Trabalho 7 – Aplicação de ordenação e busca em arquivos

Algoritmos de busca e ordenação são muito utilizados no gerenciamento de arquivos. Para exercitar o uso desses algoritmos e a prática em programação, nesse trabalho serão realizadas operações com arquivos com o fim principal de realizar buscas. Para isso será desenvolvido um programa para gerenciar arquivos, índices e realizar buscas.

Leia a descrição do trabalho **com atenção e várias vezes**, anotando os pontos principais e as possíveis formas de resolver o problema. Comece a trabalhar o quanto antes para que as dúvidas sejam sanadas a tempo de você conseguir finalizar o trabalho a tempo.

Nesse trabalho você deverá desenvolver um programa que leia um arquivo de script e que realize as operações descritas nesse arquivo. As operações possíveis incluem a criação novos arquivos de dados, criação de índices para acesso a esse arquivos, buscas utilizando índices e sem a utilização de índices. Além disso funções auxiliares podem ser executadas para obter como saída informações sobre os arquivos e buscas realizadas. Veja a seção a seguir para detalhes.

Tarefa

Este trabalho é uma extensão do Trabalho 5. Considere que existe um arquivo de script. Seu programa lê o nome desse arquivo usando scanf e, em seguida, começa a processá-lo.

Como exemplo do conteúdo desse arquivo, considere:

```
create table pessoa (codigo int, nome char[80], idade int, sexo char );
create table empresa (razaosocial char[255], endereco char[255], numero int, cidade char[80], estado char[2]);

showalltables

insert into pessoa (codigo, nome, idade, sexo) values (1, 'joao da silva', 10, 'M');
insert into pessoa (codigo, nome, idade, sexo) values (2, 'paula souza', 11, 'F');
insert into pessoa (codigo, nome, idade, sexo) values (3, 'jose ricardo martins', 90, 'M');

insert into empresa (razaosocial, endereco, numero, cidade, estado) values ('Usp', 'Av Trabalhador Saocarlense', 400, 'Sao Carlos', 'SP');
insert into empresa (razaosocial, endereco, numero, cidade, estado) values ('Empresa A', 'Av Sao Carlos', 30, 'Sao Carlos', 'SP');
insert into empresa (razaosocial, endereco, numero, cidade, estado) values ('Empresa B', 'Av XV', 132, 'Araraquara', 'SP');

create index pessoa(codigo);
create index pessoa(nome);

create index empresa(razaosocial);
create index empresa(cidade);

showallindexes

statistics

insert into pessoa (codigo, nome, idade, sexo) values (4, 'carlos almeida', 123, 'M');
insert into pessoa (codigo, nome, idade, sexo) values (2, 'joaquim jose', 123, 'M');

select pessoa codigo '1';
```

```
statistics

insert into pessoa (codigo, nome, idade, sexo) values (15, 'josefina say', 72, 'F');

sort pessoa(codigo);
sort pessoa(nome);
```

Esse arquivo começa com as seguintes instruções:

```
create table pessoa (codigo int, nome char[80], idade int, sexo char );
create table empresa (razaosocial char[255], endereco char[255], numero int, cidade char[80], estado char[2]);
```

Nessas duas instruções acima são definidas duas tabelas. A primeira tabela é chamada pessoa e a segunda, empresa. Observe que cada tabela contém campos, cada um com um tipo diferente. Neste trabalho os tipos possíveis serão: char, char[tamanho], int, float e double.

Caso seja char, o campo tem apenas 1 caracter. Caso char[tamanho] o campo tem tamanho caracteres, por exemplo: char[80] tem 80 caracteres. O tamanho de um float é dado por sizeof(float), o mesmo para double e int.

Após ler essas duas primeiras instruções, crie uma estrutura de dados que armazene informações sobre essas tabelas. Essa estrutura de dados DEVE SER DINAMICAMENTE ALOCADA.

