UFOP – UNIVERSIDADE FEDERAL DE OURO PRETO

DECOM – DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃ

BCC203 – ESTRUTURA DE DADOS II

Trabalho Prático 1:
Pesquisa Externa

por:

IAN MARTINS ARANTES 21.2.4178

PEDRO AUGUSTO VIEIRA CAMPOS LEITE - 20.2.4008

LUCAS GOMES DOS SANTOS - 20.1.4108

Ouro Preto – MG
Junho de 2023

1) Introdução:

Conforme a massa de dados de um determinado procedimento torna-se maior, surge a necessidade de fazer uso da memória secundária, tendo em vista que a memória principal não suporta tal volume de dados. Dessa maneira, a busca por determinado itens dentre esses dados passa a necessitar de um método de pesquisa externa, o qual acessa a memória externa que contém tais informações. Assim sendo, tem-se um acesso mais rápido e eficiente ao fazer uso desses registros.

Assim, o Acesso Sequencial Indexado, a Árvore Binária de Pesquisa, Árvore B e Árvore B* são técnicas que realizam esse acesso e busca nem arquivos externos. Cada uma delas usa uma estratégia para ler e pesquisar dados, visando realizar o procedimento de pesquisa.

Acerca dos objetivos desse trabalho de pesquisa, apresenta-se o estudo da complexidade dos algoritmos supracitados, aplicados em um programa na linguagem C. A implementação contou com a criação de um `TipoRegistro`, contendo uma `chave`, um `dado1` e um `dado2`, representantes em um registro.

Para a geração do arquivo, foi implementado um gerador de registros, com a definição de um valor de itens igual a 2 milhões. Assim, para a leitura de determinado número de registros, é especificado, na passagem de argumentos da chamada do programa, a quantidade de itens a serem lidos. Foram gerados 3 arquivos, dos quais um é ordenado ascendentemente, outro é ordenado descendentemente e o terceiro é desordenado aleatoriamente. Assim, segue-se o desenvolvimento da aplicação.

2) Desenvolvimento:

2.1 Análise dos Quesitos

2.1.1 Método main():

Dentro do método main, tem-se a chamada das funções para a realização de pesquisas externas. Assim, a nomenclatura desse módulo do programa é “principal.c”. Dentro dele, são definidas variáveis responsáveis pelo controle do menu e escolha das técnicas de busca, bem como das contagens de comparações e transferências realizadas em cada procedimento. São elas, nomeadas como no código fonte:

`int metodo:` é o tipo de pesquisa selecionado, sendo assim, os comandos são 1 pra Acesso Sequencial Indexado, 2 pra Árvore Binária de Pesquisa em Memória Externa, 3 pra Árvore B e 4 pra Árvore B*;

`long int quantidade:` quantidade de registros a serem pesquisados;

`int situacao:` é o modo como está ordenado o arquivo de registros, tal que 1 é representante de uma ordenação ascendente, 2 de uma ordenação descendente e 3 uma ordenação aleatória;

`int chave:` a chave a ser pesquisada dentro do arquivo;

`int contComparacoes:` sempre que a técnica de pesquisa fizer uma comparação de chaves, é incrementada a variável em questão, a fim de contar o número de vezes que essa operação é realizada;

`int contTransferencia:` sempre que a técnica de pesquisa abrir o arquivo externo para fazer uma transferência de dados (fopen), seja passar uma página completa para a memória principal ou para passar apenas um registro único, tal variável é incrementada a fim de contabilizar o número de vezes em que sãoi realizadas transferências entre os tipos de memória existentes;

`int controlePrintRegistro:` caso seja passado o argumento `[-P]` ao final do comando digitado pelo terminal, a variável `controlePrintRegistro` recebe 1, sendo essa opção a ordem pra imprimir todas as chaves do arquivo. Na situação de o valor dessa variável ser 1, chama-se a função `ImprimeChaves`, a qual aponta o cursor no arquivo para o início, varre o arquivo inteiro imprmindo os registros e retorna o cursor do arquivo para o início novamente.

