Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

Ciência da Computação

Métodos de Ordenação externa

BCC203 - Estrutura de Dados II

Felipe Marques, Lucas Chagas, Matheus Peixoto, Nicolas Mendes, Pedro Henrique de Oliveira, Pedro Morais

Professor: Guilherme Tavares de Assis

Ouro Preto
7 de agosto de 2023

Sumário

1 Introdução **1**
1.1 Especificações do trabalho prático . . . 1
1.2 Considerações iniciais . . . 1
1.3 Ferramentas utilizadas . . . 1
1.4 Especificações da máquina . . . 1
1.5 Instruções de compilação e execução . . . 2

2 Desenvolvimento **3**
2.1 Estruturas utilizadas . . . 3
2.2 QuickSort Externo . . . 3
2.3 Geração de blocos por ordenação interna . . . 5
2.4 Geração de blocos usando seleção por substituição . . . 5
2.5 Intercalação Balanceada . . . 5

3 Experimentos e Resultados **7**
3.1 Descrição dos experimentos realizados . . . 7
3.2 Abreviações utilizadas nas tabelas: . . . 7
3.3 Utilizando um arquivo de 100 registros: . . . 8
3.4 Utilizando um arquivo de 1.000 registros: . . . 9
3.5 Utilizando um arquivo de 10.000 registros: . . . 10
3.6 Utilizando um arquivo de 100.000 registros: . . . 11
3.7 Utilizando um arquivo de 471705 registros: . . . 12

4 Análise individual das métricas de cada método **12**
4.1 Intercalação 2f fitas (ordenação interna na geração dos blocos) . . . 12
4.2 Intercalação 2f fitas (seleção por substituição na geração dos blocos) . . . 13
4.3 QuickSort externo . . . 13

5 Conclusão **14**
5.1 Análise comparativa de desempenho entre os métodos . . . 14
5.2 Principais dificuldades na implementação dos métodos . . . 15

Lista de Códigos Fonte

1 Struct TipoRegistro . . . 3
2 Struct Fita . . . 3
3 Struct Intercalacao . . . 3
4 Struct RegistroParaSubstituicao . . . 3
5 Struct TipoArea . . . 4
6 Métodos para estrutura TipoArea . . . 4

1 Introdução

1.1 Especificações do trabalho prático

O objetivo principal do trabalho prático consiste na implementação de 3 distintos métodos de ordenação externa, como forma de executar um estudo com um viés pragmático e objetivo acerca dos conhecimentos apresentados em sala de aula. Os métodos de ordenação em questão correspondem à intercalação balanceada de vários caminhos (2f fitas), utilizando um método de ordenação interno para geração dos blocos inicialmente ordenados, intercalação balanceada de vários caminhos (2f fitas), utilizando a técnica de seleção por substituição e ao Quicksort externo.

Além disso, o trabalho prático foi dividido em duas fases, em que a primeira corresponde a implementação dos métodos supracitados, em termos de código, e a segunda à execução de uma roda de testes, considerando o arquivo "PROVAO.TXT", fornecido pelo docente, variando a situação de ordenação do mesmo, a quantidade de itens a serem ordenados e o método de ordenação escolhido.

Ademais, denota-se relevante a descrição de algumas especificações referentes aos processos de ordenação. Sendo elas: memória interna disponível de, no máximo, 20 itens e 40 fitas de armazenamento externo, sendo 20 de entrada e 20 de saída, enfatizando que as fitas foram simuladas por meio de arquivo binários.

Por fim, ressalta-se as estratégias inicialmente utilizadas com relação a manipulação do arquivo originalmente fornecido (ordenado aleatoriamente), em que houve a preferência de convertê-lo em binário e, por meio de um Quicksort interno, gerar o arquivo ordenado crescentemente, e, utilizando uma função que inverte os registros, gerar o arquivo ordenado de modo decrescente.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code. [1]
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [2]

1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

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

1.4 Especificações da máquina

A máquina utilizada para a realização dos testes possui a seguinte especificação As máquinas utilizadas para a realização dos testes possuem as seguintes especificações:

- Intel Core i5 9ª geração
- 16 GB de RAM
- SSD NVMe
- WSL utilizando o Ubuntu 22.04.2 LTS

[1] Visual Studio Code está disponível em https://code.visualstudio.com
[2] Disponível em https://www.overleaf.com/

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
gcc -Wall *.c -o exe -g -lm
```

Usou-se para a compilação as seguintes opções:

- *-Wall*: para mostrar todos os possível *warnings* do código.

- *-lm*: vinculação da biblioteca matemática padrão *(libm)*.

- *-o exe*: informa o nome do arquivo de saída *(exe)*.

- *-g*: inclui informações de depuração.

Para a execução do programa basta digitar:

```
./(nome executável) (método a ser usado) (quantidade de itens no arquivo) (situação da ordem do arquivo) [-P]
```

2 Desenvolvimento

2.1 Estruturas utilizadas

A estrutura TipoRegistro contém os dados referentes a cada linha de dados do PROVAO.TXT, tendo o valor da chave, nota, estado, cidade e curso, seguindo as especificações de tamanho passsadas na instrução do trabalho.