Em seguida temos a instrução:

```
showalltables
```

Nesse caso, apresentamos as informações sobre as tabelas usando o printf abaixo:

```
printf("\nTablename: %s\n", tablename);

        for (i = 0; i < nfields; i++) {

                printf("\tField: %s Type: %s Size %d\n",
                                fieldname[i],
                                fieldtype[i],
                                fieldsize[i]);
        }

        printf("\n");
```

Observe que esse printf já traz informações sobre como a saída deve ser produzida, ou seja, onde haverá pulos de linha, tabs, espaços, etc.

Em seguida, surgem três instruções:

```
insert into pessoa (codigo, nome, idade, sexo) values (1, 'joao da silva', 10, 'M');
insert into pessoa (codigo, nome, idade, sexo) values (2, 'paula souza', 11, 'F');
insert into pessoa (codigo, nome, idade, sexo) values (3, 'jose ricardo martins', 90, 'M');
```

Essas instruções são responsáveis por inserir dados de pessoas em um arquivo temporário. O

nome do arquivo temporário é dado pelo nome da tabela + a extensão ".tmp", sendo assim, para a tabela pessoa o nome do arquivo temporário será pessoa.tmp. O mesmo ocorre para as instruções abaixo, no entanto os dados serão inseridos no arquivo temporário empresa.tmp

```
insert into empresa (razaosocial, endereco, numero, cidade, estado) values ('Usp', 'Av Trabalhador Saocarlense', 400, 'Sao Carlos', 'SP');
insert into empresa (razaosocial, endereco, numero, cidade, estado) values ('Empresa A', 'Av Sao Carlos', 30, 'Sao Carlos', 'SP');
insert into empresa (razaosocial, endereco, numero, cidade, estado) values ('Empresa B', 'Av XV', 132, 'Araraquara', 'SP');
```

As instruções a seguir são responsáveis por criar índices para busca binária. A primeira delas criará um arquivo de índice chamado `pessoa-codigo.idx`, a segunda instrução criará o arquivo `pessoa-nome.idx` e assim sucessivamente.

Antes de criar um arquivo de índice, você deve copiar todos os dados do arquivo temporário relativo à aquele índice em um arquivo de dados cujo nome é dado pelo nome da tabela + a extensão "`.dat`". Para a tabela pessoa, haverá então um arquivo de dados chamado pessoa.dat e outro temporário chamado pessoa.tmp.

Antes de um índice ser criado, copiamos todos os dados contidos no arquivo temporário para dentro do arquivo de dados e depois apagamos o temporário. Dessa maneira, o temporário tem as últimas inserções anteriores à criação do arquivo de índice.

Para a primeira instrução, o nome do arquivo será pessoa-codigo.idx e dentro desse arquivo haverá uma chave (código) associada a um offset (posição do registro no arquivo de dados).

```
create index pessoa(codigo);
create index pessoa(nome);

create index empresa(razaosocial);
create index empresa(cidade);
```

Após criar o arquivo de índice, deve-se ordená-lo tal como proposto no Trabalho 5. A diferença é que agora você DEVE USAR UM ALGORITMO ESTÁVEL para ordenação.

A instrução a seguir apresenta dados sobre todos os índices já criados:

```
showallindexes
```

No formato:

```
for (i = 0; i < totalDeIndices; i++) { // deve estar na ordem de criação dos índices
    printf("\nIndex information\n");
    printf("\tTablename: %s\n", tablename[i]); // nome da tabela relacionada ao indice i
    printf("\tFieldname: %s\n\n", fieldname[i]); // nome do campo indexado
}
```

A instrução a seguir apresenta estatísticas no formato abaixo (observe que as instruções statistics, showalltables e outras podem não ter ponto e vírgula no final, ou seja, alguém pode ou não colocar ponto e vírgula no final):

```
statistics
```

