Exercício

Algoritmos de busca e ordenação são muito utilizados no gerenciamento de arquivos. Para exercitar o uso desses algoritmos e a prática em programação, neste exercício 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.

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 de 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

Considere que existe um arquivo de script. Seu programa lê o nome desse arquivo e, em seguida, começa a processá-lo.

Como exemplo do conteúdo desse arquivo de script, 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 (pessoa.dat e empresa.dat são seus arquivos correspondentes). A primeira tabela é chamada pessoa e a segunda, empresa. Observe que cada tabela contém campos, cada um com um tipo diferente. Neste exercício 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. Por exemplo, considere o arquivo de dados pessoa.dat contém os dados a seguir (apresentado em formato texto para simplificar a compreensão, no entanto, este arquivo de dados deve ser binário, com cada tipo sendo gravado em sequência, um após o outro):

9, “Ana Maria”, 26, ‘F’
10, “Joao”, 25, ‘M’
8, “Paulo”, 35, ‘M’
1, “Helena”, 22, ‘F’

Lembramos que a tabela pessoa foi criada como definido a seguir:

```
create table pessoa (codigo int, nome char[80], idade int, sexo char );
```

Logo cada um de seus registros terá 4 bytes devido ao tipo int, 80 de char, 4 bytes para int idade e 1 byte para sexo, totalizando 89 bytes.

Logo o arquivo pessoa-codigo.idx (também binário) a ser gerado deverá conter, inicialmente, os seguintes dados (o ideal é armazenar em formato binário, mas aqui apresentamos como texto para melhor visualização):

```
9 0
10 89
8 178
1 267
```

O valor 9 é a chave ou código do primeiro usuário. O valor 10 é o código do segundo usuário no arquivo de dados e assim por diante. O valor 0 na primeira linha é o deslocamento (também chamado de offset ou posição) do registro do primeiro usuário no arquivo de dados. O valor 89 é a posição do segundo registro no arquivo de dados (já que o tamanho total de um registro é de 89 bytes), e assim por diante.

Após gerar o arquivo idx, orden-o de maneira crescente segundo sua chave (neste caso a chave é definida pelo campo código), obtendo um arquivo final pessoa-codigo.idx na forma:

```
1 267
8 178
9 0
10 89
```

Observe que agora o código 8 aparece antes do 10. Observe, também que a posição do registro com código 8 acompanha essa ordenação, ou seja, o registro de código 8 está na posição 178.

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:

```
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. 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 via 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 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);
```