UNIVERSIDADE FEDERAL DE OURO PRETO – UFOP

Ciência da Computação

ESTRUTURA DE DADOS II

RELATÓRIO PESQUISA EXTERNA

Carlos Gabriel de Freitas

Gabriel Mace do Santos Ferreira

Marcus Vinícius Souza Fernandes

Ouro Preto

2021

Introdução

O problema abordado consiste na implementação de diferentes métodos de pesquisa em memória externa, de forma a encontrar dados escolhidos pelo usuário em um arquivo binário organizado de forma crescente, decrescente ou aleatória.

Na execução do programa, a montagem do método de pesquisa externa e das estruturas utilizadas é conhecida como pré-processamento, e é a parte mais demorada. Em contraste, a pesquisa do item em si é quase instantânea, na maioria dos métodos.

A fim de verificar a eficácia dos diferentes métodos implementados foram feitos testes com arquivos de diferentes tamanhos e critérios de ordenação. Vale lembrar que, ao adicionar o argumento opcional [-P] ao final da linha de execução, todas as chaves presentes no arquivo serão impressas no formato “Chave #i: (chave do registro i)”, onde i é o índice do registro.

Os testes realizados nos métodos implementados foram transcritos para tabelas, as quais contêm as médias do tempo de execução total, bem como as do número de comparações e do número de transferências.

Desenvolvimento

Acesso Sequencial Indexado (ASI)

	Tempo de Execução		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	1,56 ms	0,00 ms	0,00 ms
1000	1,56 ms	0,00 ms	3,12 ms
10.000	15,65 ms	10,93 ms	18,75 ms
100.000	107,81 ms	167,18 ms	103,12 ms
1.000.000	1231,25 ms (1s)	1943,75 ms (2s)	1214,06 ms (1s)

	Comparações		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	12	10	12509
1000	77	59	12507
10.000	728	533	12502
100.000	7227	5284	12513
1.000.000	72221	52790	12509

	Transferências		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	15	14	15
1000	127	126	127
10.000	1252	1251	1252
100.000	12502	12501	12502
1.000.000	125002	125001	125002

Percebemos que, independente da ordenação dos registros no arquivo, o Acesso Sequencial Indexado possui valores baixos para todos os quesitos analisados: tempo de execução, número de comparações e número de transferências.

Os testes para o arquivo ordenado decrescentemente foram realizados após os do arquivo ordenado aleatoriamente e crescentemente. Deste modo, a memória cache da máquina pode ter influenciado nos resultados do tempo de execução, acelerando-os, sendo este o motivo do tempo para mil registros em ordem decrescente ter sido menor que o tempo para cem registros em ordem decrescente.

Árvore Binária

	Tempo de Execução		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	14,26ms	17,18ms	9,37ms

1000	967,18ms (1s)	1207,81ms (1s)	90,62ms
10.000	105840,62ms (106s)	183373,43ms (183s)	1325,00ms (1s)
100.000	11667343,75ms* (3h)	15765484,37ms* (4h)	18717,18ms (19s)
1.000.000	116673437,50ms** (32h)	157654843,70ms** (44h)	348704,68ms (349s)

	Comparações		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	5007	4994	649
1000	50077	499924	11203
10.000	50000683	49999224	157284
100.000	704991995*	705073414*	2004961
1.000.000	7049919950**	7050734140**	24858245

	Transferências		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	5406	5348	1048
1000	504076	503923	15502
10.000	50040775	50039223	197283
100.000	705391994*	705473413*	2404960
1.000.000	7053919940**	7054734130**	28458244

***** Valores de um único teste realizado com 100 mil registros

****** Valores estipulados para os testes com quantidade de 1 milhão de registros

Podemos observar que o tempo de execução, o número de comparações e o número de transferências aumenta exponencialmente conforme a quantidade de registros presentes no arquivo aumenta.

A partir dos testes utilizando 100, 1000 e 10 mil registros foi possível concluir que a criação da árvore binária necessitava de mais tempo para o processamento quando os dados estavam ordenados de forma crescente ou decrescente, visto que a árvore se encontrava desbalanceada, pendendo toda para a direita ou esquerda, respectivamente.

Dessa forma, foi necessário utilizar os dados de apenas dois testes realizados para um arquivo contendo 100 mil registros, um para o arquivo ordenado crescentemente e outro decrescentemente, os quais possuem em torno de 3h e 4h de tempo de execução, respectivamente.

Também foi necessário induzir os resultados obtidos ao utilizar arquivos com 1 milhão de registros, por meio da comparação dos resultados de arquivos com uma menor quantidade de registros. Supondo que o tempo de execução utilizando 1 milhão de registros seja 10 vezes maior que o tempo de execução utilizando 100 mil registros, realizar os 10 testes propostos apenas em arquivos ordenados crescentemente resultaria em torno de 13 dias.

