Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Estruturas de Dados II
Prof. Tiago A. Almeida <talmeida@ufscar.br>

TRABALHO 03 - HASHING

Atenção

- **Prazo de entrega: 19/11/2014 – 23h55 (via Moodle).**
  Após o prazo, o trabalho não será considerado!

Indexação usando Tabelas Hash

O seu sistema de cadastro de jogadores profissionais de Dota 2 está sendo utilizado em larga escala e agora você contratou uma equipe para dar continuidade e manutenção.

O analista de dados da sua equipe identificou que agora a maior parte das operações é de busca e que são realizadas poucas inserções ou remoções de registros. Sendo assim, concluiu-se que utilizar uma estrutura de hashing poderá trazer grandes benefícios ao desempenho do sistema, permitindo que a maioria das buscas seja realizada com poucos acessos ao disco.

Lembrando, cada jogador (registro no arquivo de dados) é composto pelos seguintes campos:

- *Código* (composição das duas primeiras letras do apelido do jogador, seguidas das duas primeiras letras da equipe do jogador e os dois últimos dígitos do ano de nascimento, ex: `DENA89`) – esse campo é a chave primária, portanto, não poderá existir outro valor idêntico;
- *Apelido* (apelido do jogador, ex: Dendi);
- *Nome completo* (nome completo do jogador, ex: Danil Ishutin);
- *Data de nascimento* (data no formato `DD/MM/AAAA`, ex: 30/12/1989);
- *País de origem* (país onde o jogador reside, ex: Ucrania);
- *Posição* (posição em que o jogador normalmente assume, ex: Mid);
- *Heróis mais jogados* (três heróis mais usados pelo jogador (3 campos distintos: herói 1, herói 2 e herói 3), ex: Pudge, Invoker, Shadow Fiend);
- *Equipe do jogador* (nome da equipe, ex: Natus Vincere);
- *Número de partidas oficiais* (valor, com três dígitos, referente à quantidade de partidas oficiais que o jogador já disputou);

Garantidamente, nenhum campo de texto receberá caractere acentuado.

Tarefa

Desenvolva um programa que permita ao usuário manter uma base de dados de jogadores. O programa deverá permitir:

1. Inserir um novo jogador;
2. Modificar o campo **número de partidas oficiais** de um jogador a partir da chave primária;
3. Buscar jogadores a partir de sua chave primária;
4. Remover jogadores a partir de sua chave primária;
5. Listar a Tabela Hash.

Mais uma vez, nenhum arquivo ficará salvo em disco. O arquivo de dados será simulado em uma *string* e o índice primário será sempre criado na inicialização do programa e manipulado em memória RAM até o término da execução. Suponha que há espaço suficiente em memória RAM para todas as operações.

Arquivo de dados

Como este trabalho será corrigido pelo `OnlineJudge` e o sistema não aceita funções que manipulam arquivos, os registros serão armazenados e manipulados em uma *string* que simula o arquivo aberto. Você deve utilizar a variável global `ARQUIVO` e funções de leitura e escrita em strings, como `sprintf` e `sscanf`, para simular as operações de leitura e escrita em arquivo.

Dicas

- Você nunca deve perder a referência do começo do arquivo, então não é recomendável percorrer a *string* diretamente pelo ponteiro `ARQUIVO`. Um comando equivalente a `fseek(f, 192, SEEK_SET)` é `char *p = ARQUIVO + 192`.
- Diferentemente do `fscanf`, o `sscanf` não movimenta automaticamente o ponteiro após a leitura.
- O `sprintf` adiciona automaticamente o caractere `\0` no final da *string* escrita. Em alguns casos você precisará sobrescrever a posição.

O arquivo de dados deve ser organizado em registros de tamanho fixo de 192 bytes. Os campos *apelido*, *nome completo*, *país*, *posição*, *herói1*, *herói2*, *herói3* e *equipe* devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: *data de nascimento* (10 bytes), *número de partidas oficiais* (3 bytes) e *chave primária* (6 bytes). A soma de bytes dos campos fornecidos (incluindo os delimitadores necessários) nunca poderá ultrapassar 192 bytes. Os campos do registro devem ser separados pelo caractere delimitador @ (arroba). Cada registro terá 11 delimitadores, mais 19 bytes ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 162 bytes. Caso o registro tenha menos de 192 bytes, o espaço adicional

deve ser marcado de alguma forma a completar os 192 bytes. Caso contrário, o registro deverá ser truncado para ocupar 192 bytes. A seguir, é apresentado um exemplo com cinco registros. Os espaços adicionais foram preenchidos com o caractere ♯.

```
S4AL92@s4@Gustav Magnusson@01/01/1992@Suecia@Mid@Magnus@Puck@Templ
ar Assassin@Alliance@245@##############################################
############################################################DENA89
@Dendi@Danil Ishutin@30/12/1989@Ucrania@Mid@Pudge@Invoker@S!hadow F
iend@Natus Vincere@810@#######################################################
#########################################################MUDK90@Mushi
@Chai Yee Fung@27/11/1990@Malasia@Mid@Shadow Fiend@Queen of Pain@T
emplar Assassin@DK@492@#######################################################
#####################################################NOFN93@NOtail@Joha
n Sundstein@08/10/1993@Dinamarca@Support@Meepo@Io@Chen@Fnatic@657@
#################################################################
#########################################AREV96@Arteezy@Artour Ba
bev@01/07/1996@Canada@Mid@Shadow Fiend@Tinker@Outworld Devourer@Ev
il Geniuses@451@#######################################################
###################################
```

Note que não há quebras de linhas no arquivo (elas foram inseridas aqui apenas para exemplificar a sequência de registros).

