Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Pesquisa Externa

BCC203 - Estrutura de Dados II

Bruno Alves Braga
Jéssica Machado
Kézia Alves Brito
Vitor Oliveira Diniz
Professor: Guilherme Tavares de Assis

Ouro Preto
27 de junho de 2023

Sumário

1 Introdução

Neste trabalho, foi necessário entregar o código em C ou C++ e um relatório referente ao que foi desenvolvido. Os algoritmos desenvolvidos foram a implementação dos métodos de pesquisa externa apresentados em aula, mais especificamente, Acesso Sequencial Indexado, Árvore Binária, Árvore B e Árvore B*.

A codificação foi feita utilizando alguns recursos do C++.

1.1 Especificações do problema

A pesquisa externa é empregada quando os dados a serem manipulados excedem a capacidade de armazenamento da memória interna disponível. Dessa forma, os dados são armazenados em memória secundária, em dispositivos como CDs, HDs externos, fitas e etc. Nesses casos, o programador, a fim de encontrar um dado em específico, realiza uma pesquisa externa no dispositivo que armazena várias outras informações.

Assim, devemos implementar um programa em linguagem C ou C++ que nos permita ler e pesquisar dados que estejam presentes em memória secundária e fazer a análise e a avaliação dos métodos de pesquisa externa.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code + *LiveShare*.
- Linguagem utilizada: C e C++.
- Compilador utilizado: g++ (GCC) *version* 13.1.1 20230429.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX.

1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *Valgrind*: ferramenta de análise dinâmica do código.

1.4 Especificações da máquina

As máquinas onde o desenvolvimento e os testes foram realizados possuem as seguintes configurações:

- **Computador:** Vitor.
- Processador: Ryzen 7-5800H.
- Memória RAM: 16 GB.
- Sistema Operacional: Arch Linux x86_64.
- **Computador:** Kézia.
- Processador: Intel Core i5.
- Memória RAM: 8 GB.
- Sistema Operacional: Manjaro Linux.

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
make
```

Usou-se para a compilação as seguintes opções:

- *-c*: para compilar o arquivo sem vincular os arquivos do tipo objeto.
- *-o*: para vincular os arquivos do tipo objeto.
- *-Wall*: para mostrar todos os possíveis *warnings* do código.
- *-g*: para compilar com informação de depuração e ser usada pelo *Valgrind*.

Tais diretrizes de compilação estão contidas no Makefile que se encontra na página deste Trabalho Pŕatico.

Para a execução do programa basta digitar:

```
./exe <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.

2 Desenvolvimento

Os métodos de pesquisa estabelecidos pelo professor foram: Acesso Sequencial Indexado, Árvore Binária, Árvore B e Árvore B*. A seguir, entraremos em detalhes sobre como implementamos cada um desses métodos, além de mostrar como geramos os arquivos utilizados e como realizamos o controle de performance de cada teste.

2.1 Criação dos Arquivos Binários

Para podermos iniciar a implementação dos métodos, primeiro foi necessário ter um arquivo binário para realização dos testes. Os arquivos poderiam ser ordenados ascendentemente, descendentemente ou aleatoriamente. Para isso, criamos a biblioteca *gerarRegistros.h* que é responsável por toda a geração de dados. Além do arquivo binário, geramos um texto de conteúdo igual para que, assim, possamos acompanhar e verificar manualmente os testes ocorridos e as chaves pesquisadas.

Para gerar os arquivos ordenados não há segredo, precisamos apenas adicionar o contador do *for* na chave de um registro. Já para as chaves aleatórias, como cada chave é única, decidimos gerar um vetor ordenado, com cada posição contendo uma chave, e assim utilizar um método próprio do C++ para embaralhar as posições desse vetor. Além disso, para cada dado do registro, pegamos uma posição aleatória do vetor contendo os caracteres alfanuméricos que julgamos válidos. Assim, inserimos esse registro em um arquivo texto para verificar se a geração dos arquivos estava correta.

Após a geração do arquivo texto, realizamos a conversão do conteúdo para um binário, tendo, assim, a certeza de que o arquivo de teste estava correto.

Para fins didáticos, diminuímos o tamanho das cadeias de caracteres, utilizamos 100 caracteres para o dado 2 e 300 para o dado 3. Assim, podemos ter resultados mais palpáveis nos testes realizados.

