Universidade Federal de Ouro Preto - Campus Ouro Preto

Clara Luiza Alemar Ribas - 20.1.4008
Lívia Stéffanny de Sousa - 20.1.4029
Karine Isabelle Marins - 20.1.4057

BCC203 - Estrutura de Dados II

TRABALHO PRÁTICO I - PESQUISA EXTERNA

Implementação de métodos de pesquisa externa

Ouro Preto
2021

Clara Luiza Alemar Ribas - 20.1.4008
Lívia Stéffanny de Sousa - 20.1.4029
Karine Isabelle Marins - 20.1.4057

TRABALHO PRÁTICO I - PESQUISA EXTERNA

Trabalho apresentado à disciplina
Estrutura de Dados II, do curso de
Ciência da Computação,
referente ao período 21.1.

.

Professor Orientador: Guilherme Tavares de Assis

Ouro Preto
2021

Sumário

1.	**Introdução**	**4**
2.	**Objetivos**	**5**
3.	**Desenvolvimento**	**6**
	3.1. Acesso Sequencial Indexado	
	3.2. Árvore Binária de Pesquisa Externa	
	3.3. Árvore B	
	3.4. Árvore B*	
4.	**Comparação entre os testes**	**16**
5.	**Considerações Finais**	**18**
6.	**Código**	**18**
7.	**Referências bibliográficas**	**19**

1. Introdução

Este relatório referente ao primeiro trabalho prático de Estruturas de Dados II foi implementado com o intuito de realizar uma análise sobre duas importantes memórias de um computador: a memória principal e a memória secundária. Temos que a pesquisa em memória secundária envolve arquivos contendo um número de registros maior do que a memória interna pode armazenar, uma vez que a memória principal não consegue armazenar uma grande quantidade de dados, e visto que essa possui baixa velocidade e alta capacidade de armazenar um grande volume de dados. Levando em conta que se fazem necessários novos métodos para manter o equilíbrio entre a demanda do usuário e o tempo de resposta da execução do programa, alguns aspectos têm que ser levados em conta para a análise: o custo para acessar um registro é maior que em comparação à memória interna, logo, a medida de complexidade está relacionada com o custo para transferir dados entre a memória principal e a memória secundária. Logo, existem vários métodos que nos permitem fazer tais tarefas. Dessa forma, o problema proposto para a criação do trabalho prático consiste justamente em criar um programa em linguagem C para a realização de uma pesquisa em memória secundária mediante um valor de chave informado pelo usuário. O programa tem a possibilidade de fazer a pesquisa com diferentes tamanhos (quantidades) de registro do arquivo para diferentes situações de ordem das chaves (crescente, decrescente e aleatório) e com 4 diferentes métodos, sendo eles: (1) acesso sequencial indexado, (2) árvore binária de pesquisa adequada à memória externa, (3) árvore B e (4) árvore B* buscando encontrar de forma mais eficiente o registro buscado pelo usuário.

2. Objetivos

Neste trabalho tem-se como objetivo um estudo da complexidade de desempenho dos seguintes métodos de pesquisa externa: acesso sequencial indexado, árvore binária de pesquisa adequada à memória externa, árvore B e árvore B*. Bem como a implementação dos métodos citados, a fim de melhor compreender seu funcionamento e fazer análises em relação ao número de comparações, tempo de execução e número de transferências.

3. Desenvolvimento

A construção do trabalho se deu por meio da linguagem C e para cada método, há a existência de um arquivo.c acrescido de um arquivo.h responsáveis por realizarem as principais manipulações nos arquivos a serem tratados. Antes da pesquisa, é necessária a criação do arquivo binário contendo os registros com a quantidade e a situação de ordem especificada pelo usuário. O presente programa cria arquivos com os campos **long int, char[1000] e char[5000],** além da **chave** necessária para a pesquisa. A seguir, será explicado o funcionamento de cada método de pesquisa seguido de sua respectiva análise experimental da complexidade de desempenho obtida através da média da pesquisa de 10 chaves em tais métodos, onde a análise se deu para cada um dos quesitos em relação a execução da pesquisa.

