Universidade Federal de Ouro Preto – UFOP
Curso: Ciência da Computação
Disciplina: BCC203 – Estrutura de Dados II
Professor: Guilherme Tavares de Assis

TRABALHO PRÁTICO I - PESQUISA EXTERNA

João Gabriel Fernandes Zenóbio - 19.2.4065

Luiz Fernando Rodrigues Fernandes - 19.2.4008

Sthéphany Karoline Soares de Araújo Tezza - 20.1.4027

04 de novembro de 2021

Ouro Preto – Minas Gerais – Brasil

1. Introdução

Uma máquina capaz de realizar cômputos, também conhecida como computador, é formada por dois grandes grupos de memória: memória interna e externa. As memórias internas são de rápido acesso sendo conhecidas por serem voláteis, ou seja, os dados só persistem enquanto há alimentação energética. Já as externas caracterizam-se por conseguir armazenar as informações mesmo após o dispositivo ser desconectado da energia.

O processamento de dados da memória interna é muito mais rápido quando comparado à memória externa, tal fato deve-se ao modelo de acesso aos dados. A memória principal é uma RAM, ou seja, uma memória de acesso randômico, dessa forma permite o acesso direto aos dados a um custo uniforme. Ao contrário, externamente a máquina só consegue acessar os dados de modo sequencial, isto é, apenas um dado é acessado por vez.

O trabalho desenvolvido visa uma métrica para medir a complexidade de métodos de pesquisa externa. Como mencionado, a manipulação da memória externa é muito custosa, por isso será discutido técnicas que permitam que parte dos dados sejam manipulados em memória principal, melhorando a eficiência do processamento.

Dessa maneira, quatro métodos de pesquisa externa serão discutidos: acesso sequencial indexado, árvore binária de pesquisa externa, árvore B e árvore B*. Para tal, serão utilizados arquivos com número de itens variados e diferentes arranjos, podendo estar em ordem crescente, decrescente ou aleatório.

2. Desenvolvimento

2.1. Acesso sequencial indexado

2.1.1. Definição

Este método tem como princípio a busca sequencial de dados, ou seja, os itens são lidos um por vez até que a chave seja encontrada ou até que se encontre

o final de arquivo. Para sua implementação o arquivo a ser manipulado deve estar previamente ordenado, visto que é necessário a criação de uma tabela de índices.

A criação dessa tabela é representada por uma estrutura que armazena os menores ou maiores índices de cada página, dependendo da organização do arquivo, além disso, o número de itens de cada página deve estar definido. Por exemplo, para um arquivo ordenado crescentemente cada índice da tabela conterá a menor chave encontrada na página referente.

A pesquisa utiliza a tabela de índices para diminuir o número de transferências de dados da memória externa para a interna. Utilizando como exemplo um arquivo ordenado crescentemente, é realizada uma busca na tabela de índices, verificando se a chave contida em cada posição é menor que a chave desejada e se a chave da próxima posição é maior; esse procedimento é repetido até encontrar a página onde o item se encontra.

A transferência do arquivo é feita somente se encontrar a página correspondente na tabela índices. Por esse motivo há apenas uma transferência na busca, deslocando o ponteiro do arquivo até o item desejado e o carregando na memória principal.

2.1.2. Resultados obtidos

Os testes realizados utilizaram páginas com quatro itens.

- Os testes não foram realizados para arquivo ordenados aleatoriamente, uma vez que a montagem da estrutura tem como regra a ordenação previamente dos dados.

ORDENADO CRESCENTE				
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Transferências**
100	0s	0s	14,1	26
1.000	0,05s	0s	140,1	251
10.000	0,117s	0s	1.377	2501
100.000	1,245s	0s	13.752	25001
1.000.000				

ORDENADO DECRESCENTE				
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Transferências**
100	0s	0s	13,5	26
1.000	0,014s	0s	115,5	251
10.000	0,119s	0s	1.128	2501
100.000	1,298s	0s	12.003	25001
1.000.000				

2.2. Árvore binária de pesquisa externa

2.2.1. Definição

A árvore binária é uma estrutura comumente utilizada para pesquisa em memória principal, mas pode ser simulada em um arquivo para ser possível realizar uma busca binária entre registros através da estrutura formada no arquivo. A árvore binária é uma estrutura que relaciona um dado a mais dois recursivamente.

