**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: 23/11/2016 – 23h55 (via Moodle).**
  **Após o prazo, o trabalho não será considerado!**

## Indexação usando Tabelas Hash

Seu sistema de cadastro de *Pokémons* 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 *Pokémon* tem os seguintes campos, nesta ordem, armazenados no arquivo de dados:

- *Código* (composição de letras maiúsculas da primeira letra do nome da equipe, seguida da primeira letra do nome do treinador, seguida das duas primeiras letras do nome do *Pokémon* e do dia, mês, hora e minuto da captura (com dois dígitos cada)). Ex: MABU13081125 - esse campo é a chave primária, portanto, não poderá existir outro valor idêntico na base de dados;
- *Nome do Pokémon* (nome da espécie de Pokémon. Ex: Bulbasaur);
- *Tipo(s) do Pokémon* (separados por uma barra '/', caso o *Pokémon* se enquadre em mais de um. Ex: Grama/Venenoso);
- *Combat Points – CP* (pontos de combate do *Pokémon*, com 4 dígitos inteiros e 2 dígitos decimais, separados por um ponto '.'. Ex: 1013.06);
- *Data de captura* (data de captura do *Pokémon*, no formato DD/MM/AA. Ex: 13/08/16);
- *Hora de captura* (hora da captura do *Pokémon*, no formato HH:MM. Ex: 11:25);
- *Nome do treinador* (nome do treinador que capturou o *Pokémon*. Ex: Ash);
- *Nível do treinador* (nível do treinador que capturou o *Pokémon*, com 3 dígitos inteiros. Ex: 095);

- *Nome da equipe* (nome da equipe do treinador que capturou o *Pokémon*. Ex: Mystic);

*Garantidamente, nenhum campo de texto receberá caractere acentuado.*

## Tarefa

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

1. Inserir um novo *Pokémon*;
2. Modificar o campo **combat points** de um *Pokémon* a partir de sua chave primária;
3. Buscar *Pokémons* a partir de sua chave primária;
4. Remover *Pokémons* 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 que possuam letras em sua composição devem ser convertidos para que todas elas sejam maiúsculas no arquivo de dados. Os campos `nome do Pokémon`, `tipo do Pokémon`, `nome do treinador` e `nome da equipe` devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: `combat points` (7 *bytes*), `data de captura` (8 *bytes*), `hora da captura` (5 *bytes*), `nível do treinador` (3 *bytes*) e `código` (chave primária – 12 *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 9 delimitadores, mais 35 *bytes* ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 148 *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*, cada campo de tamanho variável deve ocupar no máximo 37 *bytes*. O programa deve impedir a inserção de registros com campos de mais de 37 *bytes*.

```
MABU13081125@BULBASAUR@GRAMA/VENENOSO@1013.06@13/08/16@11:25@ASH
@095@MYSTIC@##############################################################
##############################################################################
IBGE10081309@GEODUDE@PEDRA/TERRA@0808.00@10/08/16@13:09@BROCK@08
8@INSTINCT@##############################################################
##############################################################################
VMPS26082211@PSYDUCK@AGUA@1010.55@26/07/16@22:11@MISTY@093@VALOR
@##############################################################################
##############################################################################
MAPI08080500@PIKACHU@ELETRICO@0876.54@08/08/16@05:00@ASH@095@MYS
TIC@##############################################################################
##############################################################################
MACH02090259@CHARIZARD@FOGO/VOADOR@2222.22@02/09/16@02:59@ASH@09
5@MYSTIC@##############################################################
##############################################################################
```