2.2 Métodos de Pesquisa Externa

2.2.1 Acesso Sequencial Indexado

No acesso sequencial indexado, utilizamos o conceito de paginação. Ele se baseia em ler páginas de arquivos, que nada mais é do que um número pré definido de registros. Dentro dessas páginas, verificamos se há a presença do registro com uma simples busca sequencial. Além disso, fizemos uma tabela de índices, em que lemos o primeiro índice de cada página e assim montamos uma tabela de busca onde, se uma chave se encontra entre dois indices da tabela, caso ela exista no meu arquivo, estará na página equivalente à tabela.

Vale lembrar que a tabela só funciona para arquivos ordenados. Caso o arquivo esteja com as chaves aleatórias, podemos apenas ler as páginas e procurar o índice em cada página até achar o item pesquisado.

2.2.2 Árvore Binária

Como pedido, a árvore binária deveria ter sua implementação adequada para arquivo externo. Para isso, primeiramente, fizemos o arquivo binário da árvore lendo todas as páginas do nosso arquivo binário original e, para cada registro encontrado, o inseríamos na árvore. A nossa árvore em arquivo é composta pelo registro e dois apontadores, que seria equivalente ao número do registro na árvore, sendo -1 para quando ele não tem filhos.

Após a criação do arquivo da árvore binária, podemos realizar a pesquisa. Como iriamos fazer na árvore comum, a partir da raiz olhamos se o registro procurado é maior ou menor que o nó atual e ir descendo até encontrar o elemento ou uma folha, em que não se pode ter mais registro e assim retornar o resultado da pesquisa.

2.2.3 Árvore B

Para a árvore B, temos um procedimento parecido com a árvore binária. Primeiro devemos criar a raíz e ler os registros presentes no arquivo binário, seguindo o esquema de paginação para acelerar a leitura dos dados. Em sequência, inserimos os dados lidos na árvore, levando em consideração as regras de inserção em uma árvore B, crescendo e subdividindo as páginas conforme a inserção atinge o limite da ordem.

2.2.4 Árvore B*

A árvore B* funciona de maneira semelhante à árvore B, se diferenciando pela separação de páginas internas (nós) e externas (folhas), onde as páginas nós funcionam como um índice, armazenando apenas chaves, enquanto as páginas folhas, que ficam no último nível da árvore, armazenam os registros propriamente ditos.

3 Análise

3.1 Acesso Sequencial Indexado

100 Registros - Ordem Crescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
25	2999800	110722	25	3
30	22236	18061	30	3
47	23672	17251	47	3
73	22239	18561	23	3
92	15113	10878	42	3
13	23335	19126	13	3
100	34514	31594	50	3
1	20437	13424	1	3
55	18711	13126	5	3
68	19674	14502	18	3

100 Registros - Ordem Decrescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
1000000	25527	2705360	1	3
999955	20559	14040	46	3
999925	14563	11847	26	3
999901	36746	27784	50	3
999961	15926	11118	40	3
999956	4376	4162	45	3
999921	3866	5958	30	3
999907	3958	4671	44	3
999996	3863	4226	5	3
999924	4247	4724	27	3

100 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
926061	22256	14002	1	3
111045	5538	5340	21	3
157642	6317	5291	37	3
553410	7839	10950	65	4
373781	7007	9100	79	4
593676	7079	7438	87	4
907429	5675	7236	92	4
152351	6223	7180	99	4
583569	6795	6696	12	3
246055	6480	6901	29	3

1000000 Registros - Crescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
956021	347578075	366135	21	20001
214397	28389900	151045	47	20001
327168	30661501	273146	18	20001
172612	31276828	276437	12	20001
192288	29190869	226220	38	20001
578358	29182249	252528	8	20001
711265	27699243	456377	15	20001
622336	39358509	294442	36	20001
405129	36929768	213304	29	20001
764583	40019426	471693	33	20001

1000000 Registros - Decrescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
45667	2151064804	855879	34	20001
402969	58739539	452992	32	20001
367163	59425613	435551	38	20001
335061	72707964	429963	40	20001
748594	75235499	1943571	7	20001
621980	78472421	777403	21	20001
439535	65051528	403717	16	20001
536295	75224643	429439	6	20001
214524	80908295	775391	27	20001
73551	71484590	703710	50	20001