As variáveis explicitadas são pra controle da pesquisa, a fim da realização de uma investigação e uma comparação acerca do tempo de processamento de cada método e os motivos que levam às respectivas diferenças. Ademais, também são responsáveis pela visualização de tais métricas e das chaves pesquisadas.

2.1.2 Acesso Sequencial Indexado:

Este método relaciona-se à pesquisa sequencial, na qual a menor chave de cada págica é ármazenada no vetor. Assim, a pesquisa externa baseada nesta técnica utiliza um arquivo para realizar a busca.

No primeiro tipo de pesquisa, o Acesso Sequencial Indexado, foi feita a implementação de acordo com os exemplos apresentados durante a aula. Porém, foi adicionada uma função para construir o índice das páginas e uma pequena modificação para contemplar os arquivos ordenados descendentemente.

Na pesquisa sequencial, foi adicionada uma verificação que finaliza a execução do código ao se fazer a chamada de um arquivo desordenado para que o método não seja chamado desnecessariamente.

2.1.3 Árvore Binária de Pesquisa Adequada à Memória Externa:

Uma árvore binária é uma estrutura de dados em que cada nó tem, no máximo, dois filhos e cada filho tem apenas um pai.

Utilizamos um algoritmo bem sucinto para o desenvolvimento deste método, que consiste no uso de 3 funções principais e 3 estruturas, sendo elas:

```
typedef struct{
    tipoindice indice;
    int esq;
    int dir;
}Nodo;

int insereArvExt(FILE *arvoreExt, tipoindice indice, int posicaoPai, int *contComparacoes, int *contTransferencias);
void criaArvoreExt(FILE *fp, int chave, int tamanho,int *contComparacoes, int *contTransferencias, int *achouRegistro, cloc
int pesquisa(FILE *fp, int valor, int posicao, int *achouRegistro, int *contComparacoes, int *contTransferencias);
```

```
typedef struct TipoRegistro{
    int chave;
    long int dado1;
    char dado2[5000];
} TipoRegistro;
```

```
typedef struct{
    int posicao;
    int chave;
} tipoindice;
```

A função `criaArvoreExt` é a principal do código, realizando a chamada das funções de inserção e pesquisa. Através de um `TipoRegistro` auxiliar, ela salva em um arquivo as informações necessárias para a criação da árvore, de modo que esse processo se repete uma quantidade de vezes igual o tamanho do arquivo original, realizando nessa parte do processo a chamada da função insere, que por sua vez realiza as comparações entre os valores para saber onde posicionar o item.

A função de pesquisa é bem simples, realizando apenas uma busca nos dados já ordenados da árvore e realizando as comparações até encontrar o item pedido.

2.1.4 Árvore B:

A estrutura de dados Árvore B é uma árvore de ordem `M`, na qual a raíz contém de `1` até `2M` itens, as demais páginas possuem de `M` a `2M` itens e de `M+1` a `2M+1` descendentes. Ademais, as páginas folhas estão todas no mesmo nível, e os registros estão ordenados de forma crescente dentro das páginas.

Dentre as definições utilizadas no módulo de cabeçalhos da estrutura em estudo está a configuração da ordem da árvore, bem como a ordem máxima de descendentes. Além disso, a estrutura `TipoPagina` contem os campos do `TipoRegistro`, armazenando o vetor de itens naquela pagina e de apontadores para seus descendentes.

```
#define M 5
```

```
#define MM 2 * M + 1

typedef long TipoChave;

typedef struct TipoPagina *TipoApontador;

typedef struct TipoPagina
{
    short n;
    TipoRegistro r[MM];
    TipoApontador p[MM + 1];
} TipoPagina;
```