Formato:

```
printf("#Tables: %d\n", ntables); // número de tabelas criadas até então
printf("#Indexes: %d\n", nindexes); // número de índices criados até então
printf("#Inserts: %d\n", ninserts); // número de inserts feitos até então
printf("#Selects: %d\n", nselects); // número de selects feitos até então
printf("#Sorts: %d\n", nsorts); // número de sorts feitos até então
printf("#ShowAllTables: %d\n", nshowalltables); // número de showalltables feitos até então
printf("#ShowAllIndexes: %d\n", nshowallindexes); // núm. de showallindexes feitos até então

printf("#Records in last select (binary search): %d\n", recordsInLastSelectBinary);
// num. de registros recuperados via busca binária no último select feito

printf("#Records in last select (sequential search): %d\n", recordsInLastSelectSequential);
                                // número de registros recuperados via busca sequencial no
                                // último select feito
```

A instrução a seguir realiza uma busca binária usando o arquivo de índice `pessoa-codigo.idx` e, em seguida, continua com uma busca sequencial no arquivo temporário (pois pode haver conteúdo nesse arquivo). Esse select deve apresentar na tela todos os registros cujo código é igual a 1, segundo sua ordem de inserção no arquivo temporário, ou seja, deve-se garantir estabilidade na ordenação. Se um registro A passou do arquivo temporário para o de dados antes de outro B e ambos apresentam código = 1, então deve-se imprimir os dados do registro A, antes dos dados do registro B.

Observe que também é necessário contar quantos registros foram retornados via busca binária e quantos vis busca sequencial, conforme a instrução (statistics) anteriormente descrita.

```
select pessoa codigo '1';
```

Os dados a serem impressos devem ter o seguinte formato:

1) char ou char[tamanho] deve ser impresso entre aspas simples na forma:

```
printf(“'%s'”, valor);
```

2) Os campos int, float ou double são impressos apenas na forma de seus valores:

```
printf(“%d”, valor_inteiro);
printf(“%f”, valor_float);
printf(“%lf”, valor_double);
```

A impressão final deve ficar da seguinte maneira:

```
1, 'joao da silva', 10, 'M'
```

Suponha que a instrução select tenha sido outra e alguns inserts tenham ocorrido antes:

```
insert into pessoa (codigo, nome, idade, sexo) values (10, 'paula souza', 16, 'F');
insert into pessoa (codigo, nome, idade, sexo) values (11, 'paula souza', 25, 'F');
select pessoa nome 'paula souza';
```

Nesse caso, o resultado será:

```
2, 'paula souza', 11, 'F'
```

```
10, 'paula souza', 16, 'F'
11, 'paula souza', 25, 'F'
```

Observe que a cada registro é apresentado em uma linha. Não se esqueça de pular uma linha após apresentar cada registro, inclusive o último.

Há ainda uma instrução chamada sort. Essa instrução regera um certo índice. Regerar significa apagar o arquivo de índice e produzir um novo com base no arquivo de dados. Se houver qualquer registro no arquivo temporário, deve-se transferí-lo para o arquivo de dados e, em seguida, regerar o índice. Abaixo são apresentados dois comandos sort, um que regera o índice código da tabela pessoa e outro que regera o arquivo de índices relacionado ao campo nome da tabela pessoa.

```
insert into pessoa (codigo, nome, idade, sexo) values (15, 'josefina say', 72, 'F');

sort pessoa(codigo);
sort pessoa(nome);
```