```
typedef struct{
    TipoChave Chave;
    double nota;
    char estado[3];
    char cidade[51];
    char curso[31];
}TipoRegistro;
```

Código 1: Struct TipoRegistro

A estrutura Fita possui informações sobre as fitas, como a quantidade de blocos que ela possui; um vetor de inteiros correspondente à quantidade de itens de cada bloco, sendo este vetor dinâmico e alocado durante a execução do código; um ponteiro para o arquivo ao qual a estrutura se refere e, por fim, um enum que representa se a fita é de entrada ou saída.

```
enum TipoFita {ENTRADA = 1, SAIDA = 2};
typedef struct{
    int n_blocos;
    int * qtdItensBloco;
    FILE * arq;
    enum TipoFita tipo;
}Fita;
```

Código 2: Struct Fita

A estrutura Intercalacao possui informações úteis para o processo de intercalação dos blocos, contendo um campo com a quantidade de itens que já foram lidos da fita, um dado do tipo TipoRegistro e um boolean que indica se a fita está ativa ou não.

```
typedef struct{
    int qtdItensLidos;
    TipoRegistro dadoLido;
    bool fitaAtiva;
}Intercalacao;
```

Código 3: Struct Intercalacao

A estrutura RegistroParaSubstituicao possui dados que são utilizados para a geração de blocos com o método substituição por seleção, pois ela possui um campo do tipo TipoRegistro e um boolean para identificar se o item está ou não marcado, sendo que o mesmo fica marcado ao entrar na memória quando ele é menor que o último valor que saiu.

```
typedef struct {
    TipoRegistro registro;
    bool marcado;
} RegistroParaSubstituicao;
```

Código 4: Struct RegistroParaSubstituicao

2.2 QuickSort Externo

Para implementação do método de ordenação QuickSort externo, adotamos como ponto de partida o código apresentado em sala de aula, que inicialmente estava funcional para um arquivo que possuía um vetor de inteiros.

Inicialmente, o processo de concretização do método deu-se pela adaptação direta do código, originalmente fornecido pelo docente, para a estrutura principal do trabalho prático (TipoRegistro), presente no arquivo também fornecido.

Posteriormente, o enfoque deu-se na questão central da construção de uma estrutura de dados que, de modo eficiente, suprimisse as demandas e atendesse as etapas necessárias para a manutenção do pivô dinâmico do método, conceito esse introduzido em sala de aula.

Seguidamente, optamos pela constituição da estrutura TipoArea, que possui um vetor da estrutura TipoRegistro e um int n, que representa o número de registros presentes na estrutura. Além disso, destaca-se o uso da alocação estática do vetor, que minimiza o risco de uma má gestão de memória alocada dinamicamente.

```
typedef struct TipoArea{
    TipoRegistro area [TAMAREA];
    int n;
}TipoArea;
```

Código 5: Struct TipoArea

Ainda sobre a estrutura TipoArea, ressalta-se a construção de alguns métodos relacionados a estrutura, que são utilizados no código do QuickSort externo apresentado, como a função InsereItem (insere um registro no final do vetor e incrementa n em uma unidade), ObterNumCelOcupadas (retorna a quantidade de registros presentes na estrutura), RetiraUltimo (remove o ultimo item presente na estrutura) e RetiraPrimeiro (remove o primeiro item presente na estrutura). Vale ressaltar ainda que essas operações são utilizadas nas etapas de gerenciamento do pivô, estrutura central para a lógica de ordenação do algoritmo, como nas tarefas de preenchimento do pivô e remoção de um item (requisitada no instante em que o pivô está totalmente preenchido).

```
TipoArea inicializaArea(){
    TipoArea area;
    area.n = 0;
    return area;
}

void InsereItem(TipoRegistro UltLido, TipoArea *Area){
    Area -> area[Area->n] = UltLido;
    Area -> n ++;
    quicksort_interno(Area->area, 0, Area->n - 1);
}

int ObterNumCelOcupadas(TipoArea * area){
    return area->n;
}

void RetiraUltimo(TipoArea * area, TipoRegistro * R){
    *R = area->area[area->n-1];
    area->n --;
}

void RetiraPrimeiro(TipoArea * area, TipoRegistro * R){
    *R = area->area[0];

    for(int i = 0; i < area->n; i++)
        area->area[i] = area->area[i+1];