1000000 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
354626	2067928288	926652167	966056	39322
613786	1018623459	791987188	710551	34212
415297	1170182127	857580236	809554	36192
928750	1184842476	129684745	114123	22283
14587	46582159	49228993	146837	22937
695611	45035256	153759761	266386	25328
780189	44311228	14204288	102375	22048
194004	45489360	742798515	908251	38166
704385	1089466524	304075594	253564	25072
567554	68310869	49913674	261376	25228

3.2 Arvore Binária

100 Registros - Crescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
81	3861591	34518	5111	5033
1	6459807	6425	4951	4953
36	4483873	18961	5021	4988
41	4415741	26066	5031	4993
76	5032896	39321	5101	5028
83	5017202	39395	5115	5035
51	4856643	24886	5051	5003
77	4822442	42690	5103	5029
8	6989988	9058	4965	4960
42	5887707	26164	5033	4994

100 Registros - Decrescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
999907	2957080	40756	5137	5046
999957	2744232	12047	5037	4996
999992	3115632	4683	4967	4961
999990	2685076	21497	4971	4963
999981	3227009	12070	4989	4972
999978	3493666	9555	4995	4975
999983	2884888	8915	4985	4970
999969	4185023	14957	5013	4984
999942	4784066	33228	5067	5011
999919	4263120	33904	5113	5034

100 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
926061	2259210	4844	749	751
111045	2308996	7036	765	759
157642	2896965	12241	765	759
553410	2613671	10643	769	761
373781	3167496	12645	769	761
593676	2759761	9415	765	759
907429	3156398	17581	765	759
152351	4239002	19708	769	761
583569	3325198	6237	761	757
246055	3167918	12375	761	757

1000000 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
164408	58998738954	38465	25660034	25680008
570954	72015891874	38651	25660028	25680005
179923	62166425836	41554	25660046	25680014
75621	52873230915	33005	25660038	25680010
944646	50465798373	35937	25660022	25680002
138860	51976835658	36189	25660032	25680007
335442	54161645366	35611	25660050	25680016
383703	52056503566	40039	25660038	25680010
634717	57754421219	31945	25660022	25680002
829067	56256714421	50553	25660034	25680008

3.3 Árvore B

100 Registros - Crescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
17	89199	678	1336	2
8	24872	197	907	2
7	23941	196	906	2
27	22049	120	899	2
47	21809	281	908	2
35	20343	154	908	2
56	20460	187	907	2
75	20998	292	906	2
94	20902	325	910	2
1	20962	287	905	2

100 Registros - Decrescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
999993	102431	501	1067	2
999980	26346	223	877	2
999995	24228	137	877	2
999948	28643	189	878	2
999937	21835	328	882	2
999986	22348	233	877	2
999904	21084	253	880	2
999929	21084	147	875	2
999932	20844	174	878	2
999991	20959	200	881	2

100 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
926061	2685702	357	1165	2
111045	22777	175	894	2
157642	17739	159	897	2
553410	16777	190	895	2
373781	17002	166	893	2
593676	20151	199	895	2
907429	16356	198	894	2
152351	16283	209	896	2
583569	16077	130	894	2
246055	16067	153	896	2

1000000 Registros - Crescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
460836	2506491660	3472	46710245	20000
816099	1764235979	3708	31181695	20000
998472	1650402324	1920	31181702	20000
717645	2350497758	6001	31181692	20000
697639	2004256636	4381	31181698	20000
308671	1083611170	3055	31181692	20000
325430	1234554162	2904	31181697	20000
432303	1275783915	2811	31181691	20000
532476	1683919616	2553	31181688	20000
393630	1309465055	2992	31181695	20000

1000000 Registros - Decrescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
867419	1264066038	3835	27437049	20000
970232	564912527	2583	30670863	20000
491235	506153512	3681	30670862	20000
491044	510134055	3024	30670863	20000
824059	706795461	2502	30670864	20000
194134	631271270	3156	30670861	20000
265234	625594313	3997	30670861	20000
548341	571884920	3272	30670864	20000
167854	580640847	2781	30670863	20000
88730	551145329	3008	30670860	20000

