	**Crescente**	**Decrescente**	**Aleatório**
200	0.0000ms	0.0000ms	0.0000ms
2.000	0.0000ms	0.0000ms	0.0000ms
20.000	0.0000ms	0.0000ms	0.0000ms
200.000	0.0000ms	0.0000ms	0.0000ms
2.000.000	0.0000ms	0.0000ms	0.0000ms

Através da análise dos dados obtidos, é possível perceber que o pré-processamento da árvore B* é a etapa mais demorada. Isso ocorre por causa da quantidade de comparações que o algoritmo deve realizar para encontrar o local

para inserir o registro de forma que as restrições da árvore B* sejam respeitadas. Já no pós-processamento, a pesquisa ocorre de forma muito rápida e com poucas comparações, quanto menor for a quantidade de registros no arquivo, menor é a quantidade de comparações.

Analogamente ao que acontece no método da árvore B, todos os dados dos arquivos couberam em memória principal, porém, dependendo da limitação do computador em que esse código for executado, não será possível carregar todos os arquivos.

Conclusão

Para arquivos com dados ordenados de forma crescente ou decrescente o melhor método, de modo geral, de pesquisa externa é o método de acesso sequencial indexado, sua desvantagem está presente na utilização de arquivos com dados dispostos de forma aleatória, nesse cenário o método não é funcional.

Já o método de árvore binária de pesquisa externa se mostra mais custoso, só não tendo um valor tão discrepante em relação aos outros no tempo de pós-processamento. A vantagem desse método é que, como os dados estão em outro arquivo, não há muito consumo da memória principal, nem no momento de pré -processamento nem no pós-processamento.

Os métodos de árvore B e árvore B* apresentaram resultados semelhantes entre si, e são os melhores métodos para casos com dados do arquivo dispostos de forma aleatória.