Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Organização e Recuperação da Informação
Prof. Tiago A. Almeida <talmeida@ufscar.br>

TRABALHO 03 - HASHING

Atenção

1. **Prazo de entrega: 24/11/2019 às 08h00 (submissão online)**
2. **Sistema de submissão:** `http://judge.sor.ufscar.br/ori/`
3. A atividade deverá ser realizada individualmente

Indexação usando Tabelas Hash

O `UFSCarona` 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, se comparadas com o volume de buscas. 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 CARONA (registro no arquivo de dados) é composto pelos seguintes campos:

- *Código*: composição de letras maiúsculas da primeira letra do nome do motorista, seguida das três primeiras letras da placa do veículo, seguidas do dia e mês da data do trajeto (com dois dígitos cada) e a hora da partida (dois primeiros dígitos). Ex: `AFGZ240907`. Esse campo é a *chave primária*, portanto, não poderá existir outro valor idêntico na base de dados;
- *Nome do motorista* (primeiro nome, pelo qual os caronistas entrarão em contato, ex: `ALEXANDRE`);
- *Gênero* (gênero do motorista, Masculino (`M`) ou Feminino (`F`));
- *Data de nascimento* (data no formato `DD/MM/AAAA`, ex: `24/09/1999`);
- *Celular* (número de contato do motorista com DDD, ex: `(15) 99815-1234`);
- *Veículo* (veículo que será usado no trajeto, ex: `BELINA`);
- *Placa* (placa do veículo, ex: `FGZ-1234`).
- *Trajeto* (campo multi-valorado separado pelo caractere '|'. O primeiro valor será sempre a origem e o último o destino, porém podem ter um ou mais trajetos no meio, ex: `AFONSO VERGUEIRO|PANNUNZIO|UFSCAR`);

- *Data do trajeto* (data no formato `DD/MM/AA`, ex: `24/09/19`);
- *Hora da partida* (hora no formato `HH:MM`, ex: `07:30`);
- *Valor* (preço da carona no formato `999.99`, ex: `004.50`);
- *Vagas* (quantidade de vagas disponíveis, ex: `3`).

Garantidamente, nenhum campo de texto receberá caractere acentuado.

Tarefa

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

1. Inserir uma nova carona;
2. Modificar o campo **vagas** a partir da chave primária;
3. Buscar caronas a partir de sua chave primária;
4. Remover caronas 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

O arquivo de dados deve ser ASCII (arquivo texto), organizado em registros de tamanho fixo de 256 bytes (256 caracteres). Os campos *nome do motorista*, *modelo do carro* e *trajeto* devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: *código* (10 bytes), *gênero* (1 byte), *data de nascimento* (10 bytes), *celular* (15 bytes), *placa do veículo* (8 bytes), *data* (8 bytes) e *hora* (5 bytes) da carona, *valor* (6 bytes) e por fim *vagas disponíveis* (1 byte). A soma de bytes dos campos fornecidos (incluindo os delimitadores necessários) nunca poderá ultrapassar 256 bytes. Os campos do registro devem ser separados pelo caractere delimitador `@` (arroba). Cada registro terá 12 delimitadores, mais 64 bytes ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 180 bytes. Caso o registro tenha menos de 256 bytes, o espaço adicional deve ser marcado com o caractere ♯ de forma a completar os 256 bytes. Para evitar que o registro exceda 256 bytes, os campos variáveis devem ocupar no máximo 180 bytes. Exemplo:

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 nova oferta de carona deverá ser inserida no final do arquivo de dados e atualizado no índice primário.

```
GABRIEL AUGUSTO@M@22/07/1995@(11) 99542-4321@CORSA@ABC-4321@SO
ROCABA|SAO ROQUE|COTIA|SAO PAULO@30/10/19@15:00@020.00@4@#####
############################################################
############################################################
#######LETICIA FERREIRA DA SILVA@F@14/04/1990@(16) 95435-2134@
ONIX@CAC-1010@SAO PAULO|SAO JOSE DOS CAMPOS|TAUBATE|LORENA|VOL
TA REDONDA|RIO DE JANEIRO@12/10/19@07:00@150.00@3@############
############################################################
###############ALEXANDRE@M@24/09/1999@(15) 99815-1234@BELINA@F
GZ-1234@AFONSO VERGUEIRO|PANNUNZIO|UFSCAR@24/09/19@07:30@004.5
0@3@########################################################
############################################################
#######################ROBERT DOWNEY JUNIOR F :(@M@04/04/1965@
(11) 98754-1252@E-TRON GT@MAN-6969@LAS VEGAS|LOS ANGELES@25/04
/20@10:15@599.90@1@##########################################
############################################################
##############################
```

- **Atualização:** o único campo alterável é o de *Vagas*. O registro deverá ser localizado acessando o índice primário e campo deverá ser atualizado no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo *Vagas* sempre terá 1 byte.
- **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 256 bytes.