Apresenta-se, também, as funções de inserção na árvore, bem como as responsáveis por seu funcionamento e aquela que realiza pesquisa externa utilizando a estrutura em questão:

```
void inicializa(TipoApontador *Arvore);

int PesquisaArvoreB(TipoRegistro *x, TipoApontador Ap, int
*contComparacoes, int *achouRegistro);

void InsereNaPagina(TipoApontador Ap, TipoRegistro Reg, TipoApontador
ApDir, int *contComparacoes);

void Ins(TipoRegistro Registro, TipoApontador Ap, int *Cresceu,
TipoRegistro *RegRetorno, TipoApontador *ApRetorno, int *contComparacoes);

void Insere(TipoRegistro Registro, TipoApontador *Ap, int
*contComparacoes);

void ArvoreB(TipoApontador *Arvore, FILE *arquivo, int quantidade,
TipoRegistro *registro, int *contComparacoes, int *contTransferencias, int
*achouRegistro);
```

Isso posto, a função `InsereNaPagina` adiciona um registro `TipoRegistro` na página `TipoApontador` passada como parâmtero. A função caminha na árvore iterativamente até encontrar a posição em que será inserido o novo item. Após a inserção, cada registro e ponteiro nas páginas é deslocado e a propriedade da árvore B é preservada.

Em se tratando da função `Ins`, essa é responsável pelo funcionamento da estrutura. Dessa forma, nela, é procurada uma posição na qual o novo registro deverá ser inserido. Essa função cria uma página caso a árvore seja nova e atualiza o valor da variável cresceu para sinalizar a criação desta e, assim continuar os procedimentos de inserção e balanceamento.

A função `Insere`, por sua vez, recebe o registro o qual deseja-se inserir e o apontador para a raíz da árvore. Em seguida, há a chamada da função `Ins` que acessa recursivamente a estrutura. Após essa chamada é verificado se a raíz cresceu e se é necessário criar uma nova página.

A função `PesquisaArvoreB` procura recursivamente o elemento ao percorrer na árvore. Sendo passada uma variável de controle `achouRegistro` por referência, a alteração dessa para valor igual a 1 indica o registro encontrado na estrutura pesquisada.

Por fim, a função `ArvoreB` organiza a passagem dos itens do arquivo para a estrutura da árvore B em si, chamando a função de inserção.

2.1.5 Árvore B*:

No método de busca em Árvore B*, ocorre uma adaptação na estrutura da já apresentada Árvore B. Em suma, é mantida a ideia de limite de valores em uma página e o limite também da quantidade de descendentes, mas existe a partição entre páginas internas e páginas externas. Assim, as primeiras contém apenas indicadores para o caminhamento nas ramificações, tendo em vista que as informações úteis encontram-se apenas nas páginas folhas, também chamadas de páginas externas. Desse modo, compreende-se que a busca nessa estrutura é efetiva para encontrar itens nos nós-folha de uma Árvore B*.

2.2 Análise dos Resultados

2.2.1 Acesso Sequencial Indexado

A tabela de análise no acesso sequencial indexado é feita apenas para arquivos na ordem ascendente e descendente.

Ordem Ascendente

ACESSO SEQUENCIAL INDEXADO – Ordem Ascendente – Nº Transferências					
100	200	2000	20000	200000	2000000
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x

ACESSO SEQUENCIAL INDEXADO – Ordem Ascendente – Nº Comparações					
100	200	2000	20000	200000	2000000
12	14	50	1004	1040	x
14	18	90	1008	1080	x
16	22	130	1012	1120	x
18	26	170	1016	1160	x
20	20	200	1020	1190	x

ACESSO SEQUENCIAL INDEXADO – Ordem Ascendente – T PréProcessamento					
100	200	2000	20000	200000	2000000
0.377s	0.865s	5.062s	26.571s	218.065s	x
0.362s	0.757s	5.928s	28.520s	199.200s	x
0.179s	0.829s	4.478s	29.272s	197.167s	x
0.356s	0.829s	5.358s	29.479s	196.952s	x
0.476s	0.754s	5.092s	30.429s	199.864s	x

