UNIVERSIDADE DE SÃO PAULO

Engenharia de Computação

ESTRUTURA DE DADOS III

Profa. Cristina Dutra de Aguiar Ciferri

PAULO DE MOURA - 10310471
PEDRO HENRIQUE RAMPIM NATALI - 10310655
RAFAEL FARIAS PINHO - 10276737
RENAN GABRIEL VARONI - 10310676

São Carlos, 08 de Dezembro de 2018.

ÍNDICE

Introdução e Especificações Sobre a Implementação **3**

Geração de Registros - geracao_de_registros(); **4**

Impressão do Arquivo de Dados - imprimir_arq_bin(); **5**

Ordenação Interna - ordena_arquivo(); **6**

Merging [união] - merging_arquivo(); **6**

Matching [intersecção] - matching_arquivo(); **8**

Multiway Merging[união de muitas listas] - multiway_arquivo(); **10**

Sort-Merge Externo[ordenação de arquivos muito grandes] - sort_arquivo(); **10**

1. Introdução e Especificações Sobre a Implementação

O trabalho proposto na disciplina tem como principal objetivo a manipulação de arquivos por meio de registros. Não será feita uma análise detalhada dos métodos aqui pois cada um deles tem sua própria parte deste relatório. Além disso, a struct utilizada é como deveria ser, sendo assim, nesta seção apresentaremos apenas algumas peculiaridades que podem ser notadas sobre o trabalho.

Foi desenvolvido em Linux 16.04 utilizando o Visual Studio Code como IDE principal, a GCC utilizada foi a versão 5.4.0.

Temos 3 arquivos principais de função, sendo eles :

funcionalidades.c ~ onde as funções se encontram implementadas
funcionalidades.h ~ header file para funcionalidades.c
main.c ~ como o próprio nome diz, é a main do programa.

Para a passagem de parâmetros da main para a funcionalidades.c, foi necessária a utilização de argc/argv por meio da biblioteca stdarg.h, com ela podemos mandar os parâmetros em tempo de execução (junto com o ./).

Para que consiga-se compilar o código, é necessário que se tenha os 4 “txt’s” presentes no arquivo que mandaremos em conjunto com este relatório. Pois eles são o ‘banco de dados’ de onde tiramos nossos registros aleatórios.

Assim, o método de compilação sem o makefile é :

```
gcc funcionalidades.c progtrab.c -o progtrab
```

Por fim, podemos simplesmente ao fazer o download do .zip que será enviado, extrair em algum lugar da máquina. Pelo terminal, entrar na pasta que foi extraída no endereço de sua escolha (usando o comando cd). E assim, digitar **make**. Para apagar os executáveis que são criados de maneira auxiliar, digite **make clean**. E pode-se seguir fazendo os comandos descritos em tópico abaixo.

```
make / make clean
```

2. Geração de Registros - geracao_de_registros();

A primeira funcionalidade tem como objetivo a geração de um arquivo binário contendo n registros formados por 4 campos de tamanho fixo, sendo eles NUSP, nome, livro e data. Estes dados são retiradas de arquivos de texto, são eles: 'nusp'.txt, 'nomes'.txt, 'results_livros'.txt e 'data_de_aluguel'.txt.

A funcionalidade é implementada da seguinte forma:

a) os arquivos .txt são abertos para leitura, e num loop que é realizado n vezes (quantidade de registros desejada pelo usuário do programa) cada campo é lido uma vez e armazenado em um registro. Isso acontece até que o vetor tenha tamanho n, isto é, n registros já foram armazenados, e a primeira parte da função então retorna um vetor de dados do tipo REGISTRO;
b) o vetor de registros é então usado como parâmetro na função `geracao_aleatoria()`, que "bagunça" os registros no vetor para que os campos não fiquem na ordem que estão nos arquivos .txt;
c) em seguida, o vetor de registros é manipulado pela função `repeticoes()`, que vai organizar os registros de forma a respeitar as porcentagens de repetições para cada campo, indicadas na proposta do trabalho. Isso é feito campo a campo, e como exemplo tomaremos o do campo1:
    i) a repetição do campo1 deve ser de 30%, por isso seleciona-se os 70% primeiros registros do vetor, e preenche-se os restantes 30% com valores aleatórios de campo1 recuperados entre os 70%. Assim, caso o vetor tenha tamanho 10, os últimos 3 registros serão preenchidos com valores aleatórios entre os valores de campo1 dos registros 1 a 7.
d) através então da última função, `escrever_arquivo_bin()`, o vetor é percorrido e seus registros são adicionados ao arquivo binário, finalizando a função `geracao_de_registros()` com um arquivo binário de dados contendo n registros.

