Trabalho Prático: Pesquisa Externa

Aluno em Graduação da Universidade Federal de Ouro Preto do curso Ciência da Computação:

Halliday Gauss Costa dos Santos.

Matrícula: 18.1.4093.

Área: Estrutura de Dados II.

Introdução:

Pesquisar por um dado é um problema comum na computação. Diversos algoritmos foram inventados com o intuito de aumentar a velocidade de pesquisa, dado que, a pesquisa sequencial muitas das vezes é lenta para um número grande de dados. Este documento apresenta a implementação de alguns desses algoritmos na linguagem **C++**. Os métodos de pesquisa implementados são: **Acesso Sequencial Indexado**, **Árvore Binária de Pesquisa em Memória Externa**, **Árvore B** e **Árvore B***.

O Acesso Sequencial Indexado e a Árvore Binária de Pesquisa em Memória Externa pesquisam um registro localizado em memória secundária. Os métodos restantes, por questão de complexidade de implementação, armazenam em sua estrutura, na memória primária, todos os registros. Como todos os métodos tem a mesma funcionalidade, o objetivo do trabalho será analisar quais são mais eficientes para determinadas instâncias.

Antes de fazer a pesquisa deve-se criar as condições para pesquisar (criação do método). A parte de criação do método tem que ser analisada separadamente da busca. Tanto a criação do método quanto a busca por um registro são baseadas em instâncias de 100, 1.000, 10.000, 100.000 e 1.000.000 de registros. Para cada instância, o método será analisado com os registros ordenados crescentemente, ordenados decrescentemente e desordenados. E para cada tipo de ordenação será contabilizado o número de transferências para a memória principal, o número de comparações entre chaves de pesquisa e o tempo de execução. Na busca, esses três últimos critérios de análise são definidos por uma média aritmética entre os dados coletados durante a busca de 10 chaves.

Observações:

- A função fopen(), utilizada em C para abrir um arquivo, foi apontada como insegura pela IDE Visual Studio Community, então a função fopen_s() foi usada no código fonte para abrir o arquivo.

- Houve um erro (bad_alloc) ao tentar utilizar a instância de 1 milhão de registros nos métodos que armazenam os registros em memória principal. Portanto, a instância de maior valor nesses métodos é de 750 mil.

Acesso Sequencial Indexado:

O Acesso Sequencial Indexado é um método de pesquisa baseado na divisão do arquivo em páginas e criação de uma tabela de índices em memória principal. Cada índice da tabela de índices contém o endereço de uma página do arquivo. A pesquisa começa na tabela de índices e caso encontre um intervalo onde possa estar o registro buscado, a página é transferida para a memória principal e começa uma busca binária na página para encontrar o registro. O registro não se localiza na estrutura caso não ache o intervalo na tabela de índices, ou não encontre o registro na página transferida.

É importante ressaltar que esse método de pesquisa só funciona se os registros de um arquivo estiverem ordenados crescentemente. O experimento foi feito com o número de itens por páginas igual a 4, ou seja, uma página do arquivo corresponde a 4 registros. Segue abaixo a tabela com os dados da criação da tabela de índices:

	CRIAÇÃO DO MÉTODO								
INSTÂNCIAS ⬇	ACESSO SEQUENCIAL INDEXADO								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	25	0	0.000100431 s	----	----	----	----	----	----
1000	250	0	0.00110209 s	----	----	----	----	----	----
10.000	2500	0	0.0165646 s	----	----	----	----	----	----
100.000	25000	0	0.1566083 s	----	----	----	----	----	----
1.000.000	250000	0	1.28493 s	----	----	----	----	----	----

A criação da tabela de índices é bem rápida e conta somente com transferência de dados da memória secundária para a primária. O número de transferência é sempre igual ao número de registros dividido pelo número de itens por página. Com 1 milhão de registros a tabela de índices foi criada em aproximadamente 1 segundo.