Para os testes, pesquisou-se pelas seguintes chaves em todos os métodos de pesquisa:

- ➔ **Para arquivo com 100 registros:** chaves *14, 6, 85, 75, 44, 25, 33, 74, 5, 36*
- ➔ **Para arquivo com 1000 registros:** chaves 392, 579, 291, 848, 115, 12, 999, 736, 79, 698
- ➔ **Para arquivo com 10.000 registros:** chaves *2008, 2973, 9029, 7851, 7298, 1137, 4491, 2171, 3000, 2556.*
- ➔ **Para arquivo com 100.000 registros:** chaves *77427, 88697, 89077, 38151, 35143, 64745, 61757, 10785, 92972, 11396.*
- ➔ **Para arquivo com 1.000.000 registros:** chaves *833723, 372019, 610062, 505623, 489867, 150427, 850382, 660273, 785295, 151410.*

Obs.: Nas tabelas tem-se as análises para a execução dos índices, **representadas por PI,** e para execução da pesquisa em si, **representado por PP.**

3.1 Acesso sequencial indexado

O método em questão utiliza o princípio da pesquisa sequencial: todos os registros são lidos sequencialmente até encontrar uma chave maior ou igual à chave de pesquisa. Porém, ele necessita de uma entrada específica para ser executado, o arquivo deve estar obrigatoriamente ordenado pelas chaves do registro, portanto quando o mesmo se apresentar decrescente ou desordenado, este método não se aplica. É importante ressaltar que é necessária a criação de uma tabela de índices das páginas, contendo a chave do registro e o endereço no arquivo onde o primeiro registro contém essa chave, sendo essa responsável por diminuir a quantidade de transferências entre memória principal e memória secundária.

3.1.1 Análise

Durante a implementação deste método, tem-se que é necessário que o arquivo a ser tratado esteja previamente ordenado e, portanto, é vedado o uso de arquivos desordenados,

permitindo assim que seja executado somente com arquivos de ordens crescentes e decrescentes. Além disso, a utilização da tabela de índices das páginas, como já dito, diminui a quantidade de transferências entre memória principal e memória secundária. Nos testes foi-se constatado que tanto na ordenação ascendente quando na ordenação descendente as transferências por índice cresceram simultaneamente com a quantidade de registros, assim como as comparações pela execução da pesquisa e as transferências pela execução da pesquisa permaneceu o mesmo em todos os testes, tal como as comparações por índice.

3.1.2 Pesquisa

Na execução do método, como já narrado anteriormente, ocorre a criação da tabela de páginas por meio de uma estrutura de dados. Esta estrutura será responsável por armazenar o primeiro elemento de cada página. O uso da tabela facilita a pesquisa e reduz significativamente a quantidade de transferências de dados da memória principal para a memória secundária, isso pelo fato de que antes de varrer um determinado registro no arquivo, há a comparação do dado solicitado com os elementos presentes em cada posição da estrutura. Por meio da comparação citada anteriormente, é possível garantir que haja o acesso ao arquivo caso exista a possibilidade deste conter aquele dado entre os elementos ali presentes.

3.1.3 Testes e resultados obtidos

A seguir apresenta-se tabelas com os resultados encontrados nas análises para arquivos ordenados crescente, descendente e desordenados aleatoriamente, bem como prints das execuções.

ACESSO SEQUENCIAL INDEXADO					
Ordenado Crescente					
N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	4	39	385	3847	-
Comparações (PI)	0	0	0	0	-
Tempo (ms) (PI)	0ms	7.81	31.25	620.18ms	-
Transferências (PP)	1	1	1	1	-
Comparações (PP)	13.7	31.4	175.4	2206.5	-
Tempo (ms) (PP)	0ms	0ms	0ms	0ms	-

ACESSO SEQUENCIAL INDEXADO					
Ordenado Decrescente					
N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	4	39	385	3847	-
Comparações (PI)	0	0	0	0	-
Tempo (ms) (PI)	0ms	6.18	46.8	584.1	-
Transferências (PP)	1	1	1	1	-
Comparações (PP)	15.8	30.6	236.6	1668	-
Tempo (ms) (PP)	0ms	0ms	0ms	0ms	-

3.1.4 Dificuldades

Em relação às dificuldades enfrentadas para implementação do método de acesso sequencial indexado não tivemos grandes problemas, uma vez que o método foi bem compreendido pelas alunas durante a explicação do conteúdo feita pelo professor apesar dos testes feitos com registros de valores de 1.000.000 excederem demasiadamente seu funcionamento.