Todos os campos e variáveis foram tratados a fim de seguir as exigências da proposta do trabalho. Por exemplo, os campos foram preenchidos para não apresentarem espaços vazios, e os dados nos arquivos .txt foram previamente manipulados para apresentarem somente letras maiúsculas e nenhum caractere especial.

./progtrab 1 'arquivo'.bin 'número de registros'

3. Impressão do Arquivo de Dados - imprimir_arq_bin();

A funcionalidade 2 é responsável por imprimir na tela os registros de um determinado arquivo de dados, arquivo este cujo nome é introduzido na função como parâmetro.

A entrada do comando é dada por:

```
./progtrab 2 'arquivo'.bin
```

O código então recupera o nome do arquivo que será impresso e usa-o como parâmetro da função `imprimir_arq_bin(char *nome_arquivo)`. Na implementação da função, o primeiro passo é declarar uma variável auxiliar do tipo REGISTRO, que receberá cada registro do arquivo de dados ao passo que ele é percorrido e os registros são acessados.

O arquivo então é aberto apenas para leitura, usando rb, e como de praxe é feita uma verificação para identificar possíveis problemas na abertura do arquivo.

Em seguida, usa-se a função fread a fim de acessar os dados do arquivo e colocá-los na variável reg, do tipo REGISTRO. Quando realizamos o fread o ponteiro já aponta para o próximo registro, logo usa-se um while para que a função printf seja realizada até que o ponteiro aponte para algo vazio, isto é, quando os dados no arquivo chegarem ao fim. Quando isto acontecer o retorno de fread será 0, e a condição do while não mais será verdadeira (0 = 0), o que interrompe o loop.

Dessa forma, todos os registros foram impressos na tela, e por fim o arquivo é fechado.

PSEUDOCÓDIGO:

```
abre arquivo 'a' para leitura
lê registro de a
enquanto (arquivo não acabar) faça
     imprime registro de a
     lê registro de a
fim enquanto
fecha 'a'
```

4. Ordenação Interna - ordena_arquivo();

A ordenação interna tem por seu fim ordenar arquivos de forma rápida quando possível, recebendo um arquivo de entrada e outro de saída, ordenando a partir do primeiro campo, ou seja, usando-o como primeira chave; em caso de empate entre dois elementos, passa-se para a análise do próximo campo e assim sucessivamente até quarto campo como desempate. Como diferencial da Ordenação Interna, temos que ela ocorre completamente em RAM, de forma a torná-la rápida, porém, necessidade de um bom espaço para ser realizada, sendo útil para arquivos menores ou comutadores com grande capacidade de uso de memória volátil, e não recomendadas para arquivos que possam ser grandes demais para ordenar em RAM.

Busca-se o comando por:

./progtrab 3 ‘arquivo’.bin ‘arq_ordenado’.bin

O algoritmo se inicia declarando uma variável do tipo FILE* para manipular a abertura e manipulação do arquivo, com isso, abre o arquivo1, colocando-o em RAM para iniciar a ordenação, sendo colocado para modo de leitura apenas (aberto como rb), o outro arquivo é aberto de modo a se permitir leitura e escrita (aberto com wb+), sendo iniciado como um arquivo em branco. Os arquivos são chamados de arq_sem_ord (arquivo de entrada) e arq_com_ord.

Após a ordenação do vetor por meio do algoritmo de shellsort, é feito um fwrite de cada instância do vetor no arquivo de saída (ordenado). Por fim os arquivos que foram abertos no início do algoritmo são agora fechados.

5. *Merging* [união] - merging_arquivo();

A funcionalidade de *Merging* é responsável por manipular dois arquivos ordenados e produzir um único arquivo como saída, que una os dois arquivos de entrada e esteja ordenado levando em conta o campo1 como primeira chave. Isso quer dizer que dois registros podem possuir campo1 de valor igual, o que nos leva a analisar o campo2, depois o campo3 e por fim o campo4.