Instruções para as operações com os registros:

- **Inserção:** cada jogador deverá ser inserido no final do arquivo de dados e atualizado no índice primário.
- **Atualização:** o único campo alterável é o de número de partidas disputadas. O registro deverá ser localizado acessando o índice primário e o número de partidas deverá ser atualizado no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção).
- **Remoção:** o registro deverá ser localizado acessando o índice primário. A remoção deverá colocar os caracteres `*|` nas primeiras posições do registro removido. O espaço do registro removido não deverá ser reutilizado para novas inserções. Observe que o registro deverá continuar ocupando exatamente 192 bytes.

Índices

Um índice primário (Tabela Hash) deve ser criado na inicialização do programa e manipulado em RAM até o encerramento da aplicação. Duas versões de tabelas hash devem ser implementadas, que se diferem na forma de solucionar colisões:

- A versão `A` aplica a técnica de endereçamento aberto com **reespalhamento linear**;
- A versão `B` aplica a técnica de **encadeamento**.

Ambas as versões devem armazenar as chaves primárias e os RRNs dos registros. Além disso, a versão `A` possui um indicador do estado em cada posição (`LIVRE`, `OCUPADO` ou `REMOVIDO`) e a versão `B` possui um ponteiro para o encadeamento em cada posição.

Deverá ser desenvolvida uma rotina para a criação do índice. A Tabela Hash será sempre criada e manipulada em memória principal na inicialização e liberada ao término do programa.

Para que isso funcione corretamente, o programa, ao iniciar realiza os seguintes passos:

1. Pergunta ao usuário se ele desejar informar um arquivo de dados:
   - Se sim: recebe o arquivo inteiro e armazena no vetor `ARQUIVO`.
   - Se não: considere que o arquivo está vazio.
2. Inicializa as estruturas de dados do índice:
   - Solicita o tamanho e cria a Tabela Hash na RAM;
   - Popula a Tabela Hash a partir do arquivo de dados, se houver.

Interação com o usuário

O programa deve permitir interação com o usuário pelo console/terminal (modo texto) via menu.

A primeira pergunta do sistema deverá ser pela existência ou não do arquivo de dados. Se existir, deve ler o arquivo e armazenar no vetor `ARQUIVO`. Em seguida, o sistema pergunta pelo tamanho da Tabela Hash, que deverá ser sempre um número primo. Você deverá calcular o primeiro primo (T) maior ou igual ao valor informado pelo usuário.

As seguintes operações devem ser fornecidas (nessa ordem):

1. **Cadastro.** O usuário deve ser capaz de inserir um novo jogador. Seu programa deve ler os seguintes campos (nessa ordem): **apelido**, **nome completo**, **data de nascimento**, **país**, **posição**, **herói 1**, **herói 2**, **herói 3**, **equipe** e **partidas oficiais**. Note que a chave **não é** inserida pelo usuário, você precisa gerar a chave para gravá-la no registro. Você precisa garantir que a data de nascimento informada está no formato "`DD/MM/AAAA`", sendo que `DD` pertence ao intervalo $[1, 31]$, `MM` pertence ao intervalo $[1, 12]$ e `AAAA` pertence ao intervalo $[1914, 2002]$. Você também precisa garantir que o número de partidas oficiais é composto por três dígitos numéricos. Caso algum dos campos esteja irregular, exibir a mensagem "`Campo invalido! Informe novamente.`" e solicitar a digitação novamente. Caso a chave primária já exista no sistema, exibir a mensagem "`ERRO: Ja existe um registro com a chave primaria.`".
   - **Versão A:** caso a Tabela Hash esteja cheia, exibir a mensagem "`ERRO: Tabela Hash esta cheia!`". Caso a inserção seja realizada com sucesso, confirmar a inserção e exibir o número de colisões.
   - **Versão B:** as chaves de uma mesma posição devem ser encadeadas de forma ordenada por chave primária. Caso a inserção seja realizada com sucesso, confirmar a inserção.
   - Em ambas as versões, a função de Hash será dada por:

$$h(k) = (1 * k_1 + 2 * k_2 + 3 * k_3 + 4 * k_4 + 5 * k_5 + 6 * k_6) \ \ mod \ \ T$$

   onde:

k = chave primária com 6 caracteres
k_i = i-ésimo caractere da chave primária
T = tamanho da tabela Hash

2. **Alteração.** O usuário deve ser capaz de alterar o número de partidas oficiais de um jogador informando a sua chave primária. Caso ela não exista, seu programa deverá exibir a mensagem "`Registro nao encontrado!`" e retornar ao menu. Caso o registro seja encontrado, certifique-se de que o novo valor informado está dentro dos padrões (3 dígitos) e, nesse caso, altere o valor do campo no arquivo de dados. Caso contrário, exiba a mensagem "`Campo invalido! Informe novamente.`" e solicite a digitação novamente.