    area->n--;
}
```

Código 6: Métodos para estrutura TipoArea

Em resumo, a adaptação da estrutura a ser ordenada e a composição de uma estrutura eficiente e funcional para o pivô foram as principais alterações presentes na construção da implementação do método de ordenação QuickSort externo.

2.3 Geração de blocos por ordenação interna

Para a geração de blocos usando um método de ordenação interna foi criado um vetor do tipo TipoRegistro com 20 posições para representar a memória interna. Em seguida, calculamos a quantidade total de blocos que será gerada com base na quantidade total de itens que se deseja ordenar, dividimos essa quantidade por 20 e arredondamos o valor para cima, uma vez que o último bloco não precisa ter necessariamente 20 itens.

Posteriormente, fazemos um looping para ler todos os blocos. A cada repetição do for, lemos 20 registros do arquivo, exceto na última repetição já que será feita a leitura do último bloco, e armazenamos no vetor que representa a memória interna. Depois, ordenamos esse vetor utilizando o método do QuickSort, e escrevemos esse bloco de registros na fita de entrada atual. Nesse momento, a variável que armazena o número de blocos dessa fita é incrementada em 1.

Após a geração de todos os blocos nas fitas de entrada, atualizamos o vetor que armazena a quantidade de itens em cada bloco de todas as fitas. Assim, o processo de geração de blocos por ordenação interna está finalizado.

2.4 Geração de blocos usando seleção por substituição

Para a geração de blocos usando a técnica de seleção por substituição foi criada uma nova struct, na qual contém o registro em si e uma variável booleana para representar se o registro estará marcado ao entrar na memória interna.

No início do método foi criado um vetor de 20 itens com o tipo da struct acima para representar a memória interna. Posteriormente, fazemos a leitura dos itens inicias no arquivo, na qual é definida verificando se a quantidade total de itens para se ler é maior ou igual a 20, caso seja, é feita a leitura inicial de 20 itens, caso contrário, é feita a leitura do total de itens. Fazemos essa leitura a partir de um vetor auxiliar do tipo TipoRegistro, e passamos para o vetor da memória interna, setando todos com a variável "marcado"= false.

Em seguida, fazemos um looping onde acontece a retirada do menor item da memória interna, a escrita do mesmo na fita de entrada atual, a leitura do próximo item no arquivo e a inserção do mesmo no vetor da memória interna. Para a retirada do menor item utilizamos uma busca simples no vetor, tendo em vista que o mesmo possui apenas 20 posições, e, portanto, utilizando essa busca nós temos um código simples e que se mantêm eficiente. No momento em que ocorre a leitura do próximo registro, antes da inserção do mesmo na memória interna, é feita a verificação se o mesmo tem a nota menor que o último registro que saiu, caso tenha, esse próximo registro terá sua variável "marcado"setada como true. E, concluindo o looping, em cada repetição do mesmo é feita a verificação para ver se a memória interna está ocupada apenas por registros marcados, caso esteja, a mesma tem todos os seus registros desmarcados, e passa para a próxima fita de entrada, atualizando a fita anterior a respeito do seu número de blocos, armazenando também com quantos itens aquele bloco foi finalizado.

Após ler todos os itens do arquivo, saímos do looping anterior, e possuímos a memória interna com os itens que restam para serem escritos nas fitas. Então, ordenamos o vetor da memória interna utilizando o método de ordenação QuickSort, considerando tanto a marcação dos registros, quanto suas notas, e, posteriormente, escrevemos os itens na fita de entrada atual. Ao longo da passagem pelo vetor no momento de escrever os registros na fita, caso o registro na posição i esteja marcado, significa que todos os próximos também estarão marcados, e, dessa forma, a fita atual é atualizada de acordo com o número de blocos e número de itens de cada bloco, e então, o resto dos itens são escritos na fita seguinte.

Por último, a última fita, na qual teve registros escritos na mesma, também é atualizada em relação aos seus blocos. Assim, o processo de geração de blocos usando a seleção por substituição é finalizado.

2.5 Intercalação Balanceada

Para a intercalação balanceada, primeiro são criadas duas variáveis para controle das fitas de entrada e saída, sendo que começam com 0 e 20, respectivamente.

Depois é criado uma variável chamada de 'passada'. Dessa forma, supondo que todas as fitas possuem blocos, a passada irá representar quais blocos das fitas que estão sendo verificados.

Assim, quando verificar todos os blocos de determinada posição, seu valor será incrementado. Ademais, a variável também será fundamental para identificar qual a fita de saída.

Em seguida, é verificado quantos blocos existem ao total, uma vez que, em sequência, será executado um while, que chamaremos de "while 1", onde o mesmo repetirá até que a quantidade total de blocos somando todas as fitas seja exatamente um.