Feita a criação da tabela, foi analisado o desempenho da pesquisa:

INSTÂNCIAS	PESQUISA								
	ACESSO SEQUENCIAL INDEXADO								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	1	15	0.000009816	----	----	----	----	----	----
1000	1	131	0.000016613	----	----	----	----	----	----
10.000	1	1256	0.00028317	----	----	----	----	----	----
100.000	1	12506	0.000062297	----	----	----	----	----	----
1.000.000	1	125006	0.00030167	----	----	----	----	----	----

A pesquisa também demonstrou ser bem eficiente. Para buscar um registro em 1 milhão levou, em média, menos de 1 segundo. O número de comparações foi bem baixo e o de transferência foi de exatamente uma página.

Árvore Binária de Pesquisa em Memória Externa:

A Árvore Binária de Pesquisa em Memória Externa é um método baseado na Árvore Binária De Pesquisa em memória primária. A diferença está na estrutura dos registros. Um registro no arquivo contém sua chave, seus dados e dois inteiros que armazenarão a posição dos seus filhos dentro do arquivo. Caso um nó não tenha filho, o valor armazenado será -1. O filho da esquerda é menor que o nó pai e o filho da direita é maior que seu pai.

A criação da árvore se baseia na inserção de todos os registros. Para inserir um registro primeiro é conferido se a árvore está vazia e se estiver, o registro é inserido e seus filhos recebem o valor -1. Caso contrário, será feita uma comparação entre a chave do registro a ser inserido e a chave do registro da árvore (inicialmente a raiz). Se a chave do registro a ser inserido for maior, ele será direcionado para o

endereço do filho da direita dentro arquivo e, caso contrário, será levado a posição do arquivo onde se encontra o filho da esquerda. Esse processo é feito até que um filho com valor -1 seja encontrado. Em seguida, o registro é inserido no final do arquivo e sua posição é salva pelo seu pai na variável adequada.

Logo abaixo estão os dados da criação da árvore para as instâncias determinadas:

INSTÂNCIAS	CRIAÇÃO DO MÉTODO								
	ÁRVORE BINÁRIA EM MEMÓRIA EXTERNA								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	5150	15.049	0,0168591 s	5.150	15049	0,0291147 s	935	2.404	0,00687988 s
1000	501.500	1.500.499	2,1689 s	501.500	1500499	2,26018 s	13.543	36.628	0,0732151 s
10.000	50.015.000	150.004.999	3,2 m	50.015.000	150.004.999	3,25 m	175.937	487.810	1,28877 s
100.000	705.182.704	2.115.148.111	5,48 h	705.182.704	2.115.148.111	5,55 h	2.205.382	6.216.145	14,0055 s
1.000.000	9.872.557.856	29.612.073.554	22 dias	9.872.557.856	29.612.073.554	22 dias	51.282.842	144.547.107	7 dias

Percebe-se que a inserção de todos os registros na árvore é muito demorada quando os registros estão ordenados crescentemente ou decrescentemente. A inserção tem um custo $\log_2$**(n)** quando a árvore está balanceada, mas com não foi implementado algum método que a balanceie, o custo passa a ser **"n"**, ou seja, para inserir todos os registros o custo será **n²**. Para inserir 1 milhão de elementos foi estimado um tempo de 22 dias.

Quando os registros estão organizados de maneira aleatória, o tempo de execução, o número de comparações e o número de transferências são menores. Isso ocorre porque a ordem que os registros se encontram no arquivo causa um certo "balanceamento" ao inserir. Observação: a árvore não fica totalmente balanceada, sendo que a implementação de um método que a torne balanceada melhoraria o desempenho.

Em contrapartida pesquisa por um registro teve bom desempenho.