3. **Busca.** O usuário deve ser capaz de buscar por um jogador informando a sua chave primária. Caso o jogador não exista, seu programa deve exibir a mensagem "`Registro nao encontrado!`" e retornar ao menu principal. Caso o jogador exista, todos os dados do jogador devem ser impressos na tela de forma formatada, exibindo os campos na mesma ordem de inserção.

4. **Remoção.** O usuário deve ser capaz de remover um jogador. Caso ele não exista, seu programa deverá exibir a mensagem "`Registro nao encontrado!`" e retornar ao menu. Para remover um jogador, seu programa deverá solicitar como entrada ao usuário somente o campo chave primária e a remoção deverá ser feita no arquivo de dados com o marcador `*|`.

   - **Versão A:** a posição na tabela Hash deve ser atualizada com o estado `REMOVIDO`;
   - **Versão B:** a chave deve ser removida do encadeamento.

5. **Listagem.** O sistema deverá imprimir a tabela Hash.

   - **Versão A:** Deve imprimir uma posição da tabela por linha, começando pelo índice zero, o estado da posição e a chave correspondente, caso esteja com o estado `OCUPADO`. Por exemplo, considere a Tabela Hash de tamanho 11 a seguir:

[0] Ocupado: AKAL88
[1] Ocupado: XBNA87
[2] Removido
[3] Ocupado: AREV96
[4] Ocupado: MUDK90
[5] Ocupado: ERFN93
[6] Ocupado: S4AL92
[7] Ocupado: NOFN93
[8] Ocupado: LOAL88
[9] Ocupado: EGAL93
[10] Ocupado: ADAL90

- **Versão B:** Deve imprimir uma posição da tabela por linha, começando pelo índice zero, seguido das chaves, se houverem, separadas por um único espaço em branco. Por exemplo, considere a Tabela Hash de tamanho 11 a seguir:

[0]
[1]
[2]
[3] AREV96 EGAL93
[4] BOCL90 ERFN93 MUDK90 XBNA87
[5]
[6] S4AL92
[7] LOAL88 NOFN93
[8] ADAL90
[9]
[10] AKAL88

6. **Finalizar.** Libera toda a memória alocada e encerra o programa.

Implementação

Implemente suas funções utilizando como base o código fornecido. Não modifique os trechos de código ou as estruturas já prontas. Ao imprimir alguma informação para o usuário, utilize as constantes definidas. Ao imprimir um registro, utilize a função `exibir_registro()`.

Tenha atenção redobrada ao implementar a operação de listagem da tabela Hash. Atente-se às quebras de linhas requeridas e não adicione espaços em branco após o último caractere imprimível. A saída deverá ser exata para não dar conflito com o `OnlineJudge`. Em caso de dúvidas, examine o caso de teste.

Você deve criar obrigatoriamente as seguintes funcionalidades:

- Criar o índice primário (tabela Hash): deve alocar a tabela de tamanho de um número primo na inicialização do programa;
- Carregar o índice primário: deve construir o índice primário a partir do arquivo de dados;
- Inserir um registro: modificar o arquivo de dados e o índice na memória principal;
- Buscar por registros: buscar por registros pela chave primária;
- Alterar um registro: modificar o arquivo de dados;
- Remover um registro: marcar um registro para remoção no arquivo de dados e remover do índice primário;
- Listar tabela: listar a tabela Hash;

- Finalizar: deverá ser chamada ao encerrar o programa e liberar toda a memória alocada.

Utilizar a linguagem `ANSI C`.

CUIDADOS

Leia atentamente os itens a seguir.

1. O projeto deverá ser submetido no `OnlineJudge` em dois arquivos diferentes:
   - Para a versão `A`, reespalhamento linear, arquivo com o nome `{RA}_ED2_T03A.c`;
   - Para a versão `B`, encadeamento, arquivo com o nome `{RA}_ED2_T03B.c`;
2. Não utilize acentos nos nomes de arquivos;
3. Dúvidas conceituais deverão ser colocadas nos horários de atendimento. Dificuldades em implementação, consultar o monitor da disciplina nos horários estabelecidos;
4. **Identificadores de variáveis:** escolha nomes apropriados;
5. **Documentação:** inclua cabeçalho, comentários e indentação no programa;
6. **Erros de compilação:** nota **zero** no trabalho;
7. **Tentativa de fraude:** nota **zero na média** para todos os envolvidos.