Dentro desse while, irá ser verificado quantas fitas que possuem blocos, pois elas participarão do processo de intercalação, evitando a leitura de fitas com dados inválidos. Ademais Também será executado um "while 2", que será válido enquanto ainda houver uma fita que possua blocos.

Dentro do "while 2", primeiro será iniciado um vetor de 20 posições do tipo Intercalação, para que tenha um registro dos dados para a intercalação. Em seguida, teremos o cálculo para saber qual será a fita de saída, sendo o mesmo: fitaSaida = ((saida + passada - 1) % 20) + saida; Sendo que saida representa a primeira fita de saída no vetor de fitas, a passada é a atual e o decréscimo de um é feito pelo fato do valor inicial da passada ser 1. Todavia, caso todas as 20 fitas já tivessem um bloco, a próxima fita de saída seria, novamente a primeira, logo, faz-se necessário o uso do 'mod' 20 para representar uma espécie de lista circular. Ademais, caso a fita de saída seja a 20, o cálculo atual iria considerar as fitas de entrada como sendo as de destino, o que estaria errado e atrapalharia o processo. Assim, o valor é incrementado do valor da posição da primeira fita de saída, para que o destino seja devidamente marcado. Também não haverá problemas quando a fita de saída for as 20 primeiras, pois o valor de saída estará marcado como zero.

Em seguida, todos os primeiros valores dos blocos da passada atual são lidos e, então entramos em um "while 3"que será executado até que o valor fitaAtiva de todos os valores do vetor da estrutura Intercalacao seja marcado como falso.

Dessa forma, enquanto houver valores para serem lidos das fitas, será procurado a posição no vetor de Intercalacao onde a menor nota está para escrevê-la na fita de saída. E caso todos os dados de um bloco tenham sido lidos e escritos, o bloco tornará inválido e a quantidade de blocos nesta fita será diminuído em um, já que, depois, a quantidade de blocos da fita será útil para saber se o "while 2"continuará a ocorrer, já que ele precisa saber quantas fitas com pelo menos um bloco existe. Porém caso ainda exista itens na fita, o próximo item da fita será lido.

Ao sair do "while 3", a passada é acrescida em um, já que a mesma foi realizada, e as fitas de saída que receberam valores terão a sua quantidade de blocos acrescida em um, e o vetor que indica a quantidade de itens em cada bloco receberá o valor correspondente à quantidade de itens inseridos. Em sequência, a quantidade de fitas é recalculada e o "while 2"é recomeçado.

Ao terminá-lo, as fitas de entrada são convertidas em fitas de saída e as de saída são convertidas em entrada a partir da sinalização das variáveis entrada e saida. Por fim, a quantidade de blocos para as fitas de entrada são recalculados e retornamos para o "while 1".

Com o fim deste while, os valores da fita de saída que possui o único bloco serão copiados para um arquivo .txt, para que o seu conteúdo possa ser visualizado sem grande dificuldades.

3 Experimentos e Resultados

3.1 Descrição dos experimentos realizados

Para executar a rodada de testes demandada pelo documento orientador do trabalho prático, adotamos a estratégia de realizar as execuções, mantendo a quantidade de itens a serem ordenados, e variando a situação de ordenação do arquivo e o método escolhido, efetuando essa estratégia para cada quantidade de itens descrita.

Com isso, na análise dos resultados obtidos, viabilizasse um julgamento de inferência, pautado nas especificidades de cada método, dado que a quantidade de itens é constante e promovendo um estudo substancialmente eficiente a respeito do desempenho de cada método.

Posteriormente, todas as métricas obtidas foram agrupados e dispostos em tabelas, separadas de acordo com a quantidade de itens a serem ordenados e a situação da ordenação do arquivo.

3.2 Abreviações utilizadas nas tabelas:

- $Q.L$ = Quantidade de leitura
- $Q.L.G.B$ = Quantidade de leitura na geração de blocos
- $Q.L.T$(bin txt) = Quantidade de leitura na transformação de .bin para .txt
- $Q.L.T$ = Quantidade de leituras totais
- $Q.E$ = Quantidade de escrita
- $Q.E.G.B$ = Quantidade de escrita na geração de blocos
- $Q.E.T$(bin txt) = Quantidade de escrita na transformação de .bin para .txt
- $Q.E.T$ = Quantidade de escritas totais
- $Q.C.N$ = Quantidade de comparações entre notas
- $T.EXC$ = Tempo de execução (segundos)

3.3 Utilizando um arquivo de 100 registros:

Arquivo Ordenado Crescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	107	7	101
Q.L.G.B	5	81	-
Q.L.T(bin txt)	5	5	5
Q.L.T	117	93	106
Q.E	106	6	100
Q.E.G.B	5	100	-
Q.E.T(bin txt)	100	100	100
Q.E.T	211	206	201
Q.C.N	600	1680	7258
T.EXC	0.006478s	0.004913s	0.001266s

Arquivo Ordenado Descrescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	107	107	101
Q.L.G.B	5	81	-
Q.L.T(bin txt)	5	5	5
Q.L.T	117	193	106
Q.E	106	106	101
Q.E.G.B	5	100	-
Q.E.T(bin txt)	100	100	100
Q.E.T	211	306	201
Q.C.N	825	1683	7752
T.EXC	0.009877s	0.003212s	0.001146s

Arquivo Aleatório:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	107	107	198