ACESSO SEQUENCIAL INDEXADO – Ordem Ascendente – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
84.000ms	93,000ms	31.000ms	2293ms	1822ms	x
77.000ms	85.000ms	45.000ms	2801ms	1818ms	x
35.000ms	85.000ms	32.000ms	3747ms	1816ms	x
74.000ms	103.000ms	33.000ms	2393ms	1938ms	x
18.000ms	17.000ms	8.000ms	2989ms	1835ms	x

ACESSO SEQUENCIAL INDEXADO – Ordem Ascendente – T Total (s)					
100	200	2000	20000	200000	2000000
0.461s	0.958s	5.093s	28.864s	219.887s	x
0.439s	0.842s	5.973s	31.321s	201.018s	x
0.214s	0.914s	4.510s	33.019s	198.983s	x
0.356s	932ms	5.358s	31.872s	198.890s	x
0.494s	0.771ms	5.100s	33.418s	201.699s	x

Para valores acima de 200.000 foi observado uma falha de segmentação que não conseguimos tratar.

A partir das análises, percebe-se o aumento em um grau cada vez maior do tempo de processamento da estrutura. Para uma quantidade baixa de registros, o tempo é próximo a 0 segundos.

À medida em que o número de registros a serem nos testes aumenta, aumenta-se também o número de páginas e, portanto, o número de comparações durante a pesquisa é maior.

Como o acesso durante esse procedimento é limitado à arquivos ordenados, seu uso também é limitado, já que nem sempre tem-se o controle sobre os dados que estão sendo tratados,

Ordem Descendente

ACESSO SEQUENCIAL INDEXADO – Ordem Descendente – Nº Transferências					
100	200	2000	20000	200000	2000000
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x
101	201	2001	20001	200001	x

ACESSO SEQUENCIAL INDEXADO – Ordem Descendente – Nº Comparações					
100	200	2000	20000	200000	2000000
10	23	162	1020	1190	x
18	28	122	1016	1160	x
12	21	82	1012	1120	x
11	25	42	1008	1080	x
10	29	2	1004	1040	x

ACESSO SEQUENCIAL INDEXADO- Ordem Descendente – T PreProcessamento					
100	200	2000	20000	200000	2000000
0.337s	0.655s	5.197s	26.792s		
0.368s	0.805s	4.988s	28.520s		
0.395s	0.777s	5.118s	29.272s		
0.162s	0.770s	5.320s	29.479s		
0.338s	0.668s	4.638s	30.429s		

ACESSO SEQUENCIAL INDEXADO – Ordem Descendente – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
16.000ms	16.000ms	33.000ms	2104ms	x	x
18.000ms	20.000ms	35.000ms	x	x	x
19.000ms	26.000ms	34.000ms	x	x	x
7.000ms	23.000ms	39.000ms	x	x	x
26.000ms	16.000ms	34.000ms	x	x	x

ACESSO SEQUENCIAL INDEXADO – Ordem Descendente – T Total (s)					
100	200	2000	20000	200000	2000000
0.353s	0.641s	5.230s	28.896s	x	x
0.386s	0.825s	5.023s	x	x	x
0.414s	0.797s	5.152s	x	x	x
0.169s	0.793s	5.359s	x	x	x
0.364s	0.684s	4.672s	x	x	x

2.2.2 Árvore Binária de Pesquisa Adequada à Memória Externa:

Ordem Ascendente

Arvore Binária em memória secundaria – Ordem Ascendente – Nº Transferências					
100	200	2000	20000	200000	2000000
5720	15390	2005400	200054000	x	x
5290	15430	2005800	200058000	x	x
5310	15470	2006200	200062000	x	x
5330	15510	2006600	200066000	x	x
5350	15550	2007000	200070000	x	x