1000000 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
44307	2129813792	1197	33802679	20000
372212	1373132962	1595	31052239	20000
357170	1475228073	1345	31052234	20000
659122	1406505477	1449	31052233	20000
520954	1368689539	1499	31052232	20000
615661	1363732869	1483	31052239	20000
714867	1415517470	1081	31052229	20000
635094	1438835693	1958	31052233	20000
951124	1323633244	1735	31052241	20000
425	1232102224	2030	31052231	20000

3.4 Árvore B*

100 Registros - Crescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
97	3211842	478	974	2
81	114247	480	975	2
76	105942	233	973	2
74	124046	306	973	2
52	87272	364	972	2
69	100237	375	974	2
58	85092	210	971	2
36	92137	366	973	2
94	95422	312	974	2
96	91045	230	974	2

100 Registros - Decrescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
999998	164647	553	1025	2
999967	125859	389	1023	2
999959	140497	308	1022	2
999922	166106	412	1023	2
999961	166069	326	1023	2
999953	155018	341	1022	2
999921	174677	239	1022	2
999950	156338	375	1023	2
999997	157310	284	1024	2
999912	145466	311	1021	2

100 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
926061	135495	497	960	2
111045	144403	367	960	2
157642	107129	353	962	2
553410	103248	530	961	2
373781	98568	394	961	2
593676	103921	354	960	2
907429	104430	352	962	2
152351	100202	284	961	2
583569	100485	197	959	2
246055	101188	309	961	2

1000000 Registros - Crescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
670728	621712177	2525	43043582	20000
48575	533992271	3056	43043579	20000
967720	527946133	1643	43043586	20000
375854	526744398	2966	43043581	20000
91039	537825089	2304	43043582	20000
10569	516418039	3456	43043581	20000
919540	515854206	2505	43043582	20000
202746	513139754	7674	43043579	20000
780430	511002650	2244	43043584	20000
788816	522193942	3026	43043581	20000

1000000 Registros - Decrescente

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
319292	760610777	3096	27624098	20000
600192	664047236	3577	27624101	20000
700594	675965972	3116	27624099	20000
253973	682116046	3477	27624096	20000
184904	747247821	3086	27624100	20000
448932	734429228	3376	27624096	20000
293529	698157970	3987	27624097	20000
783339	662556007	4067	27624098	20000
802463	646770579	4719	27624099	20000
423118	659346163	3206	27624100	20000

1000000 Registros - Aleatório

Chave	T. Criação Árvore/Tabela(ns)	T. Pesquisa (ns)	Comparações	Transferências
848066	1471150420	1162	32073671	20000
897630	1359831249	1212	32073672	20000
46017	1353426640	1393	32073669	20000
399665	1361379183	1302	32073671	20000
399844	1375875669	1553	32073672	20000
110198	1384303129	1373	32073670	20000
227375	1259362297	1623	32073676	20000
253821	1268575516	891	32073669	20000
971076	1276583744	1513	32073672	20000
283714	1252859289	1583	32073672	20000

4 Conclusão

Inicialmente, ao fazer uma breve análise comparativa dos métodos, é possível perceber que quando se usa o sistema de paginações, o sistema de acesso sequencial indexado é bem eficiente, por mais que seja um dos piores métodos usados. Apresentamos um número máximo de comparações igual ao tamanho máximo de uma página, pois, ao achar o índice que pode conter o item, fazemos uma simples busca sequencial do registro desejado. Quanto ao número de transferências, podemos perceber que nas pesquisas com poucos registros no arquivo, sempre obtemos um número de transferências igual ao número de páginas totais do arquivo. Assim, podemos concluir que uma possível solução para aumentar a eficiência do acesso sequencial indexado seria simplesmente aumentar o tamanho da página, de modo que ela aproveite a memória interna mais eficientemente, e assim diminuímos o número de transferências. De todo modo, esse método só deve ser considerado caso você possua um arquivo já ordenado, pois, no momento que não podemos construir uma tabela de índices, ele se torna extremamente ineficiente, tendo uma complexidade de On para a busca e um tempo de pesquisa grande. Sendo esse problema ainda mais aparente com altas quantidades de registros.