De forma resumida, a ordenação é feita do mesmo jeito que na funcionalidade anterior, assumindo-se o próximo campo como “desempate” quando dois registros tiverem campos de mesmo valor. A partir disso, produz-se um arquivo de saída

ordenado e que contém todos os registros dos dois arquivos de entrada. O comando da funcionalidade é dado por:

./progtrab 4 ‘arq1’.bin ‘arq2’.bin ‘arq_saida’.bin

Mais uma vez, antes de se iniciar o algoritmo, algumas variáveis auxiliares são declaradas, bem como os arquivos são inicializados (arquivo_1 e arquivo_2 com rb, pois serão apenas lidos, e arquivo_saida com wb+, o que indica que um arquivo em branco foi criado e aberto tanto para escrita quanto para leitura).

De forma geral, o algoritmo funciona da seguinte forma: enquanto não terminam os registros de ambos os arquivos de entrada, eles são percorridos um a um e seus registros são comparados. Cada vez que um registro é comparado, é conferido se algum dos dois arquivos está no fim, se for este o caso (arquivo A no fim) o outro arquivo (arquivo B) é percorrido de forma independente a partir dali, e tem todos os seus registros escritos no arquivo de saída (já que o arquivo A já foi completamente percorrido e teve todos os registros escritos). Além disso, a cada comparação pode acontecer dos registros possuírem chaves iguais, então a chave torna-se o próximo campo (como já explicado anteriormente); caso chega-se na última chave e os campos de ambos os registros ainda são iguais, apenas um deles é escrito (visto que os registros são exatamente iguais).

Simplificando bastante, o algoritmo pode ser resumido no seguinte PSEUDOCÓDIGO, onde reg1 e reg2 são os atuais registros dos arquivos 1 e 2, respectivamente:

```
abre arq1 e arq2 para leitura
abre arq.saida para escrita
enquanto (arq1 e arq2 não terminam)
  compara-se reg1 e reg2 pelo campo1
  caso 1: reg1<reg2 e reg1 ñ é último de arq1
    escreve reg1 em arq.saida
    reg1 recebe proximo registro
  caso 2: reg1<reg2 e reg1 é o último de arq1
    escreve reg1 em arq.saida
    escreve arq2 em arq.saida
    encerra
  caso 3: reg1=reg2
    compara-se reg1 e reg2 pelo campo2
    caso 1: reg1<reg2 e reg1 ñ é último de arq1
      escreve reg1 em arq.saida
      reg1 recebe proximo registro
    caso 2: reg1<reg2 e reg1 é o último de arq1
```

```
            escreve reg1 em arq.saida
            escreve arq2 em arq.saida
            encerra
        caso 3: reg1=reg2
            compara-se reg1 e reg2 pelo campo3
            caso 1: reg1<reg2 e reg1 ñ é último de arq1
               escreve reg1 em arq.saida
               reg1 recebe proximo registro
            caso 2: reg1<reg2 e reg1 é o último de arq1
               escreve reg1 em arq.saida
               escreve arq2 em arq.saida
               encerra
            caso 3: reg1=reg2
                  compara-se reg1 e reg2 pelo campo4
                  caso 1: reg1<reg2 e reg1 ñ é último de arq1
                     escreve reg1 em arq.saida
                     reg1 recebe proximo registro
                  caso 2: reg1<reg2 e reg1 é o último de arq1
                     escreve reg1 em arq.saida
                     escreve arq2 em arq.saida
                     encerra
                  caso 3: reg1=reg2
                     escreve reg1 e reg2 em arq.saida
                     reg1 e reg2 recebem próximo registro
fim enquanto //arq1 e arq2 completamente escritos no
             //arquivo de saída
finaliza //fecha os três arquivos
```

É importante chamar a atenção para os comandos “encerra” e para as linhas que contém “recebe próximo registro”. Caso o algoritmo seja encerrado antes de chegar ao fim, isto é, passa por “encerra”, quer dizer que ambos arquivos foram completamente escritos no arquivo de saída, então não há necessidade de continuar a execução. Por outro lado, é importante ressaltar que cada vez que um “reg” recebe o próximo registro de um arquivo, a execução retorna para a linha em que é conferido se ambos os arquivos não acabaram, e a partir daí ocorre a comparação desde a primeira chave (campo1: nusp).

O algoritmo então implementa com sucesso os três passos principais de uma operação cosequencial, como o *merging*: inicialização dos arquivos, sincronização dos registros e gerenciamento da condição de fim.

6. *Matching* [intersecção] - matching_arquivo();

A função de matching é documentada no código do trabalho como `matching_arquivo(argv[ARG1], argv[ARG2], argv[ARG3])`, isto é, tem como parâmetros os arquivos de dados que serão analisados para a intersecção (argv[ARG1] e argv[ARG2]), e o arquivo de saída, onde serão escritos os dados analisados (argv[ARG3]).