Árvore Binária em Memória Externa – Ordem Ascendente – Nº Comparações					
100	200	2000	20000	200000	2000000
10118	30038	4002398	400023998	x	x
10138	30078	4002798	400027998	x	x
10158	30118	4003198	400031998	x	x
10178	30158	4003598	400035998	x	x
10198	30198	4003998	400039998	x	x

Arvore Binária em memória externa – Ordem Ascendente – T PreProcessamento					
100	200	2000	20000	200000	2000000
6,469 s	7,580 s	290,470 s	26077,339 s	x	x
5,450 s	7,200 s	285,708 s	25722,466 s	x	x
4,459 s	7,368 s	275.238 s	26301,411 s	x	x
5,115 s	7,829 s	282,659 s	25890,234 s	x	x
5,891 s	7,919 s	289,246 s	26390.285 s	x	x

Árvore Binária em memória externa – Ordem Ascendente – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
76 ms	35 ms	107 ms	570 ms	x	x
43 ms	37 ms	116 ms	996 ms	x	x
45 ms	46 ms	155 ms	1479 ms	x	x
48 ms	48 ms	209 ms	1876 ms	x	x
58 ms	51 ms	259 ms	2395 ms	x	x

Árvore Binária em memória externa – Ordem Ascendente – T Total (s)					
100	200	2000	20000	200000	2000000
6,545 s	7,615 s	290.577 s	26077,909 s	x	x
5,493 s	7,237 s	285,824 s	25723,462 s	x	x
4,504 s	7,368 s	275,393 s	26302,890 s	x	x
5,163 s	7,877 s	282,868 s	25892,110 s	x	x
5,949 s	7,970 s	289,505 s	26392.680 s	x	x

Ordem Descendente

Árvore Binária em memória externa – Ordem Descendente – Nº Comparações					
100	200	2000	20000	200000	2000000
10179	40359	4003599	400035999	x	x
10159	40319	4003199	400031999	x	x
10139	40279	4002799	400027999	x	x
10119	40239	4002399	400023999	x	x
10099	40199	4001999	400019999	x	x

Árvore Binária em memória externa – Ordem Descendente – T PreProcessamento					
100	200	2000	20000	200000	2000000
5,714 s	24,757 s	278,549 s	26381,771 s	x	x
4,477 s	11,498 s	291,611 s	25657,789 s	x	x
5,470 s	9,239 s	281,198 s	26254,952 s	x	x
5,482 s	9,623 s	290,565 s	26671,466 s	x	x
5,007 s	11,463 s	289,770 s	26462,015 s	x	x

Ordem Aleatória

Árvore Binária em memória externa – Ordem Aleatória – Nº Transferências					
100	200	2000	20000	200000	2000000
736	1895	25760	352772	4595697	x
734	1898	25763	352775	4595685	x
735	1900	25763	352773	4595693	x
736	1898	25761	352775	4595697	x
735	1897	25764	352763	4595681	x

Árvore Binária em memória externa – Ordem Aleatória – Nº Comparações					
100	200	2000	20000	200000	2000000
1098	3063	44204	632827	8464882	x
1096	3066	44207	632830	8464873	x
1098	3069	44207	632828	8464881	x
1098	3066	44205	632830	8464885	x
1097	3065	44208	632819	8464887	x

Árvore Binária em memória externa – Ordem Aleatória – T PreProcessamento					
100	200	2000	20000	200000	2000000
3,062 s	4,858 s	25,315 s	204,015 s	2438,674 s	x
3,394 s	3,810 s	25,267 s	189,067 s	2677,897 s	x
3,825 s	5,359 s	25,039 s	196,657 s	2373,457 s	x
2,844 s	5,281 s	26,164 s	173,866 s	2339,989 s	x
3,465 s	4,158 s	25,128 s	181,127 s	2557,726 s	x

Árvore Binária em memória externa – Ordem Aleatória – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
105 ms	33 ms	26 ms	27 ms	38 ms	x
32 ms	25 ms	26 ms	25 ms	46 ms	x
179 ms	29 ms	27 ms	42 ms	34 ms	x
89 ms	30 ms	31 ms	25 ms	33 ms	x
89 ms	29 ms	29 ms	23 ms	29 ms	x