Árvore B

	Tempo de Execução		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	0,00ms	0,00ms	0,00ms
1000	0,00ms	0,00ms	0,00ms
10.000	31,25ms	31,25ms	32,81ms
100.000	279,68ms	285,93ms	296,87ms
1.000.000	2910,93ms (3s)	2935,93ms (3s)	3578,12ms (4s)

	Comparações		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	1664	1395	1466
1000	26863	20296	21740

10.000	372491	265290	292054
100.000	4773657	3272822	3659114
1.000.000	58178839	38905635	43914598

	Transferências		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	101	101	101
1000	1001	1001	1001
10.000	10001	10001	10001
100.000	100001	100001	100001
1.000.000	1000001	1000001	1000001

É possível inferir que, independente da ordenação em um arquivo contendo 1 milhão de registros, a árvore B pode ser considerada rápida, conseguindo pesquisar um item em torno de 4 segundos ou menos.

Outro ponto interessante desse método de pesquisa externa é que, independente da ordenação presente no arquivo, o número de transferências é exatamente igual à quantidade de registros presentes (os quais serão lidos do arquivo e inseridos na estrutura) mais um (pesquisa do item propriamente dita).

Árvore B*

	Tempo de Execução		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	0,00ms	0,00ms	0,00ms
1000	4,68ms	3,12ms	4,68ms
10.000	39,06ms	34,37ms	35,93ms
100.000	576,56ms	546,87ms	553,12ms
1.000.000	6575,00ms (7s)	5510,93ms (6s)	6960,93ms (7s)

	Comparações		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	1967	1457	1657
1000	30308	20957	23964
10.000	408291	271948	310642
100.000	5133796	3339478	3859484
1.000.000	61878454	39572288	45982280

	Transferências		
Quantidade	**Crescente**	**Decrescente**	**Aleatória**
100	101	101	101
1000	1001	1001	1001
10.000	10001	10001	10001
100.000	100001	100001	100001
1.000.000	1000001	1000001	1000001

O tempo de processamento da árvore B* cresce junto com o número de registros em um arquivo dado que o custo para a criação das estruturas que compõem a árvore é proporcional aos registros recebidos. Ademais, o alto número de comparações pode ser fundamentado no pré processamento da árvore, bem como na pesquisa do item desejado posteriormente.

Conclusão

O Acesso Sequencial Indexado é o mais rápido dentre todos os métodos de pesquisa externo implementados, possuindo o menor tempo de execução que os demais em todos os casos. Vale lembrar que ele não consegue encontrar o registro se o arquivo estiver ordenado aleatoriamente, portanto, neste quesito, é indispensável utilizar outro método em aplicações reais.

Um outro ponto importante do ASI é que seu pré-processamento (construção do índice de páginas) é quase instantâneo e a pesquisa é demorada, diferentemente dos outros métodos, em que o pré-processamento é demorado e a pesquisa é quase instantânea.

A árvore B é a solução para o problema da árvore binária de Muntz Uzgalis. Por meio dos testes realizados é possível afirmar que a prerrogativa é verdadeira dado que o tempo para o processamento da árvore B é drasticamente reduzido. Além disso, é possível afirmar que por permitir que a árvore cresça de forma balanceada há menos comparações dado que a árvore não pende para um dos lados ao utilizar um arquivo crescente ou decrescente.

A árvore B* apresenta um tempo de execução maior se comparada a árvore B, devido a sua estrutura que exige a criação de uma árvore B composta das chaves de seus registros, enquanto os registros em si são delegados as folhas da árvore. Por conseguinte também é possível deduzir que seu maior número de comparações em relação a árvore B é causado durante o pré-processamento.

Um fator interessante é que o número de transferências da árvore B* é igual ao da árvore B: independente do método de ordenação, será igual a quantidade de registros presentes no arquivo mais um, visto que é necessário fazer a leitura de cada registro do arquivo e inseri-los na árvore e posteriormente fazer a pesquisa do item desejado.

Durante a realização do trabalho prático, o grupo teve diversas dificuldades. Primeiramente, o método de ASI não estava funcionando para arquivos ordenados decrescentemente, pois não havia o devido tratamento de mudar o tipo de comparação com o índice de páginas segundo a ordenação do arquivo. O método também ocasionava uma falha de segmentação caso a quantidade de registros fosse superior a mil, pois a constante MAXTABELA não tinha sido atualizada para um valor que permitisse armazenar um número maior de índices (que é igual a quantidade de registros dividido pelo número de itens por página).

Na implementação dos métodos da árvore binária, árvore B e árvore B* houve grande dificuldade na interpretação dos métodos na criação da árvore e

posteriormente inserção dos registros na estrutura em questão, acarretando grandes empecilhos na implementação do código.

Ainda houve outra dificuldade durante a implementação da árvore B, visto que a variável n, responsável por contar o número de registros presentes na página, não estava sendo inicializada com 1, quando ocorria o aumento da altura da árvore (quando o processo de divisão atingia a página raiz).