3.2 Árvore Binária de Pesquisa Externa

A árvore binária de pesquisa é muito utilizada quando a memória principal armazena todos os registros. Em memória secundária, as árvores binárias podem ser utilizadas onde os nós da árvore são armazenados no disco, e os apontadores à esquerda e à direita de cada nó se tornam endereços de disco em vez de endereços de memória. Neste método de pesquisa, inicialmente construímos uma árvore binária em memória externa, isto é, em um arquivo binário, para por fim realizamos a pesquisa. Para que isso ocorra, os itens são lidos, e durante o processo de criação e inserção do arquivo binário que representa a árvore, inserimos também, as linhas referentes aos filhos desse nó da árvore. É importante citar que, cada nó da árvore é representado por uma informação na qual chamamos de chave, que é usada para realizar as comparações. Assim, por definição, seguimos a padronização de regras, na qual a chave menor do que a qual estamos comparando, é inserida no filho da esquerda e a maior no filho da direita. Porém é muito custoso fazer isso em arquivo, a utilização de muitos deslocamentos em arquivos, tendo uma complexidade média n*log n onde n corresponde ao tamanho de registros e log n é a altura da árvore. Para pesquisar alguma chave de registro na árvore, pode ser considerado como uma

operação trabalhosa para a máquina, pois a todo momento é necessário ir ao arquivo fazer a leitura dos registros analisando qual o caminho a se percorrer até encontrar ou não.

3.2.1 Análise

Para este método é criado um novo arquivo que representa uma árvore binária em um arquivo de memória externa. Para que isso ocorra, durante a criação e inserção no arquivo binário, os itens são lidos e também são inseridas as linhas que representam os filhos do determinado item, que representa o nó da árvore. Cada nó contém uma chave, a qual foi utilizada para realizar as comparações. Por definição, nós com menor chave são inseridos à esquerda, enquanto nós com chave maior são inseridos à direita. Por este motivo, como o gerador permite a geração de arquivos ordenados de três formas diferentes, é possível notar árvores preenchidas de 3 formas distintas: Quando o arquivo é composto por chaves ordenadas de forma crescente, seguindo o padrão de preenchimento da árvore, todos os nós estarão inseridos à direita, enquanto com o arquivo contendo as chaves ordenadas de forma decrescente o preenchimento é inverso ao anterior. Já o arquivo original ordenado de forma aleatória permite preenchimento diferente, em que não há um padrão a ser observado.

3.2.2 Pesquisa

A busca neste método simula a pesquisa em uma árvore binária comum, ou seja, percorre-se a árvore binária gerada realizando comparações entre os nós para indicar a direção a ser realizada a pesquisa. Com a particularidade dos diferentes tipos de arquivos, observa-se que quando o mesmo é preenchido com chaves aleatórias, a pesquisa é mais eficiente que os outros dois tipos de ordenação, sendo necessárias menos comparações, transferências e também menor tempo para execução.

3.2.3 Testes e Resultados Obtidos

Segue abaixo, tabelas com os resultados dos valores que foram suportados pela máquina, bem como prints das execuções.

Árvore Binária Externa					
Ordenado Crescente					
N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	300	3000	30000	300000	-
Comparações (PI)	198	1998	19998	199998	-
Tempo (ms) (PI)	18,75ms	123,4375ms	1.456,25ms	22.131,25ms	-

Transferências (PP)	36,7	471,9	4248,1	500438,4	-
Comparações (PP)	75,4	806,7	8498,8	114026	-
Tempo (ms) (PP)	0ms	10,9375ms	42,1875ms	815,625ms	-