Árvore Binária em memória externa – Ordem Aleatória – T Total (s)					
100	200	2000	20000	200000	2000000
3,167 s	4,891 s	25,341 s	204,078 s	2438,712 s	x
3,426 s	3,835 s	25,293 s	189,092 s	2677,943 s	x
4,004 s	5,455 s	25,066 s	196,699 s	2373,491 s	x
2,933 s	5,311 s	26,195 s	173,891 s	2340,022 s	x
3,554 s	4,187 s	25,247 s	181,150 s	2557,755 s	x

Foi observado, nas situações descendentes e ascendentes a impossibilidade de se testar os casos de 200000 e 2000000 no computador em questão devido a uma limitação de memória.

Para os casos na estrutura de árvore binária, aponta-se que essa não mantém-se balanceada em sua criação e reconstituição. Dessa maneira, para arquivos ordenados ascendentemente ou descendentemente, a ramificação dos nós ocorre para um lado apenas, e a pesquisa torna-se custosa, principalmente para arquivos grandes. Esse fato justifica a falha da memória para arquivos ordenados de tamanho muito grande.

No caso de arquivos aleatórios, o problema de balanceamento discutido não se faz marcante, de modo que a eficiência do método é bem maior. Por conta desse motivo, os tempos de pesquisa tendem a serem menor, bem como o número de comparacões.

2.2.3 Árvore B:

Apresenta-se uma tabela com a documentação dos resultados obtidos pelos testes com amostras de 100, 200, 2 000, 20 000, 200 000 e 2 000 000 registros.

Ordem Ascendente

Árvore B – Ordem Ascendente – Nº Transferências					
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000

Árvore B – Ordem Ascendente – Nº Comparações					
100	200	2000	20000	200000	2000000
1890	4314	59232	419274	11876259	142817437
1890	4316	59234	419274	11876263	142817441
1890	4316	59234	419274	11876263	142817441
1890	4316	59234	419274	11876263	142817441
1890	4316	59234	419274	11876267	142817445

Árvore B – Ordem Ascendente – T PreProcessamento (s)					
100	200	2000	20000	200000	2000000
0	0,001	0,025	0,071	1,716	81,593
0	0	0,014	0,068	1,727	87,816
0	0,001	0,024	0,068	1,8	86,221
0	0	0,019	0,072	2,896	78,799
0,001	0	0,022	0,073	1,704	98,679

Árvore B – Ordem Ascendente – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
0	1	0	0	0	0
0	0	0	0	0	31
0	1	0	0	0	32
0	2	0	0	14	11
0	0	0	6	0	235

Árvore B – Ordem Ascendente – T Total (s)					
100	200	2000	20000	200000	2000000
0	0.002	0,025	0,071	1,716	81,539
0	0	0,014	0,068	1,727	87,847
0	0,002	0,024	0,068	1,8	86,253
0	0, 002	0,019	0,072	2,91	78,81
0,001	0	0,022	0,079	1,704	98,914

Ordem Descendente

Árvore B – Ordem Descendente – Nº Transferências					
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000

Árvore B – Ordem Descendente – Nº Comparações					
100	200	2000	20000	200000	2000000
1774	3975	50149	609119	7191813	81712280
1774	3975	50149	609119	7191813	81712291
1774	3975	50149	609119	7191813	81712290
1774	3975	50149	609119	7191813	81712295
1774	3975	50149	609119	7191813	81712305

Árvore B – Ordem Descendente – T PreProcessamento (s)					
100	200	2000	20000	200000	2000000
0,001	0,002	0.017	0.166	1.796	23,343
0,001	0,001	0.031	0.159	1.831	24,154
0,001	0,002	0.018	0.156	1.791	24,518
0,001	0,002	0.017	0.174	1.770	23.780
0,001	0,002	0.017	0.152	1.909	25.599