INSTÂNCIAS	PESQUISA								
	ÁRVORE BINÁRIA EM MEMÓRIA EXTERNA								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	50	149	0,00185457 s	51	152	0,00101828 s	9	26	0,00103904 s
1000	500	1.499	0,00326362 s	501	1.502	0,0023907 s	10	29	0,00111569 s
10.000	5.000	14.999	0,0236269 s	5.001	15.002	0,0198804 s	18	53	0,000718872 s
100.000	50.000	149.999	0,193823 s	50.001	150.002	0,194258 s	25	74	0,00175905 s
1.000.000	500.000	1.499.999	0,582892 s	500.001	1.500.002	0,5939032 s	30	90	0,02912847 s

A complexidade da pesquisa é $\log_2$**(n)** quando a árvore está balanceada e, caso contrário, se torna linear. Ou seja, a pesquisa também tem um melhor desempenho quando a árvore chega mais perto do balanceamento (registros não ordenados). Portanto, na pesquisa ocorre um alto número de comparações e transferências quando o arquivo de registros está ordenado, baixo número de comparações e transferências quando os registros estão armazenados de maneira aleatória.

Árvore B:

A Árvore B é uma estrutura de dados que tem um balanceamento automático, o qual é feito na inserção. Portanto, não ocorre o problema de balanceamento supracitado.

Geralmente, as implementações do método de inserção e pesquisa em uma Árvore B são feitas utilizando recursividade. Porém, o código fonte mostrado não faz o uso dessa ferramenta, o que pode resultar em uma melhoria no uso da memória, pois não terá chamadas recursivas armazenadas em memória esperando a execução de outras chamadas. No entanto, terá um ponteiro a mais na estrutura de registro, que é o ponteiro que aponta para a página pai. Sem as chamadas recursivas utiliza-se esse ponteiro para subir novamente na árvore.

Como a Árvore B nessa implementação é criada em memória primária, não foi possível fazer a análise dos dados com a instância de 1 milhão. Ocorreu um erro de alocação diante do uso excessivo de memória. No entanto, foi possível fazer a análise com instância igual a 750 mil registros.

Os testes foram feitos com uma Árvore B de ordem 2:

INSTÂNCIAS	CRIAÇÃO DO MÉTODO								
	ÁRVORE B								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	100	2.591	0,000368497	100	1.157	0,000373783 s	100	1.903	0,000320925 s
1000	1.000	40.802	0,00393756 s	1.000	17.813	0,00384128 s	1.000	30.688	0,00367101 s
10.000	10.000	554.503	0,0337009 s	10.000	240.313	0,0348725 s	10.000	385.577	0,0372575 s
100.000	100.000	7.024.261	0,358298 s	100.000	3.022.847	0,380417 s	100.000	4.874.447	0,327213 s
750.000	750.000	62.235.585	2,51221 s	750.000	26.905.669	2,53459 s	750.000	47.664.717	2,5397 s

Para inserir todos os registros na árvore, independente da ordenação do arquivo de registros, foi gasto 2,5 segundos na maior instância e nas instâncias restantes o tempo de execução não chegou a 1 segundo. O número de comparações não é tão alto visto que a estrutura já faz o balanceamento automaticamente.

A pesquisa mostrou-se extremamente eficiente independentemente da ordenação do arquivo.

INSTÂNCIAS	PESQUISA								
	ÁRVORE B								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	0	20	0,00139508 s	0	13	0,00120592 s	0	22	0,000696219 s
1000	0	28	0,00462132 s	0	25	0,000832895 s	0	32	0,000861967 s
10.000	0	38	0,000864233 s	0	31	0,00089406 s	0	32	0,000068338 s
100.000	0	50	0,000681117 s	0	41	0,00110889 s	0	36	0,00006796 s
750.000	0	50	0,00135355 s	0	36	0,00141911 s	0	57	0,000161217 s

Percebe-se que para pesquisar um registro, em média, o número de comparações é menor igual a 50 em todas as instâncias.

Árvore B*:

A Árvore **B*** é uma das variações da Árvore B. A organização de ambas as estruturas é bem parecida. A diferença é que a Árvore **B*** é formada por índices e registros, todos os registros se encontram em alguma página folha e os índices servem somente para guiar até um determinado registro buscado. A semelhança

está na estrutura dos índices que é basicamente uma Árvore **B**. Portanto, o balanceamento de uma Árvore **B*** também é automático

O código fonte também não utiliza recursividade no método de inserção e pesquisa em uma Árvore **B***, o que também resulta numa melhoria no uso da memória. No entanto, também terá um ponteiro a mais na estrutura de registro, que é o ponteiro que aponta para a página pai.