Árvore Binária Externa
Ordenado Decrescente

N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	300	3000	30000	300000	-
Comparações (PI)	99	999	9999	99999	-
Tempo (ms) (PI)	18,75ms	151,5625ms	1.668,75ms	19.066,5ms	-
Transferências (PP)	61,3	526,1	5.749,6	42986	-
Comparações (PP)	552,9	5006,5	5751,4	42988	-
Tempo (ms) (PP)	0ms	9,375ms	56,25ms	514,0625ms	-

Árvore Binária Externa
Desordenado Aleatoriamente

N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	796,44	14.877,8	337.909,2	-	-
Comparações (PI)	1.045,6	24.545,5	598.127,4	-	-
Tempo (ms) (PI)	29,5139ms	493,75ms	14.031,25ms	-	-
Transferências (PP)	6,3	7,9	12,9	-	-
Comparações (PP)	16	16,4	22,1	-	-
Tempo (ms) (PP)	0ms	0ms	1,5625ms	-	-

3.2.4 Dificuldades

Como este método de pesquisa binária já havia sido introduzido e trabalhado no período anterior, em Estrutura de Dados I, já havia certa familiaridade com a implementação do mesmo. A grande dificuldade surge no momento de adaptar o algoritmo para a manipulação de arquivos binários, o que demonstrou-se complicado mas pode ser realizado com ajuda do conteúdo disponibilizado em aula pelo professor.

3.3 Árvore B

A árvore B surgiu para solucionar o problema enfrentado pelas árvores binárias, sendo uma proposta para manter o crescimento das árvores equilibrado e permitir inserções e retiradas. Uma característica imprescindível dessa árvore é o fato da mesma ser uma árvore n-ária, isto é, ela pode possuir mais de dois descendentes por cada nó da árvore. Em sua página raiz ela pode conter entre 1 e 2M itens, suas demais páginas contém no mínimo M itens e M+1 descendentes. As páginas folha, ou seja, todas que não são a página raiz aparecem no mesmo nível. Uma árvore B sempre vai estar balanceada e um dos modos que garante esse balanceamento é o de que ela cresce pra cima.

3.3.1 Análise

Na implementação de uma árvore B, diferente do Acesso sequencial indexado, qualquer arquivo que contenha as características pode ser utilizado, independentemente de estar ou não ordenado, já que como a árvore cresce para cima, se codificada corretamente vai transformar o arquivo recebido em uma árvore B.

3.3.2 Pesquisa

A operação de pesquisa em uma árvore B é semelhante à pesquisa em uma árvore binária de pesquisa. Desde que a chave pesquisada esteja presente dentro do arquivo ela será encontrada dentro da árvore B, comparando a chave com as chaves que estão na página raiz até encontrar a chave desejada ou o intervalo no qual ela se encaixa. A maior parte do tempo gasto em todo o método da árvore B se dá na inserção do arquivo, a pesquisa é feita de forma mais rápida, dado que nossos apontadores guiam o programa até que ele consiga encontrar (ou não, caso não exista) a chave que buscamos.

3.3.3 Testes e resultados obtidos

A seguir apresenta-se tabelas que com os resultados, bem como prints das execuções e das falhas encontradas.

ÁRVORE B
Ordenado Crescente

N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	100	1000	-	-	-
Comparações (PI)	538	9524	-	-	-
Tempo (ms) (PI)	2,468ms	32,48ms	-	-	-
Transferências (PP)	0	0	-	-	-
Comparações (PP)	5,7	9,2	-	-	-
Tempo (ms) (PP)	0,0041ms	0,0021ms	-	-	-