Dessa forma, o primeiro passo a ser realizado é a construção de um novo arquivo, adicionando dois novos campos que representam as posições dos registros filhos ao atual. A cada escrita de um registro ao novo arquivo é realizada a modificação do registro superior para guardar a posição do seu registro inferior, esquerdo ou direito, que será utilizada em outras transferências caso outro dado esteja relacionado com o registro inferior pela ordem da estrutura.

A busca por um registro no arquivo de simulação é binária, acessando através de uma ordem, crescente ou decrescente, eliminando metade do restante do arquivo ainda não explorado a cada iteração.

2.2.2. Resultados obtidos

ALEATÓRIO				
Número de Registros	**Transferências na criação**	**Tempo de criação**	**Transferências na pesquisa**	**Tempo de Pesquisa**
100	20.847,4	0,051s	6,7	0s
1.000	4.025.511,80	3,412s	17	0s
10.000				
100.000				
1.000.000				

2.3. Árvore B

2.3.1. Definição

É uma estrutura definida por ser n-ária, ou seja, possuir mais de dois descentes por página. Sua organização é dada por uma ordem M, a qual diz que em sua página raiz deve conter entre 1 e 2m itens e demais folhas no mínimo m itens e m+1 descendentes e no máximo 2m itens e 2m+1 descendentes. Além disso, é necessário manter as páginas folhas em um mesmo nível.

A inserção de itens sempre é realizada de modo a deixar a árvore balanceada, sendo o crescimento da árvore realizado horizontalmente, de baixo para cima, por este motivo qualquer forma de ordenação de itens é válida.

Para a inserção, é preciso caminhar pela árvore até encontrar o local correto, após encontrá-lo, verificar se a página possui menos de 2m itens. Se a página quebrar a regra da composição, são realizados alguns procedimentos: divisão da página, elevação do elemento central para a página pai e transferência dos maiores elementos para uma nova página.

A pesquisa é simplificada, utiliza-se o conceito da árvore binária de pesquisa: comparamos a chave com o elemento pai e caso não encontrado seguimos o caminho da esquerda ou direita comparando os valores até encontrar o elemento, ou chegar ao fim da árvore. Na pesquisa não há transferência entre as memórias, visto que os dados encontram-se na árvore em memória principal.

2.3.2. Resultados obtidos

ORDENADO CRESCENTE					
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Comparações na inserção**	**Transferências**
100	0s	0s	6,9	634	100
1.000	0,046s	0s	10,4	10.518	1.000
10.000	0,418s	0s	14	147.130	10.000
100.000	4,23s	0s	18	1.900.750	100.000
1.000.000					

ORDENADO DECRESCENTE					
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Comparações na inserção**	**Transferências**
100	0,003s	0,001s	6,3	634	100
1.000	0,039s	0s	10	5.278	1.000
10.000	0,375s	0,001s	14	73.448	10.000
100.000	3,63s	0s	18	940.962	100.000
1.000.000					

ALEATÓRIO					
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Comparações na inserção**	**Transferências**
100	0,006s	0s	5,5	580,8	100
1.000	0,032s	0,001s	13,2	9.209	1.000
10.000	0,318s	0s	16	123.965	10.000
100.000	0,1033s	5.3e-05	16	139.973	100.000
1.000.000					

2.4. Árvore B*

2.4.1. Definição

A árvore B* apresenta mecanismos de inserção, remoção e busca bem parecidos com os da árvore B, com a diferença de que a técnica de redistribuição das chaves também é empregada durante as operações de inserção, dessa maneira a divisão pode ser adiada até que duas folhas irmãs estejam completamente cheias, quando enfim o conteúdo dessas folhas é separado em três folhas (uma nova criada e as outras duas existentes anteriormente).

As chaves na árvore B*, sempre estão nas páginas externas, suas páginas internas apenas possuem índices que podem ou não já ter feito parte das chaves presentes na árvore, dessa forma é possível fazer uma busca sequencial nas páginas externas caso se deseje.

