Trabalho 5 – Ordenação e busca de dados

1) Descrição

Nesse trabalho iremos implementar um programa que seja capaz de criar arquivos de dados ordenados e realizar buscas usando esses arquivos. O objetivo é ganhar experiência em programação e aplicar os algoritmos de ordenação e busca vistos em aula. Esse trabalho não requer muitos novos conhecimentos, porém terá muitas linhas de código, portanto inicie o quanto antes!

1.1) Arquivos schema (esquema) e dat (dados)

Considere um arquivo cuja extensão é `schema`, por exemplo: `usuarios.schema`.

Esse arquivo textual, também denominado "arquivo de schema", contém o formato de um arquivo de dados disponível no sistema. Considere o exemplo abaixo para o arquivo `usuarios.schema`:

```
tabela usuarios
     codigo int order
     nome char[80] order
     idade int
     endereco char[80]
     numero char[10]
     cidade char[50]
     estado char[2]
```

As linhas desse arquivo de schema definem:

1) a primeira linha, ou seja, `tabela usuarios.dat`, define que o nome do arquivo de dados é usuarios.dat. Observe que todo arquivo de dados terá, portanto, extensão .dat adicionada ao nome da tabela presente neste arquivo de schema.
2) a segunda linha em diante apresenta os dados que compõem um registro ou struct de usuário, contendo:
   I. o identificador do dado
   II. o tipo do dado (**DEVE-SE SUPORTAR SOMENTE int E char[*], EM QUE * IDENTIFICA QUALQUER NÚMERO INTEIRO**)
   III. nenhuma ou uma palavra-chave que adiciona característica especial ao dado

No exemplo (**QUALQUER OUTRO SCHEMA DEVE SER SUPORTADO NO TRABALHO, I.E., PODE-SE TER MAIS OU MENOS CAMPOS EM ORDENS DISTINTAS**) `usuarios.schema` acima, a segunda linha define que o arquivo de dados armazena um código, do tipo inteiro para cada um dos usuários. A palavra especial **order** significa que deve existir um arquivo com o mesmo nome do arquivo de dados adicionado do nome daquela variável, por exemplo, neste caso deve-se criar um arquivo chamado `usuarios-codigo.idx`, o qual deve representar (ver detalhes desse arquivo explicado mais abaixo) uma ordenação de todos os registros de usuários cadastrados em `usuarios.dat` usando como chave de ordenação o campo código.

O arquivo de dados contém diversos registros. Cada registro segue a definição dada no arquivo

de schema anteriormente exemplificado. Por exemplo, considere o arquivo de dados `usuarios.dat` 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 Rua Brigadeiro Faria Lima 3456 São Paulo SP
10 Joao 25 Av. São Carlos 123 São Carlos SP
8 Paulo 35 Rua São Sebastião 456 São Paulo SP
1 Helena 22 Alameda Santos 2156 São Paulo SP
```

1.2) Arquivo idx (índice)

Considere agora o arquivo `usuarios-codigo.idx`. Esse arquivo deverá conter todas as chaves (**RETIREI DAQUI A PALAVRA ÚNICAS. O ARQUIVO DE ÍNDICE DEVE, DE FATO, CONTER TODAS AS CHAVES**) do arquivo de dados mais o offset (ou posição) do registro no arquivo de dados original.

Observe que nesse exemplo há quatro registros para usuários. Logo o arquivo `usuarios-codigo.idx` (também binário) a ser gerado deverá conter, inicialmente, os seguintes dados, considerando a implementação numa máquina com processador 32 bits:

```
9 0
10 230
8 460
1 690
```

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 230 é a posição do segundo registro no arquivo de dados (já que o tamanho total de um registro é de 230 bytes), e assim por diante.

Após gerar o arquivo idx para cada uma das variáveis definidas no arquivo de schema (ou seja, pode ser que o arquivo de schema obrigue a geração de mais de um arquivo de ordenação), ordene o arquivo idx de maneira crescente segundo sua chave, obtendo um arquivo final `usuarios-codigo.idx` na forma:

```
1 690
8 460
9 0
10 230
```

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 460.

OBS: Pode-se realizar ordenação em memória.
IMPORTANTE: Deve-se ordenar segundo os campos definidos no arquivo textual de schema antes de iniciar a execução das instruções armazenadas no arquivo de script.

1.3) Arquivo script

Considere agora um arquivo denominado arquivo de script no formato texto (assim como o arquivo de schema). Por exemplo, considere que o nome desse arquivo é `usuarios.script`. Esse arquivo contém instruções para:

- manipular buscas no arquivo de dados usando para isso o arquivo idx, ou seja, o arquivo de ordenação: comando **search**.

Considere o arquivo `usuarios1.script` abaixo:

```
search usuario-codigo.idx "8" nome
search usuario-codigo.idx "10" cidade
```

Esse arquivo pode ter quantas linhas quisermos. As linhas do exemplo acima significam:

1) Primeira linha – usando o arquivo de ordenação usuarios-codigo.idx, busque pelo código 8 e depois vá até o arquivo de dados, na posição indicada (posição 460 neste caso) e carregue o registro correspondente apresentando apenas o nome do usuário, caso encontrado;

2) Segunda linha – usando o arquivo de ordenação usuarios-codigo.idx, busque pelo código 10 e depois vá até o arquivo de dados, na posição indicada (posição 230 neste caso) e carregue o registro correspondente apresentando apenas a cidade do usuário;

Caso o arquivo .idx indicado não exista, a busca deverá retornar sem sucesso.

Considere um outro arquivo `usuarios2.script` abaixo:

```
search usuario-nome.idx "Ana Maria" estado
search usuario-nome.idx "Helena" idade
```

As linhas desse exemplo significam:

1) Primeira linha – usando o arquivo de ordenação usuarios-nome.idx, busque pelo nome "Ana Maria" e depois vá até o arquivo de dados, na posição indicada (posição 0 neste caso) e carregue o registro correspondente, retornando apenas o estado do usuário, caso encontrado;

2) Segunda linha – usando o arquivo de ordenação usuarios-nome.idx, busque pelo nome "Helena" e depois vá até o arquivo de dados, na posição indicada (posição 690 neste caso) e carregue o registro correspondente, retornando apenas a idade do usuário, caso encontrado;

Repare que, nesse caso, o campo nome não é único. Por simplificação, você deverá retornar o dado do primeiro registro que encontrar por meio da busca binária.

Tarefa

Considere que já existam:

1. um arquivo .dat com os dados
2. um arquivo .schema com a definição dos dados
3. arquivos .script com comandos a serem executados

A entrada para seu programa será o nome do arquivo de schema e o nome do arquivo de script, por exemplo:

```
usuarios.schema
usuarios.script
```

A saída dependerá de cada linha no arquivo de script. Para cada busca (search), a saída deverá ser o resultado da busca adicionado do número de iterações necessárias para realizar a busca binária no arquivo de ordenação. No formato:

```
saida-solicitada_numero-de-iteracoes\n
```

Em que saida-solicitada pode ser qualquer variável que compõem o registro de um usuário.

Exemplo, caso o resultado seja o nome da cidade de um usuário e o programa tenha levado 5 iterações para encontrar o registro:

```
"São Paulo"_5\n
```

Por exemplo, caso o resultado seja a idade (23 anos) de um usuário e o programa tenha levado 8 iterações para encontrar o registro:

```
"23"_8\n
```

Caso seja solicitada uma busca num arquivo de índice inexistente, ou para um campo inexistente no esquema (tal como CPF 100 no exemplo do arquivo usuarios.dat), você deve produzir a saída:

```
NULL_0\n
```

Caso seja solicitada uma busca para um usuário inexistente, tal como busca por codigo 999 seguindo o mesmo exemplo, você deve produzir a saída:

```
NULL_numero-de-iteracoes\n
```

IMPORTANTE: A busca binária deve ser realizada diretamente sobre o arquivo de índice idx, ou seja, não será aceito trabalho que carregar todos os dados do arquivo de índice em memória para, posteriomente, proceder com a busca binária.

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 90 segundos para a execução dos casos de teste e um limite de 18.4MB 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. O erro de "Violação de Memória" significa que seu programa está usando mais do que 18.4MB. Esse erro também pode significar vazamento de memória, acesso indevido a arranjo ou arquivo.
   - Exemplo 1 de violação de memória:
```
char *array = (char *)malloc(sizeof(char) * N);
scanf("%s", &filename); // acesso indevido, violacao de memoria
free(array);
```

- 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]);
}
```

5. Ao enviar, aguarde um pouco mais de 90 segundos, **sem recarregar a página nem pressionar ESC**, para obter a resposta. Caso demore mais do que 2 minutos sem dar uma resposta, feche e abra novamente a página do servidor. Verifique se seu programa tem algum problema de entrada de dados (ou se tem algum loop infinito ou parada para pressionamento de tecla). Caso não tenha, aguarde 5 minutos e tente novamente.