ÁRVORE B
Ordenado Decrescente

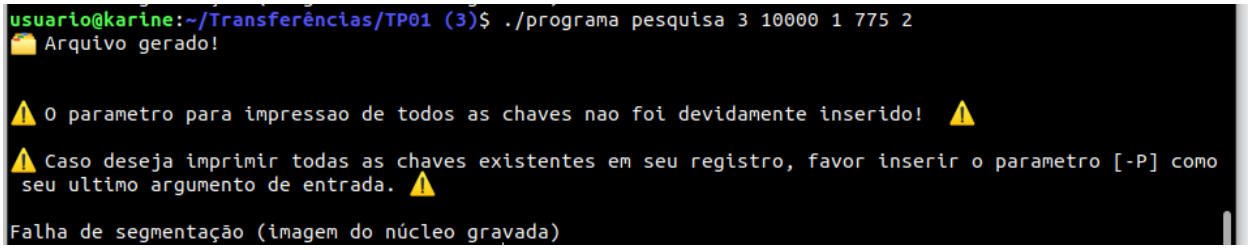
N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	100	1000	-	-	-
Comparações (PI)	324	5278	-	-	-
Tempo (ms) (PI)	2,26ms	35,12ms	-	-	-
Transferências (PP)	0	0	-	-	-
Comparações (PP)	6,1	8	-	-	-
Tempo (ms) (PP)	0,0035ms	0,0092ms	-	-	-

ÁRVORE B
Desordenado Aleatório

N° de registros	100	1000	10.000	100.000	1.000.000

Transferências (PI)	100	1000	-	-	-
Comparações (PI)	475,8	7921,9	-	-	-
Tempo (ms) (PI)	1,94ms	20,74ms	-	-	-
Transferências (PP)	0	0	-	-	-
Comparações (PP)	5,2	8,2	-	-	-
Tempo (ms) (PP)	0,0057ms	0,019ms	-	-	-

Para os casos com 10.000 ou mais registros o programa apresentou instabilidade e não foi possível fazer as análises. A seguir, segue o problema apresentado para tais casos.



3.3.4 Dificuldades

Aqui, a dificuldade está em executar o programa para arquivos com mais de 10.000 registros. Tentou-se de todas as formas contornar tal problema mas não obteve sucesso e as análises só foram possíveis de serem feitas com até 1000 registros. No mais a implementação do código foi bem tranquila.

3.4 Árvore B*

A árvore B* é muito semelhante à árvore B, no entanto, todos os registros estão armazenados no último nível (páginas folhas ou externas) e os níveis acima são constituídos por índices (páginas internas) contendo apenas as chaves dos registros cuja disposição é a mesma da árvore B. O algoritmo, como dito, é bem semelhante à árvore B, possuindo apenas de diferente o registro seletivo para o tipo página, que permite ao programador especificar durante a implementação se a página é externa ou interna, a fim de utilizar do conteúdo de tais os quais fazem parte de um tipo união.

3.4.1 Análise

Como já dito, a implementação da árvore B* assemelha-se muito a de uma árvore B, contudo deve-se atentar a pontos importantes nesse processo. Na construção da árvore devemos

identificar as páginas internas e as externas (páginas folha), além do processo de pesquisa, onde ao se encontrar a chave no índice da árvore ela guiará a pesquisa até que seja encontrado em uma página folha. Assim o valor encontrado na página pai servirá de índice para chegar ao desejado. Na implementação do método, qualquer tipo de ordenação pode ser utilizado, pois da mesma forma que a árvore B, independente da ordenação do arquivo os elementos serão inseridos na árvore e organizados e reajustados quando for necessário.

3.4.2 Pesquisa

Para realizar uma pesquisa na árvore B* é preciso levar em conta que a pesquisa sempre termina em uma página folha, já que são nelas que poderemos encontrar o registro buscado. O restante da árvore se comporta como uma ponte, dado que mesmo se a chave desejada estiver em uma página pai, ela servirá como índice. Ao encontrar a chave desejada em uma página do índice, a pesquisa não termina, a pesquisa continua até que se encontre uma página folha.

3.4.3 Testes e Resultados Obtidos

Segue prints de algumas execuções para exemplificar, bem como tabelas com os resultados obtidos nos experimentos.