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 *Pokémon* deverá ser inserido no final do arquivo de dados e atualizado nos índices.
- **Atualização:** o único campo alterável é o de `combat points`. O registro deverá ser localizado acessando o índice primário e a pontuação deverá ser atualizada no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo `combat points` sempre terá 7 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 *Pokémon*. Seu programa deve ler os seguintes campos (nessa ordem): `nome do Pokémon`, `tipo do Pokémon`, `combat points`, `data de captura`, `hora da captura`, `nome do treinador`, `nível do treinador` e `nome da equipe`. Note que a chave **não é** inserida pelo usuário, você precisa gerar a chave para gravá-la no registro. Antes de armazenar um campo, você deve fazer algumas verificações:
   - O `nome do Pokémon` informado deve ser composto apenas por letras.
   - O `tipo do Pokémon` informado deve ser composto apenas por letras e, caso haja mais de um tipo a ser informado, deve ser composto apenas por letras e pelo separador '/' (um tipo não deve iniciar ou terminar com '/').
   - Os `combat points` informados devem ser compostos por 7 *bytes* e estar no formato `NNNN.NN`, onde N pertence a [0, 9] e o separador corresponde a '.';
   - A `data de captura` informada deve estar no formato "`DD/MM/AA`", onde `AA` é maior ou igual a 16 (referente ao ano corrente e de lançamento do jogo), `MM` pertence ao intervalo [1, 12], `DD` pertence ao intervalo [1, 31], [1, 30], [1, 29] ou [1, 28], de acordo com o mês e o ano, e os dois separadores correspondem a '/';

- A `hora da captura` informada deve estar no formato `HH:MM`, onde `HH` pertence ao intervalo [0, 23], `MM` pertence ao intervalo [0, 59] e o separador corresponde a ':';
- O `nível do treinador` informado deve ser compostos por 3 *bytes* e pertencer ao intervalo [1, 100];
- O `nome da equipe` informado deve sempre corresponder a um dos três nomes a seguir: `Valor`, `Instinct` ou `Mystic` (Como a entrada não tem padrão, recomenda-se que seja feita a conversão das letras para somente maiúsculas ou somente minúsculas antes de fazer a comparação).

Caso algum dos campos esteja irregular, exibir a mensagem "`Campo inválido! Informe novamente: `" e solicitar a digitação novamente. Se a chave primária gerada já existir no arquivo de dados, a seguinte mensagem de erro deverá ser impressa: "`ERRO: Já existe um registro com a chave primária AAAA99999999.\n`", onde `AAAA99999999` corresponde à chave primária do registro que está sendo inserido.

- **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:

$$h(k) = (1 * k_1 + 2 * k_2 + 3 * k_3 + 4 * k_4 + 5 * k_5 + 6 * k_6 + 7 * k_7 \ldots$$

$$\ldots + 8 * k_8 + 9 * k_9 + 10 * k_{10} + 11 * k_{11} + 12 * k_{12}) \ \ mod \ \ T$$

onde:

$k$ = chave primária com 12 caracteres
$k_i$ = $i$-ésimo caractere da chave primária
$T$ = tamanho da Tabela Hash

2. **Alteração.** O usuário deve ser capaz de alterar os pontos de combate de um *Pokémon*, informando a sua chave primária. Caso ela não exista, seu programa deverá exibir a mensagem "`Registro não encontrado!`" e retornar ao menu. Caso o registro seja encontrado, certifique-se de que o novo valor informado está dentro dos padrões (7 *bytes*, no formato `NNNN.NN`, com `N` pertencente ao intervalo [0, 9] e um ponto '.' como separador) e, nesse caso, altere o valor do campo no arquivo de dados. Caso contrário, exiba a mensagem "`Campo inválido!`" e solicite a digitação novamente.

3. **Busca.** O usuário deve ser capaz de buscar por um *Pokémon* informando a sua chave primária. Caso o *Pokémon* não exista, seu programa deve exibir a mensagem "`Registro nao encontrado!`" e retornar ao menu principal. Caso o *Pokémon* exista, todos os dados do *Pokémon* 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 *Pokémon*. Caso ele não exista, seu programa deverá exibir a mensagem "`Registro nao encontrado!`" e retornar ao menu. Para remover um *Pokémon*, 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] Livre
     [1] Ocupado: ITDR11060330
     [2] Ocupado: MPNI16091101
     [3] Ocupado: IJEE17111948
     [4] Ocupado: ISVU14072355
     [5] Ocupado: VCHO05060003
     [6] Ocupado: MABU13081125
     [7] Ocupado: VFWA14060222
     [8] Ocupado: VBPO07040241
     [9] Ocupado: ICRA13051631
     [10] Ocupado: VVPI23041542

   - **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] IJEE17111948 ITDR11060330
     [2] MPNI16091101
     [3]
     [4] ISVU14072355
     [5] VCHO05060003
     [6] MABU13081125 VVPI23041542
     [7] ICRA13051631 VFWA14060222
     [8] VBPO07040241
     [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`.

# 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.