Árvore B – Ordem Descendente – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
1	0	0	0	0	16
0	1	0	0	0	0
1	0	1	0	0	0
0	1	1	1	0	0
1	1	0	0	0	16

Árvore B – Ordem Descendente – T Total (s)					
100	200	2000	20000	200000	2000000
0,002	0,002	0.017	0.166	1.796	23,359
0,001	0,002	0.031	0.159	1.831	24,154
0,002	0,002	0.019	0.156	1.791	24,518
0,001	0,003	0.018	0.175	1.770	23.780
0,002	0,003	0.017	0.152	1.909	25.615

Ordem Aleatória

Árvore B – Ordem Aleatória – Nº Transferências					
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	2000000

Árvore B – Ordem Aleatória – Nº Comparações					
100	200	2000	20000	200000	2000000
1839	4205	58227	699519	6825723	67256699
1839	4206	58233	699520	6825727	67256703
1852	4217	58247	699540	6825748	67256724
1852	4217	58247	699540	6825748	67256724
1852	4217	58247	699540	6825748	67256724

Árvore B – Ordem Aleatória – T PreProcessamento (s)					
100	200	2000	20000	200000	2000000
0.001	0,006	0.008	0.158	1.082	11.486
0.001	0,002	0.018	0.149	1.098	12.671
0	0,002	0.029	0.140	1.115	12.328
0	0,003	0.025	0.150	1.100	12.866
0,003	0,002	0.033	0.153	1.393	12.941

Árvore B – Ordem Aleatória – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
0	0	0	0	0	0
0	1	0	0	0	6
1	1	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0

Árvore B – Ordem Aleatória – T Total (s)					
100	200	2000	20000	200000	2000000
0.001	0.006	0.008	0.158	1.082	11,486
0.001	0.003	0.018	0.149	1.098	12,677
0.001	0.003	0.029	0.140	1.115	12,328
0	0.004	0.025	0.150	1.100	12,866
0,003	0.002	0.034	0.153	1.393	12,941

Em linhas gerais, tem-se que a quantidade de transferências nos processamentos da árvore B é igual ao número de registros lidos, e o crescimento do número de comparações é cada vez maior, conforme o número de registros pesquisados aumenta. Ademais, a quantidade de transferências e comparações mantiveram-se alto, mas as pesquisas foram realizadas de modo eficiente e rápido.

Comparando-se no quesito da ordenação dos registros, apresenta-se que os modos ascendente e descendente têm resultados próximos nas contagens, uma vez que a estrutura da Árvore B mantém-se balanceada durante os procedimentos de inserção.

Analisando-se o tempo de processamentos e pesquisas observa-se que o maior tamanho da árvore implica um maior tempo gasto durante a execução da criação da estrutura. O tempo de pesquisa manteve-se próximo de 0 segundos em todos os testes e por isso foi documentado na unidade de milissegundos (ms).

2.2.4 Árvore B*:

Apresenta-se uma tabela com a documentação dos resultados obtidos pelos testes com amostras de 100, 200, 2 000, 20 000, 200 000 e 2 000 000 registros.