2.4.2. Resultados obtidos

ORDENADO CRESCENTE					
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Comparações na inserção**	**Transferências**
100	0,003s	0s	9,2	1.125	100
1.000	0,22s	0s	14,5	11.925	1.000
10.000	0,359s	0s	18	119.925	10.000
100.000	3,582s	0s	22,9	1.199.925	100.000
1.000.000					

ORDENADO DECRESCENTE					
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Comparações na inserção**	**Transferências**
100	0,006s	0s	8,2	195	100
1.000	0,011s	0s	12,5	2.130	1.000
10.000	0,287s	0s	15	21.405	10.000
100.000	3,03s	0s	20	214.260	100.000
1.000.000					

ALEATÓRIO					
Número de Registros	**Tempo total de criação**	**Tempo total de busca**	**Comparações na busca**	**Comparações na inserção**	**Transferências**
100	0s	0s	9,6	565,8	100
1.000	0,014s	0s	12,3	6.672,7	1.000
10.000	0,805s	0s	18	67.048,2	10.000
100.000					100.000
1.000.000					

3. Dificuldades encontradas

Uma grande dificuldade foi a construção dos arquivos binários de acordo com cada situação desejada. De início não sabíamos como inserir todos os dados gerados no arquivo. Primeiramente queríamos passar um vetor completo criado em memória principal para o arquivo, mas conforme o alto número de dados pensamos ser melhor criar um registro por vez e inserir no arquivo.

Outra dificuldade ao inserir os registros é que não tínhamos acesso diretamente aos dados do arquivo binário, dessa forma não era mostrado os dados criados estavam sendo inseridos corretamente. Para corrigir tal situação foi preciso uma nova funcionalidade que listava as informações contidas no arquivo, entretanto inicialmente a função não retornava todos os dados, até que vimos ser necessário sempre mover o ponteiro para o início do arquivo.

Na parte de acesso sequencial indexado tivemos que alterar a um detalhe ao pegar a posição da página no arquivo. Da forma como estava no exemplo do *slide*, verificamos se a posição era menor que a quantidade de elementos passados para o método, dessa forma o método não era bem-sucedido. Para contornar tal situação, foi criado uma variável que pega exatamente o número de páginas geradas a partir do número de itens e da quantidade de itens por página.

```
int cont = tam / TAMPAGINA;
```

```
while(posicao < cont && indices[posicao] <= chave){
            posicao++;
            comparacoesBusca++;
 }
```

Além disso, uma dúvida surgiu ao adicionar um novo método de pesquisa para substituir a sequencial. Pensamos em adicionar uma estrutura de árvore binária, entretanto não mudaria tanto o número transferências, dado que precisaríamos inserir item por item na árvore. Dessa forma, optamos por utilizar somente a busca binária utilizando o vetor de páginas gerado.

Na árvore B* uma das dificuldades enfrentadas foi a struct com uma parte interna e outra externa, afinal eu nunca tinha visto esse tipo de struct antes e trabalhar com ela não foi nada fácil, até que fosse possível entender como fazer o programa saber qual das structs iria acessar, essa foi sem dúvida a maior dificuldade encontrada.

4. Análise dos métodos

4.1. Tempo total de criação

O tempo total de criação corresponde ao tempo de criação da estrutura de pesquisa do método. Em relação a isso, podemos dizer que a árvore binária possui o pior tempo, tendo aproximadamente 3,412s para a quantidade de mil registros, enquanto os demais métodos não chegaram a 1s. Apesar de não ter sido possível

medir os resultados desse método a partir dos 10 mil registros, sabemos que o tempo aumentará exponencialmente.

Os métodos com melhores desempenhos de tempo foram a árvore B* e árvore B, sendo o melhor a árvore B*.

4.2. Tempo total de busca

Em relação ao tempo de busca de um registro, percebemos que o tempo foi infinitamente pequeno para os métodos de pesquisa estudados, aproximando do valor zero. Apesar disso, o método da árvore B, para arquivo ordenados decrescentemente e de forma aleatória, mostrou uma pequena variação, sendo essa de 0,001s.

4.3. Quantidade de comparações

Temos a quantidade de comparações relacionadas à busca de chaves dos seguintes métodos: acesso sequencial indexado, árvore B e árvore B*. Novamente, temos que o método de acesso sequencial foi o mais inferior em relação aos demais, uma vez que o número de comparações foi crescendo juntamente com o número de registros, enquanto os outros, mesmo com esse aumento de dados permaneceu quase constante esse valor de comparações.

Além disso, também foi computada as comparações realizadas na inserção de um registro. Entre os métodos de árvore B e árvore B*, o que realizou o menor número de comparação foi o B*.

4.4. Transferência

A transferência entre as memórias é extremamente custosa computacionalmente, além de contribuir para a possibilidade de danificar o arquivo em sua manipulação constante. Como esperado, o método de acesso sequencial se mostrou o mais eficiente, uma vez que utiliza o conceito de páginas, transferindo vários registros contidos em uma página ao invés de transferir item por item, como é feito nas árvores B e B*.

