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: 29/11/2017 – 23h55 (via Moodle)**

Indexação usando Tabelas Hash

O seu sistema de cadastro de jogos na *Steam* 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 jogo (registro no arquivo de dados) é composto pelos seguintes campos:

- *Código*: composição de letras maiúsculas das duas primeiras letras do nome do jogo, seguido das duas primeiras letras do nome da desenvolvedora, do dia e mês do lançamento (com dois dígitos cada) e da classificação etária do jogo (dois dígitos), ex: `ROPS070700`. Esse campo é a *chave primária*, portanto, não poderá existir outro valor idêntico na base de dados;
- *Nome do jogo* (título pelo o qual os jogadores conhecem o jogo, ex: `Rocket League`);
- *Desenvolvedora* (nome da empresa que produziu o título, ex: `Psyonix`);
- *Data de lançamento* (data no formato `DD/MM/AAAA`, ex: 07/07/2015);
- *Classificação etária* (inteiro com 2 dígitos, representando a faixa etária recomendada, utilize zero quando um título não houver restrições de conteúdo para nenhuma faixa etária, ex: `00`);
- *Preço-base* (valor de ponto flutuante com precisão de dois dígitos referente ao preço base do produto sem descontos, ex: `36.99`);
- *Desconto* (inteiro com 3 dígitos, contendo a porcentagem de desconto que será abatida no preço original do título durante a temporada de vendas, ex: `040` - nesse caso o título em questão ficaria com o preço final igual a 22.19.
- *Categorias* (campo multi-valorado separado pelo caractere '|' se houver mais de uma categoria, ex: `ACAO|CORRIDA|ESPORTES|INDIE`);

Garantidamente, nenhum campo de texto receberá caractere acentuado.

Tarefa

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

1. Inserir um novo jogo;
2. Modificar o campo **desconto** de um jogo a partir da chave primária;
3. Buscar jogos a partir de sua chave primária;
4. Remover jogos 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.

O arquivo de dados deve ser ASCII (arquivo texto), organizado em registros de tamanho fixo de 192 bytes. Os campos *título do jogo*, *nome da desenvolvedora* e *categorias* devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: *código* (10 bytes), *data de lançamento* (10 bytes), *preço-base* (7 bytes), *classificação etária* (2 bytes) e *desconto* (3 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 7 delimitadores, mais 32 bytes ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 153 bytes. Caso o registro tenha menos de 192 bytes, o espaço adicional deve ser marcado com o caractere # de forma a completar os 192 bytes. Para evitar que o registro exceda 192 bytes, os campos variáveis devem ocupar no máximo 51 bytes.

```
LEWA041200@LEGO JURASSIC WORLD@WARNER BROS@04/12/2006@00@0055.5
5@079@ACTION|ADVENTURE|CASUAL|MULTIPLAYER|RPG|RACING@##########
############################################################
###MEKO140118@METAL GEAR SOLID V: THE PHANTOM PAIN@KOJIMA PRODU
CTIONS@14/01/2010@18@0192.25@033@ACTION|ADVENTURE|CASUAL|INDIE|
MULTIPLAYER|RPG@##################################################
######BO2K241103@BORDERLANDS 2@2K GAMES@24/11/2014@03@0055.92@1
00@ACTION|ADVENTURE|INDIE|MULTIPLAYER|RPG|RACING@##############
############################################################
#########THED271000@THE BINDING OF ISAAC@EDMUND MCMILLEN@27/10/
2015@00@0037.65@023@ACTION|ADVENTURE|INDIE|MULTIPLAYER|RPG|RACI
NG@##########################################################
############HAVA160314@HALF-LIFE 2@VALVE@16/03/2003@14@0062.25@
033@MULTIPLAYER@##############################################
############################################################
```

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 do arquivo de dados:

- **Inserção:** cada jogo deverá ser inserido no final do arquivo de dados e atualizado no índice primário.
- **Atualização:** o único campo alterável é o de *Desconto*. O registro deverá ser localizado acessando o índice primário e o desconto deverá ser atualizado no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo de *Desconto* sempre terá 3 dígitos.
- **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 deseja 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 jogo. Seu programa deve ler os seguintes campos (nessa ordem): **nome do jogo**, **desenvolvedora**, **data de lançamento**, **classificação etária**, **preço-base**, **desconto** e **categorias**. Note que a chave **não é** inserida pelo usuário, você precisa gerar a chave para gravá-la no registro. Garantidamente, os campos serão fornecidos de maneira regular, não sendo necessário um pré-processamento da entrada. Se um novo registro possuir a chave gerada igual a de um outro registro já presente no arquivo de dados, a seguinte mensagem de erro deverá ser impressa: "`ERRO: Já existe um registro com a chave primária AAAA999999.\n`", onde `AAAA999999` corresponde à chave primária do registro que está sendo inserido e `\n` indica um pulo de linha após a impressão da frase.
    - **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:

$$h(k) = [\sum_{i=1}^{8}(i * f(k_i))] \ \ mod \ \ T$$

ou seja,

$$h(k) = [f(k_1)+2*f(k_2)+3*f(k_3)+4*f(k_4)+5*f(k_5)+6*f(k_6)+7*f(k_7)+8*f(k_8)] \ \ mod \ \ T$$

onde:

$h(k)$ = função de Hash
k = chave primária com 10 caracteres
T = tamanho da tabela Hash
$f(k_i)$ = função de mapeamento do caractere k_i para um inteiro, sendo que

$$f(k_i) = \begin{cases} k_i, & \text{se } k_i \text{ for número (0 – 9)} \\ \text{índice de } k_i \text{ no alfabeto + 10}, & \text{se } k_i \text{ for letra (A = 10, B = 11, ..., Z = 35)} \end{cases}$$

2. **Alteração.** O usuário deve ser capaz de alterar o desconto de um jogo informando a sua chave primária. Caso ele não exista, seu programa deverá exibir a mensagem "`Registro não encontrado!\n`" e retornar ao menu. Caso o registro seja encontrado, certifique-se de que o novo valor informado está dentro dos padrões (*i.e.*, 3 bytes, com o valor entre 000 e 100) e, nesse caso, altere o valor do campo diretamente no arquivo de dados. Caso contrário, exiba a mensagem "`Campo inválido!\n`" e solicite a digitação novamente. Ao final da operação, imprima "`OPERACAO REALIZADA COM SUCESSO!\n`" ou "`FALHA AO REALIZAR OPERACAO!\n`".

3. **Busca.** O usuário deve ser capaz de buscar por um jogo informando a sua chave primária. Caso o jogo não exista, seu programa deve exibir a mensagem "`Registro nao encontrado!\n`" e retornar ao menu principal. Caso o jogo exista, todos os dados 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 jogo. Caso ele não exista, seu programa deverá exibir a mensagem "`Registro nao encontrado!\n`" e retornar ao menu. Para remover um jogo, 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: LEWA041200
[1] Ocupado: MEKO140118
[2] Ocupado: CAAC180614
[3] Ocupado: BO2K241103
[4] Ocupado: HAVA160314
[5] Ocupado: XCFI201105
[6] Ocupado: THED271000
[7] Livre
[8] Ocupado: GRRO120803
[9] Livre
[10] Livre

- **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] LEWA041200 MEKO140118 XCFI201105
[1] CAAC180614
[2] BO2K241103 HAVA160314
[3]
[4]
[5]
[6] THED271000
[7]
[8] GRRO120803
[9]
[10]

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

Implementação

Implemente suas funções utilizando como base os códigos fornecidos. 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`.

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.

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. 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.