Q.L.G.B	5	81	-
Q.L.T(bin txt)	5	5	5
Q.L.T	117	193	203
Q.E	106	106	198
Q.E.G.B	5	100	-
Q.E.T(bin txt)	100	100	100
Q.E.T	211	306	298
Q.C.N	958	1501	8766
T.EXC	0.005822s	0.002027s	0.001811s

3.4 Utilizando um arquivo de 1.000 registros:

Arquivo Ordenado Crescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	2052	52	1001
Q.L.G.B	50	981	-
Q.L.T(bin txt)	50	50	50
Q.L.T	2152	1083	1051
Q.E	2051	51	1001
Q.E.G.B	50	1000	-
Q.E.T(bin txt)	1000	1000	1000
Q.E.T	3101	2051	2001
Q.C.N	13700	19680	81058
T.EXC	0.009693	0.009649	0.009262

Arquivo Ordenado Descrescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	2052	2052	1001
Q.L.G.B	50	981	-
Q.L.T(bin txt)	50	50	50
Q.L.T	2152	3083	1051
Q.E	2051	2051	1001
Q.E.G.B	50	1000	-
Q.E.T(bin txt)	1000	1000	1000
Q.E.T	3101	4051	2001
Q.C.N	23096	32456	86716
T.EXC	0.013391	0.016083	0.007101

Arquivo Aleatório:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	2052	2052	4638
Q.L.G.B	50	981	-
Q.L.T(bin txt)	50	50	50
Q.L.T	2152	3083	4688
Q.E	2051	2051	4638
Q.E.G.B	50	1000	-
Q.E.T(bin txt)	1000	1000	1000
Q.E.T	3101	4051	5638
Q.C.N	24180	31756	117173
T.EXC	0.023580	0.012580	0.019738

3.5 Utilizando um arquivo de 10.000 registros:

Arquivo Ordenado Crescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	30502	502	10001
Q.L.G.B	500	9981	-
Q.L.T(bin txt)	500	500	500
Q.L.T	31502	10983	10501
Q.E	30501	501	10001
Q.E.G.B	500	10000	-
Q.E.T(bin txt)	10000	10000	10000
Q.E.T	41001	20501	20001
Q.C.N	223000	199680	819058
T.EXC	0.081968s	0.028108s	0.068159s

Arquivo Ordenado Descrescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	30502	20502	10001
Q.L.G.B	500	9981	-
Q.L.T(bin txt)	500	500	500
Q.L.T	31502	30983	10501
Q.E	30501	20501	10001
Q.E.G.B	500	10000	-
Q.E.T(bin txt)	10000	10000	10000
Q.E.T	41001	40501	20001
Q.C.N	386466	419415	881389
T.EXC	0.066030s	0.04151s	0.066676s

Arquivo Aleatório:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	30502	20502	70123
Q.L.G.B	500	9981	-
Q.L.T(bin txt)	500	500	500
Q.L.T	31502	30983	70623
Q.E	30501	20501	70123
Q.E.G.B	500	10000	-
Q.E.T(bin txt)	10000	10000	10000
Q.E.T	41001	40501	80123
Q.C.N	411638	444635	1430751
T.EXC	0.047434s	0.072365s	0.251964s

3.6 Utilizando um arquivo de 100.000 registros:

Arquivo Ordenado Crescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	305002	5002	100001
Q.L.G.B	5000	99981	-
Q.L.T(bin txt)	5000	5000	5000
Q.L.T	315002	109983	105001
Q.E	305002	5001	100001
Q.E.G.B	5000	100000	-
Q.E.T(bin txt)	100000	100000	100001
Q.E.T	410001	205001	100000
Q.C.N	2904350	1999680	8840982
T.EXC	0.464657s	0.25220s	0.60338s

Arquivo Ordenado Descrescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	305002	205002	100001
Q.L.G.B	5000	99981	-
Q.L.T(bin txt)	5000	5000	5000
Q.L.T	315002	309983	105001
Q.E	305001	205001	100001
Q.E.G.B	5000	100000	-
Q.E.T(bin txt)	100000	100000	
Q.E.T	410001	405001	200001
Q.C.N	4742322	4895142	8839567
T.EXC	0.421266s	0.344553s	0.601545s

Arquivo Aleatório:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	305002	305002	876751
Q.L.G.B	5000	99981	-
Q.L.T(bin txt)	5000	5000	5000
Q.L.T	315002	409983	881751
Q.E	305001	305001	876751
Q.E.G.B	5000	100000	-
Q.E.T(bin txt)	100000	100000	100000
Q.E.T	410001	505001	976751
Q.C.N	5481275	5819382	17760405
T.EXC	0.396322s	0.399807s	3.008466s

3.7 Utilizando um arquivo de 471705 registros:

Arquivo Ordenado Crescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	1910408	23588	471706
Q.L.G.B	23586	471686	-
Q.L.T(bin txt)	23586	23586	23586
Q.L.T	1957580	518860	495292
Q.E	1910407	23587	471706
Q.E.G.B	23586	471705	-
Q.E.T(bin txt)	471705	471705	471705
Q.E.T	2405698	966997	943411
Q.C.N	15741913	9433783	41710040
T.EXC	1.726162s.	1.153091s	2.879335s

Arquivo Ordenado Descrescente:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	1910408	1438703	471706
Q.L.G.B	23586	471686	-
Q.L.T(bin txt)	23586	23586	23586
Q.L.T	1957580	1933975	495292
Q.E	1910407	1438702	471706
Q.E.G.B	23586	471705	-
Q.E.T(bin txt)	471705	471705	471705
Q.E.T	2405698	2382112	943411
Q.C.N	25413394	25757331	41709305
T.EXC	2.173459s.	1.865122s	2.837897s

Arquivo Aleatório:

Método	Intercalação 2f (ordenação interna)	Intercalação 2f (seleção por substituição)	QuickSort
Q.L	1910408	1910408	4403216
Q.L.G.B	23586	471686	-
Q.L.T(bin txt)	23586	23586	23586
Q.L.T	1957580	2405680	4426802
Q.E	1910407	1910407	4403216
Q.E.G.B	23586	471705	-
Q.E.T(bin txt)	471705	471705	471705
Q.E.T	2405698	2853817	4874921
Q.C.N	30098881	32608154	91504382
T.EXC	2.246830s	2.099823s	16.958632s

4 Análise individual das métricas de cada método

4.1 Intercalação 2f fitas (ordenação interna na geração dos blocos)

Analisando os dados experimentais obtidos na execução do método de intercalação 2f, utilizando ordenação interna para geração dos blocos inicialmente ordenados, nota-se, em um primeiro momento um fato interessante, que corresponde à preservação das métricas de leitura e escrita independentemente da situação de ordenação do arquivo (além do número de comparações não apresentar grandes variações com relação a quantidade de itens a serem ordenados), o que viabiliza a inferência de que o método, em termos de desempenho e eficiência, não obtém