ÁRVORE B*
Ordenado Crescente

N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	100	1000	-	-	-
Comparações (PI)	725	11507	-	-	-
Tempo (ms) (PI)	6,086ms	35,94ms	-	-	-
Transferências (PP)	0	0	-	-	-
Comparações (PP)	10,8	16,5	-	-	-
Tempo (ms) (PP)	0,009ms	0ms	-	-	-

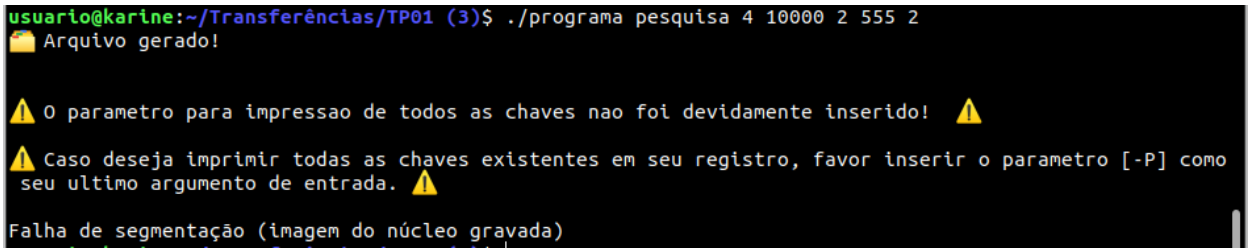
ÁRVORE B*
Ordenado Decrescente

N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	100	1000	-	-	-
Comparações (PI)	321	5273	-	-	-
Tempo (ms) (PI)	4,5ms	33,0125ms	-	-	-
Transferências (PP)	0	0	-	-	-
Comparações (PP)	11,2	16,1	-	-	-
Tempo (ms) (PP)	0,006ms	0ms	-	-	-

ÁRVORE B*
Desordenado aleatoriamente

N° de registros	100	1000	10.000	100.000	1.000.000
Transferências (PI)	100	1000	-	-	-
Comparações (PI)	663,7	8.833,3	-	-	-
Tempo (ms) (PI)	4,2ms	26,5625ms	-	-	-
Transferências (PP)	0	0	-	-	-
Comparações (PP)	9,5	14,6	-	-	-
Tempo (ms) (PP)	0,006ms	0ms	-	-	-

Novamente, para os casos com 10000 ou mais registros o programa apresentou instabilidade e não foi possível fazer as análises. Apresenta-se o problema:



3.4.4 Dificuldades

A principal dificuldade se deu no entendimento e funcionamento do tipo seletivo, que possui união (UU) e afins, já que não havíamos trabalhado com ele anteriormente. Ressalta-se também a dificuldade para tratar as páginas externas e internas separadamente na inserção, uma vez que anteriormente estávamos considerando que a inserção só seria feita na página externa e não tínhamos a condicional da página interna, assim o código não respondia ao que desejávamos. Ainda, durante a implementação houve dificuldade na contagem de comparações, uma vez que inicialmente a contagem estava sendo feita de forma errada e os resultados eram muito grandes em relação ao esperado. Entretanto, após estudar, errar inúmeras vezes, ver e rever a aula disponibilizada pelo professor, foi possível a implementação sem maiores problemas.

4. Análise comparativa de desempenho entre os métodos

4.1 Ordenação de forma crescente

Ressaltando-se os resultados para a execução da pesquisa, comparando-se o tempo de teste, o Acesso Sequencial Indexado, Árvore B e a Árvore B* apresentam melhor performance frente aos casos definidos. Ainda, percebe-se que à medida que a quantidade de dados aumenta, o tempo de execução da Árvore Externa aumenta exponencialmente. Em contrapartida, à medida que a quantidade de dados cresce, o Acesso Sequencial Indexado apresenta maior número de comparações. Falando em relação a quantidade de comparações, a Árvore Binária apresenta resultados superiores aos outros métodos, por realizar grandes números de comparações. O Acesso Sequencial Indexado, inicialmente teve uma quantidade satisfatória de comparações, mas à medida que os dados foram aumentando, as comparações tiveram um grande salto em seu valor. Em relação à quantidade de transferências neste tipo de execução, às Árvores B e B* possuem melhor desempenho.

Tratando da análise para a criação de índices, no quesito quantidade de transferências, o Acesso Sequencial apresenta resultado mais satisfatório, seguido pela Árvore de Pesquisa Externa e Finalmente as Árvores B e B*. Ainda nesta análise, as Árvores B e B* possuem a quantidade de transferências relacionada à quantidade de registros existentes.