ARVORE B ESTRELA – ORDEM ALEATÓRIA – T Total (s)					
100	200	2000	20000	200000	2000000
2,050 s	0,877 s	16,776 s	95,292 s	869,540 s	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – ORDEM ALEATÓRIA – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
78ms	16ms	18ms	16ms	15ms	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – ORDEM ALEATÓRIA – T PreProcessamento					
100	200	2000	20000	200000	2000000
1,972 s	0,861 s	16,758 s	95,276 s	869,525 s	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – ORDEM ALEATÓRIA – Nº Comparações					
100	200	2000	20000	200000	2000000
2035	4843	73580	997264	12646665	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – ORDEM ALEATÓRIA – Nº Transferências					
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Descendente – T Total (s)					
100	200	2000	20000	200000	2000000
1,947 s	3,521 s	16,457 s	117,893 s	841,828 s	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Descendente – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
82ms	24ms	17ms	14ms	16ms	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA - Ordem Descendente – T PreProcessamento					
100	200	2000	20000	200000	2000000
1,865 s	3,521 s	16,440 s	117,879 s	841,812 s	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Descendente – Nº Comparações					
100	200	2000	20000	200000	2000000
2035	4843	73580	997264	12646665	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Descendente – Nº Transferências					
100	200	2000	20000	200000	2000000
100	200	2000	20000	200000	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Ascendente – T Total (s)					
100	200	2000	20000	200000	2000000
2,144 s	4,141 s	18,125 s	102,203 s	965,485 s	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Ascendente – T Pesquisa (ms)					
100	200	2000	20000	200000	2000000
78ms	21ms	20ms	14ms	15ms	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Ascendente – T PréProcessamento					
100	200	2000	20000	200000	2000000
2,066 s	4,120 s	18.105 s	102,189 s	965,470 s	x
50	100	1000	10000	100000	1000000

ARVORE B ESTRELA – Ordem Ascendente – Nº Comparações					
100	200	2000	20000	200000	2000000
2035	4843	73580	997264	12646665	x
50	100	1000	10000	100000	1000000

No caso da Árvore B*, existe semelhança entre sua implementação com a Árvore B apresentada, de maneira que o custo de processamento seja similar. No quesito de comparações, tem-se que a diferenciação de nós externos e internos influencia nessa métrica, já que as páginas internas são meios apenas de navegação na árvore, e as comparações de todos os itens nessas páginas não são rigidamente necessários.

3) Conclusão:

Destarte, no presente projeto, foram implementados quatro (4) métodos distintos para a execução de uma pesquisa externa, aplicados em 6 conjuntos de registros diferentes, ordenados de três modos diferentes. Assim sendo, foi possível comparar os desempenhos de tais testes tendo em vista os quesitos de custo de memória, sendo simbolizado pelo número de comparações e número de transferências realizadas, e de custo de tempo, representado pelo cálculo do tempo de pesquisa somado ao tempo de processamento em cada estrutura de dados.

Em contrapartida, considerando a demanda de uma grande quantidade de memória em alguns casos, alguns testes não foram concluídos. Isso justifica-se pela capacidade de cada computador ter um processador ser diferente, e em alguns deles, a quantidade de memória utilizada não é suportada.

Analisando os testes, apresenta-se que a Árvore B é otimizada e balanceada, de maneira que as simulações de custos apresentassem melhores resultados ao serem analisados nos testes com essa estrutura. A partir da partição de metade dos apontadores para descendentes dos nós, é estabelecido um custo logarítmico, de modo que o tempo total é mais reduzido.

Da mesma forma, analisando os testes é possível notar que a tabela de índices é imprescindível para a eficiência da pesquisa sequencial indexada, pois auxilia muito na pesquisa externa. Seu ponto negativo é o fato do arquivo ter que estar ordenado para poder usar o método. Além disso, a complexidade na transferência é a melhor possível.

Dentre as dificuldades encontradas pelo grupo, estão os citados problemas com tamanhos dos arquivos, o desenvolvimento de técnicas sem o conhecimento concreto das implementações, buscando entender as limitações da estrutura, bem como o efeito em cadeia de erros de contagem e pesquisa que apareceram conforme a implementação foi sendo concluída. O grupo também juntou-se para concluir a implementação da árvore B*, a qual foi um grande desafio em questões de divisões de páginas e atualização dos ponteiros de encontro de registro.

Apesar dos desafios, a finalização foi resultado de um aprofundado processo investigativo, de modo a aplicar na prática os conhecimentos teóricos adquiridos em sala de aula na disciplina Estrutura de Dados II (BCC203). Assim, o grupo foi capaz de expandir e concretizar os saberes sobre implementação de acesso à memória externa para a busca de registros e dados.