ganhos substanciais em função da situação de ordenação do arquivo, denotando previsibilidade e consistência no que refere-se aos gastos computacionais envoltos para a sua execução.

Seguidamente, outro fato notável diz respeito ao número de leituras na etapa de geração dos blocos do método, que apresenta visível discrepância com o método que usa seleção por substituição para a geração de blocos. Alinhado com a base teórica, tal fato é resultado da implicação dos blocos terem tamanhos fixos, permitindo que mais itens sejam lidos simultaneamente, reduzindo o número de leituras.

Com relação ao tempo de execução, evidencia-se que o fator situação de ordenação do arquivo não interfere significativamente em tal métrica, pois, fixando a quantidade de registros, os melhores tempos variaram entre situações de ordenação distintas, além da variação ser mínima, permitindo a inferência de que, dado o fator supracitado relacionado ao tamanho dos blocos, a natureza da disposição da entrada é responsável por essas minúsculas flutuações, uma vez que as demais métricas avaliadas permaneceram constantes, como foi descrito.

4.2 Intercalação 2f fitas (seleção por substituição na geração dos blocos)

Analisando os resultados obtidos na execução dos testes para o método de intercalação 2f, utilizando seleção por substituição para geração dos blocos inicialmente ordenados, nota-se, de modo imediato, que o mesmo obteve, na maioria das quantidades registros, o melhor desempenho em todas as métricas nos arquivos cuja situação de ordenação seja crescente. Tal constatação está estritamente em conformidade com o respaldo teórico, que, no caso do arquivo ordenado crescentemente, implica na geração de um único bloco, com todos os itens, e, consequentemente, com uma passada, ordena os registros, materializando o melhor caso para esse método.

Nesse caso, há uma redução significativa nas métricas de comparações, escrita, leitura e tempo de execução, o que resulta em uma evidente economia nos recursos computacionais, caso a entrada corresponda a situação supracitada.

Outro fato evidente corresponde a quantidade de comparações totais em função da quantidade de itens a serem ordenados, em que a quantidade de comparações, com base nos experimentos, foi substancialmente maior proporcionalmente. Alinhado ao discurso teórico, tal fato ancora-se na relação da quantidade de registros lidos no processo de seleção por substituição, que é unitária, e com o algoritmo em si, que, por meio do sistema de marcação, precisa comparar a chave do item recém-lido com o último item a sair toda vez que um novo item é lido, resultando no número de comparações elevado presente nos testes.

Por fim, efetuando uma análise com um enfoque na métrica tempo de execução, nota-se que a situação do arquivo ordenado aleatoriamente implicou em um pior tempo de execução, se comparado com as demais situações. Entretanto, os tempos obtidos, assim como a intercalação 2f com ordenação interna, apresentaram ínfimas variações, entretanto, fazem-se presentes.

4.3 QuickSort externo

Em primeira análise, nas tabelas das análises, todas as linhas relacionadas com geração de blocos ficaram vazias, pois como na política do QuickSort externo não há nenhum tipo de geração de blocos não é realizado nenhuma operação com blocos.

A partir dos dados coletados, podemos inferir algumas conclusões. Como o método de ordenação usado é uma adaptação do QuickSort na memória interna, situações como: "caso médio", "melhor caso"e "pior caso"podem acontecer dependendo do estado inicial do arquivo.

Observando os testes realizados com os arquivos crescentes, podemos inferir que estamos no "melhor caso", logo obtemos um tempo de processamento ótimo e até superior ao resultado da intercalação em alguns casos, Porém quando observamos os arquivos ordenados de maneira decrescente, notamos um resultado extremamente próximo dos resultados com arquivos crescentes sendo que este tipo de estado inicial deveria ser o pior caso, isso ocorre pois o QuickSort externo possuía maneiras de evitar o pior caso gerando sub arquivos com tamanhos parecidos, fazendo com que o processo em cima de um arquivo decrescente seja tão rápido quanto em um crescente.

Observando os testes realizados com arquivos aleatórios podemos ver como este método realmente atua na prática, com arquivos pequenos, a diferença entre o QuickSort e as

intercalações não é notória, entretanto, a partir de um arquivo de 10000 itens o tempo de execução do QuickSort já se torna muito maior com relação às intercalações, chegando a demorar 8 vezes mais na amostra de 471705 itens. As comparações e as transferências também se distanciam muito dos outros métodos de ordenação.

5 Conclusão

5.1 Análise comparativa de desempenho entre os métodos

Comparando as métricas presentes na rodada de teste demandada, nota-se que diferentes métodos apresentaram diferentes desempenhos em conjunturas diferentes. Partindo de uma perspectiva generalista, enuncia-se que tais diferenças se dão pelos algoritmos e lógica empregada na composição desses métodos, em que conjunturas específicas, direta ou indiretamente, beneficiam métodos específicos.