4.2 Ordenação de forma decrescente

Nesta forma de ordenação, tratando da execução da pesquisa, as Árvores B e B* também não apresentam grande vantagem em relação ao tempo em relação ao Acesso Sequencial Indexado. Entretanto é notável a diferença na eficiência em relação à Árvore Binária de Pesquisa Externa, a qual apresenta aumento exponencial de tempo com o aumento da quantidade de dados, diferentemente da B e B*. No quesito quantidade de comparações, também é notável a distinção das Árvores B e B*. Quando comparados entre si, a diferença dos resultados dos dois métodos é pequena, mas ao comparar com os outros dois algoritmos percebe-se grande

eficiência. A quantidade de transferências para arquivos decrescentes se mantém igual à de arquivos crescentes.

Considerando a criação de índice, as execuções acontecem de forma semelhante, logo as observações anteriormente feitas são válidas.

4.3 Ordenação de forma aleatória

Levando em consideração que no método de Acesso Sequencial não se pode utilizar arquivos ordenados aleatoriamente, comparando-se o desempenho das Árvores B e B* com a Binária, é possível notar que as primeiras continuam apresentando melhor performance, ainda que mais sútil neste caso. Apesar de apresentarem resultado satisfatório, a Árvore de Pesquisa Externa ainda realiza mais comparações do que a B e B*. Ainda, observa-se que, novamente, o número de transferências para criação de índices é igual ao número de registros nas Árvores B e B*, e nesta mesma execução, as comparações da Árvore Binária Externa crescem exponencialmente.

5. Considerações Finais

Em detrimento dos fatos aqui apresentados, pode-se concluir os objetivos propostos por este trabalho, os quais dizem respeito a execução dos 4 métodos estudados: (1) acesso sequencial indexado, (2) árvore binária de pesquisa adequada à memória externa, (3) árvore B e (4) árvore B*. Ainda, ressalta-se que o trabalho foi de grande valia, uma vez que pode-se ver na prática que a teoria apresentada condiz com a realidade.

Assim, confirmou-se que dos métodos trabalhos os que obtiveram melhores performances nos casos definidos foram os métodos árvore B e B*. Já em relação aos demais métodos obtivesse execução considerável e resultados condizentes com o que se esperava da teoria estudada.

6. Código

Para ter acesso ao código em .zip basta acessar, com o email institucional o link a seguir:

https://drive.google.com/drive/folders/1l6sjRjMRNLuGMEZuftkQjuMdeqUCz75R?usp=sharing

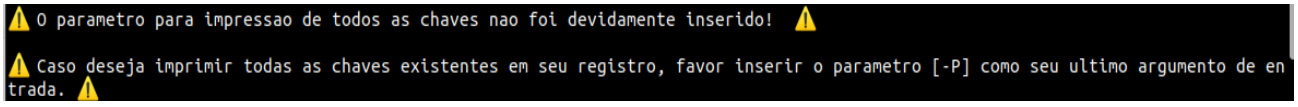
Ressalta-se que a execução do código se dá da seguinte maneira:

pesquisa <método> <quantidade> <situação> <chave> [-P]

onde:

- <método> representa o método de pesquisa externa a ser executado, podendo ser um número inteiro de 1 a 4, de acordo com a ordem dos métodos mencionados;
- <quantidade> representa a quantidade de registros do arquivo considerado;
- <situação> representa a situação de ordem do arquivo, podendo ser 1 (arquivo ordenado ascendentemente), 2 (arquivo ordenado descendentemente) ou 3 (arquivo desordenado aleatoriamente);
- <chave> representa a chave a ser pesquisada no arquivo considerado;
- [-P] representa um argumento opcional que deve ser colocado quando se deseja que as chaves de pesquisa dos registros do arquivo considerado sejam apresentadas na tela.

O último parâmetro deve ser necessariamente inserido, caso seja [-P] as chaves do arquivo gerado serão impressas, caso seja qualquer outro valor (ex: 1,2,3), uma mensagem de aviso é apresentada, como exemplificado abaixo:



7. Referências bibliográficas

★ Tavares, Guilherme. Pesquisa Externa. Disponível em: <http://www.decom.ufop.br/guilherme/BCC203/externa.pdf/>. Acesso em 04 de novembro de 2021.