Como todos os registros ficam em uma página folha, a árvore terá mais elementos do que a quantidade de registros inseridos, ou seja, terá um consumo maior de memória e, por esse motivo que o valor da instância para arquivos ordenados crescentemente será menor que o valor da maior instância da Árvore **B**. O valor da maior instância da Árvore **B*** para arquivos ordenados crescentemente é de 650 mil registros.

Os testes foram realizados levando em consideração que a ordem da Árvore é 2 e a quantidade de itens numa página folha é 4:

	CRIAÇÃO DO MÉTODO								
INSTÂNCIAS ⬇	ÁRVORE B*								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	100	3.087	0,000452316 s	100	997	0,000405498 s	100	2.098	0,00386242 s
1000	1.000	44.372	0,00290003 s	1.000	14.199	0,00267765 s	1.000	30.074	0,00288229 s
10.000	10.000	568.964	0,0349072 s	10.000	183.529	0,0331225 s	10.000	381.468	0,0341747 s
100.000	100.000	6.939.847	0,386313 s	100.000	2.248.549	0,366253 s	100.000	4.556.932	0,318033 s
750.000	650.000	51.855.598	2,14096 s	750.000	19.687.094	2,02766 s	750.000	40.804.715	2,504 s

Para inserir todos os registros na árvore, independente da ordenação do arquivo de registros, foi gasto 2,1 segundos na maior instância e o restante não chegou a 1 segundo. O número de comparações não é tão alto, visto que a estrutura já faz o balanceamento automaticamente.

A pesquisa também se mostrou extremamente eficiente independentemente da ordenação do arquivo.

INSTÂNCIAS	PESQUISA								
	ÁRVORE B*								
	ORDENAÇÃO DOS REGISTROS:								
	CRESCENTE			DECRESCENTE			ALEATÓRIO		
	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO	Nº DE TRANSF.	Nº DE COMPAR.	TEMPO DE EXECUÇÃO
100	0	18	0,00109114 s	0	17	0,00199238 s	0	19	0,00236125 s
1000	0	20	0,00481086 s	0	27	0,000575022 s	0	23	0,000057766 s
10.000	0	30	0,00106774 s	0	31	0,000738505 s	0	19	0,00726423 s
100.000	0	38	0,000634299 s	0	39	0,00108057 s	0	25	0,000464398 s
750.000	0	30	0,00137243	0	52	0,00110889 s	0	45	0,000102319 s

Percebe-se que o número de comparações é muito baixo e no máximo 52. Logo, o desempenho da pesquisa é bem elevado. A pesquisa na Árvore **B*** se mostrou melhor que a pesquisa na Árvore **B**, independente da ordenação do arquivo em instâncias de 100 até 100 mil registros.

Conclusão:

Conclui-se a Árvore **B*** foi a estrutura que apresentou o melhor desempenho, tanto em inserção de todos os registros (criação do método de pesquisa), quanto em pesquisa, mas não ficou tão longe da performance da Árvore **B**.

Dos métodos implementados em memória externa, o Acesso Sequencial Indexado obteve o melhor desempenho para registros ordenados. A Árvore Binária de Pesquisa foi o método com a pior performance, mas também não ficou tão atrás da Árvore **B** e Árvore **B*** na pesquisa em registros aleatórios.

A dificuldade de implementar por conta própria a inserção numa Árvore B e Árvore B* levou a uma implementação sem recursividade, sendo que o restante dos métodos foram implementados sem muitas dificuldades. No entanto, algumas melhorias poderiam ser feitas: como a criação de um arquivo.cpp e um arquivo.h para armazenar uma estrutura de registro genérica, o que resultaria em uma simplificação de todo o código.

A realização desse trabalho foi de suma importância para o melhor entendimento das estruturas de dados de pesquisa mais utilizadas na computação. Esses algoritmos mostram uma eficiência altíssima em inserir, remover e pesquisar registros.