Para quantidades de itens superiores ou iguais a 100000, o método que destacou-se, em todas as métricas comparativas, foi a intercalação 2f com seleção por substituição para geração dos blocos. Tal fato ancora-se na base teórica introduzida, em que, o método de seleção por substituição gera blocos de tamanhos variados e, no caso de arquivos maiores, a probabilidade de gerar blocos com mais itens aumenta consideravelmente, e, novamente, baseando-se no respaldo teórico, blocos maiores implicam em menos blocos, que, por sua vez, implicam em menos passadas necessárias a efetuação da ordenação o que, finalmente, resulta na redução de todas as métricas supracitadas, tornando explícito o quão poderoso esse método pode ser em relação ao que utiliza a ordenação interna para geração dos blocos, por exemplo, que sempre gerará blocos de tamanhos fixos, o que não assegura, ao método, em uma análise assintótica, um poder de otimização no processo de ordenação semelhante ao que utiliza a seleção por otimização.

Por conseguinte, essa questão do método de geração dos blocos inicialmente ordenados, na intercalação de vários caminhos, implica em outras questões relevantes. Dentre elas, ao analisarmos os dados da rodada de experimentos e tendo como perspectiva a situação de ordenação decrescente e a métrica tempo de execução, nota-se que tal configuração constitui o "pior caso"para a intercalação com seleção por substituição, uma vez que os dados dispostos de modo decrescente implica na geração de mais blocos com tamanhos menores, que, por sua vez, torna o processo de ordenação mais lento. Agora, analisando a intercalação com a ordenação interna, sabe-se que gera-se uma quantidade considerável de blocos, todavia todos uniformes, o que, numa vaga análise assintótica, num primeiro momento, levaria a conclusão de que a intercalação com a ordenação interna seria consideravelmente superior. Contudo, ao analisarmos o tempo de execução, a intercalação com seleção por substituição foi melhor em todas as quantidades de itens em tal situação de ordenação, o que viabiliza a inferência de que a estratégia do uso da seleção por substituição para geração dos blocos inicialmente ordenados confere ao método uma preeminência no quesito desempenho, promovendo a inferência de que, dados métodos implementados, a implementação 2f com substituição por seleção é o melhor método para um campo de aplicação cuja natureza dos dados seja ampla, nos quesitos de ordenação e/ou volume, sendo indicado para a maioria dos casos de entrada.

Considerando a quantidade de itens como parâmetro de análise, temos que para as quantidades inferiores a 100000 itens, o QuickSort se destacou no quesito tempo de execução que, na maioria das situações de ordenação do arquivo, sobressaiu-se em relação aos demais métodos. Tal constatação está ancorada no impacto de algumas estratégias adotadas pelo algoritmo, como a tendência em gerar sub arquivos com tamanhos parecidos, diminuindo o número de partições e passadas, otimizando todo processo de ordenação. Entretanto, uma desvantagem evidente presente no método diz respeito ao número de comparações, que é elevadíssimo, em função da construção do algoritmo, que gere dinamicamente um vetor, o pivô, além de, em cada leitura de registro, fazer comparações do item com os limites presentes no algoritmo, resultando no alto número de comparações. Portanto, infere-se que o campo de aplicação ideal para o método corresponde a conjecturas que operam sobre uma quantidade menor de dados, que, levando em consideração de que estamos trabalhando em memória secundária, implica que, na maioria dos casos, trata-se de um elevado volume de dados, o que

leva a conclusão de que o QuickSort externo, dentre os métodos implementados, apresenta o pior desempenho, em todas as métricas, considerando o propósito central da ordenação externa, que é manipular grandes volumes de dados.

Outra constatação promovida pela análise dos resultados obtidos corresponde a algumas características referentes a estabilidade e previsibilidade do método. No que se refere aos quesitos supracitados, o método intercalação 2f com ordenação interna para geração dos blocos inicialmente ordenados destaca-se com relação aos demais métodos, pois, independentemente da situação de ordenação do arquivo, o número de escrita e leitura permanecem constantes para a mesma quantidade de itens a serem ordenados, além da baixa variação no número de comparações. Portanto, infere-se que tal comportamento do método implica que o campo de aplicação ideal faz-se presente em conjecturas ligadas à uma gestão eficiente do hardware para a ordenação, pois, há um bom uso da memória interna, pois o método utiliza toda a capacidade de ordenação. Ademais, os blocos possuem tamanhos fixos, o que promove um agente facilitador no cálculo do número de unidades de armazenamento, viabilizando um uso peremptório do hardware disponível, o que manifesta-se extremamente útil em operações que incidem sobre dados que possuem tamanho considerável, pois, nesses casos, deve-se minimizar o desperdício de hardware e focar na previsibilidade do uso da memória externa disponível. Além disso, comparando o tempo de execução com o intercalação 2f com seleção por substituição, embora o tempo tenha sido superior, a diferença é mínima, e, nos contextos descritos, a junção das características anteriormente citadas com o tempo de execução satisfatório compõem a indicação desse método para tais conjecturas supracitadas.