A entrada do comando, assim como nas outras funcionalidades, é do tipo:

./progtrab 5 ‘arq1’.bin ‘arq2’.bin ‘arq_saida’.bin

Analisaremos agora a estrutura da funcionalidade.

O primeiro passo, também padrão para a maioria das funcionalidades, é a criação de variáveis auxiliares do tipo REGISTRO, no caso reg1 e reg2, que serão usadas na comparação entre os registros dos arquivos de entrada.

Depois disso, temos a inicialização dos três arquivos (dois de entrada e um de saída), e eles são inicializados de acordo com suas funções (os de entrada são inicializados apenas para leitura, logo rb, e o de saída apenas para escrita, logo wb).

Em seguida, dá se início às comparações entre os dados, levando-se em conta o campo1. Isso quer dizer que dois arquivos serão comparados registro a registro, e quando dois registros de arquivos diferentes possuírem o campo1 com mesmo valor, ambos serão escritos no arquivo de saída e os próximos registros de ambos os arquivos serão acessados. No caso de um registro ser menor que o outro (quando não ocorre o *match*) o algoritmo avança no arquivo com registro de menor chave, isso porque os arquivos de entrada estão ordenados, então se um é menor que o outro sabe-se que os possíveis *matches* encontram-se afrente do registro menor.

A implementação é então explicada pelo seguinte PSEUDOCÓDIGO:

```
abre arq1 e arq2 para leitura
abre arq.saida para escrita
enquanto (arq1 e arq2 não estiverem no final)
     se reg1<reg2 // comparação de registros com campo1
          reg1 recebe proximo registro de arq1
     senao se reg1>reg2
          reg2 recebe proximo registro de arq2
     senao        // reg1 = reg2 em relacao a campo1
          escreve reg1 em arq.saida
          escreve reg2 em arq.saida

fim enquanto
finaliza          // fecha os arquivos
```

O *matchin*g, assim como o *merging*, é uma operação cosequencial. Isto quer dizer que a entrada são duas listas já ordenadas, e assim como na funcionalidade anterior o algoritmo abrange a inicialização de cada um dos arquivos, bem como a sincronização dos registros e o gerenciamento da condição de parada.

7. *Multiway Merging[união de muitas listas] - multiway_arquivo();*

A funcionalidade [6] tem como objetivo realizar a operação de *merging* em mais de 2 arquivos ordenados, o que a diferencia da funcionalidade [4]. Aqui, teremos como entrada n arquivos ordenados, e como saída um arq.saida também ordenado, que contém todos os registros dos n registros passados como parâmetro.

O código é implementado usando a estrutura árvore de seleção, e ela é preenchida comparando-se dois a dois entre registros dos arquivos de entrada, sendo o menor valor "elegido" a pai, e este será comparado com outro registro, quando o menor entre eles irá se tornar o próximo pai. Dessa forma, a raíz da árvore será preenchida pelo registro de menor valor.

Para cada arquivo de dados é criado um vetor de registros. Os nós da árvore de seleção foram implementados por um *struct* denominada *node*, que recebe o índice do vetor de registro e o valor do primeiro campo do registro em questão. Os nodes são colocados na árvore e, como dito acima, o menor registro é alocado como raiz, em seguida sendo escrito no arquivo de saída. Quando isso é feito o índice do node é aumentado em 1, ou seja, aquela node agora contém o próximo registro do arquivo. Isso é feito até todos os registros serem escritos no arquivo de saída.

A entrada do comando é dada por:

./progtrab 6 'arq1.bin' 'arq2.bin' …. 'arq(n)'.bin 'arquivo_resultante'.bin

8. *Sort-Merge Externo[ordenação de arquivos muito grandes] - sort_arquivo();*

Nessa funcionalidade, pegamos um arquivo de registros e separamos em subarquivos que caibam nas páginas de disco, que no caso comportam um total de 1000 registros, separados os arquivos aplicamos o método de ordenação para termos vários arquivos ordenados que podem ser juntos novamente pelo multiway merging. No entanto somente 4 arquivos podem ser juntos de uma vez por causa do tamanho do buffer.

Não conseguimos executar esta função. Ainda assim, colocamos o raciocínio dela no código para que seja avaliado. Aqui passamos como seria chamada caso funcionasse:

./progtrab 7 'arq1'.bin 'arq_resultante'.bin