Para a árvore binária, obtivemos um número de comparações um pouco excessivo, atribuímos isso às comparações realizadas ao construir a árvore. Na árvore binária adequada à memória externa, primeiramente temos que construir o arquivo da árvore, para depois inicializar a busca e, como vemos em nossos testes, a criação desse arquivo é de grande participação para a demora desse método. Assim, caso possuamos um arquivo binário já pronto, ele se demonstra um método bem efetivo mas possui alguns grandes problemas. Inicialmente, como ideia para diminuir o número de transferências, tentamos usar uma busca binária com páginação para ajudar a amenizar o número de transferências do pior caso, que seria On^2, quando a árvore só cresce para um lado, ordenada de forma crescente ou decrescente. Porém, tivemos alguns problemas e não conseguimos implementar a busca com paginação. Outro grande problema é que, devido à complexidade em seu pior caso, é inviável realizar essa busca com 1.000.000 de registros, pois a busca demoraria horas, sendo assim, não apresentamos a tabela para esse número de registros. Fora isso, seu caso médio a torna um excelente tipo de busca.

Para a árvore B, uma árvore n-ária que foi criada com o propósito de resolver vários problemas da árvore binária simples, temos que levar em consideração algumas regras para inserção, de modo que nossa árvore sempre será balanceada, e, assim, resolvemos o problema de pior caso. Podemos ver que, independente do número de registros, por ser uma árvore n-ária, sua busca é extramente rápida, às vezes não alcançando a casa dos 5 dígitos em nanosegundos. Por ter uma execução extremamente rápida de pesquisa, por enquanto seria o melhor método de busca. Sua dificuldade vem da sua implementação, em que se utiliza de métodos recursivos e da checagem de várias regras, tendo uma dificuldade um pouco mais elevada. Vemos que, como os registros couberam interamente na memória, o número de transferências se dá pelo número de páginas lidas, sendo assim, poderíamos aumentar a eficiência aumentando o número de registros por página, desde que se respeite o limite de memória do computador.

A árvore B* é uma leve melhoria da árvore B, inclusive, por esse motivo, a maioria dos bancos de dados implementa esse tipo de busca. A diferença entre as duas é que a B* implementa uma "falsa árvore"em cima das folhas, sendo uma árvore apenas de índices, contendo mais elementos do que a B em suas folhas e melhorando a eficiência de nossa busca. Assim como a árvore B, seu maior tempo de execução se dá pela criação da árvore em memória interna e não pelo baixíssimo tempo de pesquisa. Seu número de transferências se dá pela quantidade de leituras de páginas que fazemos do arquivo binário base. Assim como nos outros métodos, essa eficiência poderia ser melhorada pelo aumento do número de registros em uma página, já que a memória interna é consideravelmente mais rápida que a externa.

Quanto às dificuldades que encontramos nas implementações, podemos citar um certo estorvo no tamanho do arquivo binário, que, em sua primeira forma, possuia mais de 10 GB, travando em algumas execuções, e, por isso, por fins didáticos, decidimos diminuir drasticamente o tamanho das cadeias de caracteres dos dados. Como comentado anteriormente, para a árvore binária, em seu pior caso, tivemos uma vasta dificuldade em testar, porque sua complexidade aumenta o tempo de execução drasticamente, tornando quase impossível o teste em seu pior caso. Como o material didático carecia de informações sobre árvore binária adequada para busca externa, houve uma certa dificuldade em sua implementação. A árvore B teve a implementação mais tranquila, pois havia o conteúdo bem documentado do professor, que facilitou o entendimento do método. A árvore B*, por se basear na árvore B, não possuía detalhes de sua implementação, nos forçando a desenvolver o método com todas as novas regras de inserção e crescimento de nós. Por ser um método recursivo, possui uma complexidade extremamente elevada de implementação. Devido a seu alto número de nós e folhas, e por possuir uma "árvore falsa"em cima das folhas, em alguns testes acabamos ficando sem memória, travando o sistema operacional.

Percebemos, também, que em alguns testes obtivemos um valor elevado de comparações, às vezes sendo maior que o número de registros, mas podemos ver que, pelo tempo de execução, o método é bem eficiente e podemos atribuir esse problema a um leve erro no contador que utilizamos para as comparações.