Por fim, como apresentado na introdução desse item, diferentes métodos são indicados para diferentes conjecturas e situações e, para optar pelo uso de um método em específico, deve-se levar em considerar múltiplas variáveis, como a circunstância do hardware disponível e a natureza dos dados a serem trabalhados, por exemplo. Portanto, cada método possui seu campo de aplicação ideal e, dessa forma, não é cabível o julgamento ou a instituição do "melhor"ou "pior"método de ordenação externa, mediante as inferências e análises feitas.

5.2 Principais dificuldades na implementação dos métodos

Após implementação do código, rodada experimental de testes e análise dos dados obtidos, no tocante às principais dificuldades, enfrentadas pelo grupo, na implementação do trabalho, vale ressaltar que as mesmas fizeram-se presentes em múltiplas etapas do processo do trabalho prático, desde o pré-processamento dos dados originários até a implementação factual dos métodos de ordenação externa.

A priori, a dificuldade inicial deu-se na obtenção dos dados, pois, o fato do arquivo fornecido estar no formato .txt dificultou a leitura das informações, uma vez que os nomes (de cidades, unidades federativas e cursos) apresentam tamanhos variados e, alguns, possuem espaços. Embora os intervalos de cada campo tenham sido devidamente especificados pelo documento orientador do trabalho prático, a leitura, ainda, teve de considerar demais fatores da disposição dos dados, como espaços em branco não preenchidos e etc, dificultando o processo de leitura. De qualquer forma, a leitura inicial do arquivo .txt foi devidamente efetuada e, com base nos dados lidos, gerou-se um arquivo binário estritamente equivalente (ordenado aleatoriamente), arquivo esse que foi utilizado para gerar o arquivo ordenado crescentemente e o de modo decrescente. Vale ressaltar ainda que optamos por manipular arquivos binários em função da simplicidade dos comandos, viabilizando uma implementação menos suscetível a erros de leitura, que poderia resultar em uma leitura equivocada dos dados originais ou em uma lastimável perda de dados.

A posteriori, surgiu e, gradualmente, acentuou-se uma dificuldade correspondente à lógica a ser desenvolvida para a promoção do processo de intercalar blocos, que, se comparada com o restante dos processos dos métodos de ordenação externa, destacou-se exacerbadamente no quesito complexidade de procedimentos e quantidade de etapas necessárias, e, definitivamente, foi o principal empecilho que o grupo teve contato durante o processo de implementação do trabalho. Embora a tarefa de intercalar blocos seja significativamente simples e intuitiva no campo teórico, sua implementação na linguagem C foi amargamente dotada de estorvos e óbices, seja por fatores diretamente ligados a linguagem ou fatores ligados a construção lógica do processo em si. Como fatores associados a linguagem, ressalta-se uma série de problemas ligados

a manipulação de arquivos existentes, em que uma série de alterações no arquivo ficam restritas ao buffer, não sendo efetivamente escritas no arquivo.

Já no que refere-se à construção da lógica da intercalação, a principal dificuldade diz respeito às múltiplas situações que podem ocorrer durante o processo de intercalação e que, para que o processo ocorra de modo correto, devem possuir as devidas tratativas correspondentes. Dentre elas, destacam-se o fato de que um bloco de um fita pode-se esvaziar primeiro que os demais blocos ou que uma ou mais fitas podem ficar inativas. Para solucionar os empecilhos supracitados, adotados a estratégia de composição de estruturas de controle, visando a efetivação de uma supervisão de todas as métricas relacionadas às fitas e aos blocos, como a quantidade de blocos por fitas, a quantidade de itens em cada bloco e o estado de ativação da fita no momento do processo de intercalação.

Ademais, outra dificuldade enfrentada durante a implementação do trabalho foi a complexa manipulação de ponteiros dos arquivos, dada o considerável número de arquivos simulando fitas. Embora os ponteiros tenham sido organizados em uma estrutura que os agrupavam em um vetor, afim de promover uma manipulação simplificada dos mesmos, a etapa de estipular a devida posição do ponteiro do arquivo, afim de assegurar a continuidade no processo de intercalação, demandou uma análise criteriosa da lógica que estava sendo desenvolvida, levando a um árduo processo de revisão e reconstrução da mesma.