Índices

Um índice primário (*Tabela Hash*) deverá ser criado na inicialização do programa e manipulado em RAM até o encerramento da aplicação. Duas versões de tabelas hash deverão 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 precisará realizar os seguintes passos:

1. Perguntar ao usuário se ele deseja informar um arquivo de dados:
    - Se sim: receber o arquivo inteiro e armazenar no vetor `ARQUIVO`.
    - Se não: considerar que o arquivo está vazio.

2. Inicializar as estruturas de dados do índice:

   - Solicitar o tamanho e criar a Tabela Hash na RAM;
   - Popular 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, deverá ler o arquivo e armazenar no vetor `ARQUIVO`. Em seguida, o sistema deverá perguntar 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 poder cadastrar uma nova carona. Seu programa deve ler os seguintes campos (nessa ordem): **nome do motorista**, **gênero**, **data de nascimento**, **celular**, **modelo do veículo**, **placa**, **trajeto**, **data da carona**, **hora**, **valor** e **quantidade de vagas**. 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

$$f(k_i) = \begin{cases} k_i, & \text{se } k_i \text{ for número (0 – 9)} \\ \text{índice de } k_i \text{ no alfabeto + 10}, & \text{se } k_i \text{ for letra (A = 11, B = 12, ..., Z = 36)} \end{cases}$$

2. **Alteração.** O usuário deve poder alterar a quantidade de vagas de uma carona informando a sua chave primária. Caso ela 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.*, 1 byte, com o valor entre 0 e 9) 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 poder buscar por uma carona informando a sua chave primária. Caso a oferta não exista, seu programa deve exibir a mensagem "`Registro nao encontrado!\n`" e retornar ao menu principal. Caso 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 poder remover uma oferta de carona. Caso ela não exista, seu programa deverá exibir a mensagem "`Registro nao encontrado!\n`" e retornar ao menu. Para remover uma carona, 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:** deverá 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: BOAK241103
[4] Ocupado: HAVA160314
[5] Ocupado: XCFI201105
[6] Ocupado: THED271000
[7] Livre
[8] Ocupado: GRRO120803
[9] Livre
[10] Livre

- **Versão B:** deverá 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] BOAK241103 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 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. Em caso de dúvidas, examine os casos de teste abertos.

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 manualmente. Você também pode utilizar o comando `strncpy` para escrever em *strings*, esse comando, diferentemente do `sprintf`, não adiciona o caractere nulo no final.
- Ao utilizar o comando `strcpy`, certifique-se que a *string* destinatária possui tamanho maior ou igual que a de origem, caso contrário poderá realizar escrita em espaço inapropriado da memória. Como alternativa use a `strncpy`.
- Não é possível retornar mais de um valor diretamente em C, mas a linguagem disponibiliza a criação de `structs` e também a passagem por referência para simular tal recurso.
- A função `strtok` permite navegar nas *substrings* de uma certa *string* dado(s) o(s) delimitador(es). Porém, tenha em mente que ela deve ser usada em uma cópia da *string* original, pois ela modifica o primeiro argumento.
- Utilize ferramentas de depuração, tais como `GDB` e `Valgrind`, para encontrar erros específicos e aumentar sua produtividade.

CUIDADOS

1. O projeto deverá ser submetido no `Judge` em dois arquivos diferentes:
    - Para a versão `A`, reespalhamento linear, arquivo com o nome `{RA}_ORI_T03A.c`;
    - Para a versão `B`, encadeamento, arquivo com o nome `{RA}_ORI_T03B.c`;
2. Não utilize acentos nos nomes de arquivos;
3. Dificuldades em implementação, consultar os monitores da disciplina nos horários estabelecidos;
4. **Documentação:** inclua cabeçalho, comentários e indentação no programa;
5. **Erros de compilação:** nota **zero** no trabalho;
6. **Tentativa de fraude:** nota **zero na média** para todos os envolvidos.