4.5. Métodos com arquivos ordenados crescentemente

Para arquivos ordenados de forma crescente, o melhor tempo de criação foi encontrado no acesso sequencial indexado, seguido da árvore B*, árvore B e o pior sendo árvore binária de acesso externo. Novamente, o tempo de busca manteve-se o valor zero.

Quanto ao número de comparações na busca, a árvore B foi o melhor método, seguido por B* e acesso sequencial mantendo um número crescente de comparações de acordo com o aumento do número de registros. E, novamente, árvore B* mostrou melhor desempenho em comparação para inserção de itens.

Em números de transferências, o pior resultado foi o da árvore binária, o qual possui um valor absoluto muito maior que o número de registros inseridos. Em contrapartida, o melhor foi o pesquisa sequencial indexada e árvore B e B* mantendo o custo igual ao número de itens manipulados.

4.6. Métodos com arquivos ordenados decentemente

Para os arquivos que estão ordenados de forma decrescente, no quesito “tempo total de criação”, quando o arquivo é de menor tamanho o melhor método é o acesso sequencial indexado seguido da árvore B, quando o tamanho do arquivo vai aumentando o método com menor tempo de criação é a árvore B*.

Em relação ao tempo de pesquisa, os métodos são quase equivalentes, com uma pequena diferença de tempo, em que a árvore B está em desvantagem, como os computadores não suportaram rodar os testes com arquivos tão grandes, tivemos poucas amostras de velocidade no momento da pesquisa.

Já falando sobre o número de comparações na busca a árvore B é o método que menos faz comparações, tanto em arquivos maiores quanto menores, ela vem seguida pela árvore B*, enquanto o acesso sequencial indexado faz o maior número de comparações na busca.

Sobre o número de comparações na inserção o acesso sequencial indexado não pode entrar nesse quesito, afinal ele pega os dados de arquivos fora da memória principal, comparando as árvores B e B*, a árvore B* tem um número de comparações muito menor do que sua “parente”, fazendo com que ela seja de longe

melhor para arquivos ordenados de forma decrescente que estejam em memória principal.

4.7. Métodos com arquivos desordenados

No caso dos arquivos desordenados, o método do acesso sequencial indexado não pode ser avaliado, pois um pré-processamento necessário para a sua execução é exatamente a ordenação dos dados.

Já quando analisada a árvore binária simulada, este é o único formato de arquivo possível, já que um conjunto de dados ordenado traria a pior organização de uma estrutura de árvore, em que os filhos são sempre alocados à direita, tornando a busca por ela sequencial. Porém, devido a alta complexidade de criação do arquivo que simula a árvore binária, o tempo para execução e o overhead deste método se torna inviável até mesmo para os arquivos pequenos. Por outro lado, a busca em um arquivo de árvore binária é extremamente veloz e requereu, nos experimentos um número, um número em torno de 1 a 2% da quantidade de registros.

Já entre os métodos da árvore B e B*, ambos apresentaram tempos de comparação e busca mínimos e número de comparação nos processos de busca e inserção equivalentes dentro do fato de que os dados foram organizados aleatoriamente.

5. Conclusão

A partir da implementação dos quatro algoritmos de pesquisa podemos afirmar que mostraram melhores desempenhos os métodos árvore B* e árvore B. O acesso sequencial indexado, apesar de não possuir o desempenho muito notório, mostrou destaque em relação aos demais quanto ao número de transferências entre as memórias externa e interna. Por fim, a árvore binária de pesquisa externa se mostrou bem menos eficiente em relação aos quesitos apresentados em detrimento dos demais tipos de pesquisa, mostrando que deve ser utilizado como último recurso caso seja necessário construir um arquivo que simularia a estrutura.

Porém, é necessário observar que o número de transferências no momento da busca dentro da árvore binária já montada é muito baixo e regular, que caso comparada aos métodos da árvore B e B*, quando não estiverem utilizando apenas

memória principal, pode se mostrar eficiente também, mas não tanto quanto estes dois últimos, que demonstraram aproveitar melhor as propriedades de sua organização e do sistema. Assim, é visível que os resultados obtidos podem se alterar em um trabalho futuro em que todas as estruturas usufruem de um mesmo sistema de memória virtual e paginação.