Informações importantes

- Sobre a avaliação

1. Um dos objetivos da disciplina de ICC2 é o **aprendizado individual** dos conceitos de programação. A principal evidência desse aprendizado está nos trabalhos, que são individuais neste curso. Você deverá desenvolver seu trabalho sem copiar trechos de código de outros alunos, nem codificar em conjunto. Portanto, compartilhem idéias, soluções, modos de resolver o problema, mas não o código.
    - O plágio vai contra o código de ética da USP.
    - Quando autores e copiadores combinam, estão ludibriando o sistema de avaliação.
    - O trabalho em grupo e a cooperação entre colegas é em geral benéfico e útil ao aprendizado. Para ajudar um colega você pode lhe explicar estratégias e idéiar. Por exemplo, pode explicar que é preciso usar dois loops para processar os dados, ou que para poupar memória basta usar uma certa estrutura de dados, etc. O que você **não** deve fazer é mostrar o seu código. Mostrar/compartilhar o código pode prejudicar o aprendizado do seu colega:
        - depois de o seu colega ter visto o seu código, será muito mais difícil para ele imaginar uma solução original e própria;
        - o seu colega não entenderá realmente o problema: a compreensão passa pela prática da codificação e não pela imitação/cópia.
    - Um colega que tenha visto a sua solução pode eventualmente divulgá-la a outros colegas, deixando você numa situação muito complicada, por tabela.
    - O texto acima foi baseado e adaptado da página http://www.ime.usp.br/~mac2166/plagio/, da qual recomendo a leitura completa.

2. Todos os códigos fontes serão comparados por um (ou mais) sistema(s) de detecção de plágio, e **os trabalhos com alta similaridade detectada terão suas notas zeradas**, tanto aqueles relativos ao código de origem quanto do código copiado.

3. A avaliação incluirá a porcentagem de acertos verificada pelo SQTPM e também a análise do seu código, incluindo endentação, comentários, bom uso da memória e práticas de programação. Portanto faça seu código com cuidado, da melhor forma possível.

- Sobre o sistema de submissão (SQTPM):

1. Seu código deverá ser submetido num arquivo fonte .c. Esse arquivo deverá **obrigatoriamente** conter no início um comentário com seu nome, número USP, turma e data da entrega do trabalho.

2. A compilação do código é feita pelo comando:
```
gcc -o prog proc.c -lm -Wall
```

3. A saída do seu programa deve ser exatamente igual à saída esperada, incluindo: espaços em branco, quebras de linha e precisão decimal.

4. Há um limite de 20 segundos para a execução dos casos de teste e um limite de 16.5MB de memória total para ser utilizada. Você deverá gerenciar bem o tempo de execução e o espaço ocupado para que seu programa fique dentro desses limites, para evitar uso excessivo, pois o sistema irá invalidar o caso em que o limite foi excedido.

5. Ao enviar, aguarde um pouco mais de 30 segundos, **sem recarregar a página nem pressionar ESC**, para obter a resposta. Caso demore mais do que 1 minuto para dar uma resposta, feche e abra novamente a página do servidor. Verifique se seu programa tem algum problema de entrada de dados (não está lendo todos os dados), se tem algum loop infinito ou parada para pressionamento de tecla (getch() getchar() system("PAUSE")). Caso não tenha, aguarde alguns minutos e tente novamente.

6. O erro de "Violação de Memória" significa acesso indevido a arranjo ou arquivo. Use compilação com **-g** e o programa valgrind para tentar detectar a linha de código com erro.
    1. Exemplo 1 de violação de memória:
```
char *array = (char *)malloc(sizeof(char) * N);
scanf("%s", &filename); // acesso indevido, violacao de memoria
free(array);
```

    2. Exemplo 2 de violação de memória:
```
int **mat = (int **)malloc(sizeof(int *) * 3);
mat[0] = (int *)malloc(sizeof(int)*10);
for (i = 1; i < 3; i++) {
     // apenas a posicao 0 de mat foi alocada as outras nao
     // portanto esta liberando regiao nao alocada
     // gerando violacao de memoria
     free(mat[i]);
 }
```

    3. Exemplo 3 de violação de memória:
```
int A[N];
int B[N];
int j = 0, i = 5;
while (j < N){
```

```
      A[i] = B[j]; // como j e i sao indices diferentes
      j++;         // quando j = (N-1) i = (N-1)+5 e
      i++;         // havera escrita indevida em A
}
```