INSTITUTO FEDERAL
Catarinense
Campus Blumenau

CCA0916 - Estruturas de dados I - BCC

Bacharelado em Ciência da Computação

Prof. Dr. Paulo César Rodacki Gomes

Lista de exercêcios - 10 - Tabelas Hash

1 Objetivo

O objetivo desta atividade prática em laboratório é implementar em uma linguagem orientada a objetos ou em ANSI C uma estrutura de tabela de disperção (*hash table*) para armazenar cadastros de alunos utilizando **estratégia de listas encadeadas** para tratamento de colisão na tabela, de acordo com a representação esquemática da figura 1:

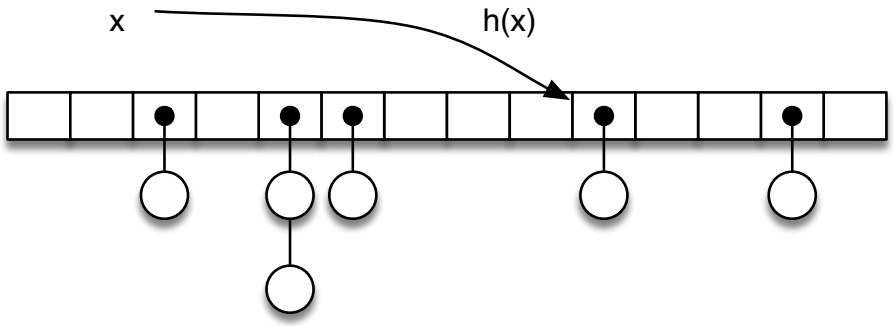

Figura 1: Representação da estratégia de tratamento de colisão

O cadastro de um aluno é armazenado em um elemento de uma lista encadeada, e a chave de busca é seu numero de matrêcula. A tabela é um *array* de ponteiros para o primeiro elemento de cada lista. Os slides com a matéria de *hash table* estão no Google Classroom.

Observação 1: as descrições das assinaturas dos métodos estão em sintaxe Java. Você deve adaptar para a linguagem orientada a objetos de sua escolha.

Observação 2: se você optar por implementar o exercêcio em ANSI C, a descrição está no final deste documento.

2 Instruções – implementação em linguagem orientada a objetos

O diagrama de classes da figura 2 mostra esquematicamente a implementação a ser feita:

2.1 Descrição dos métodos a serem implementados

1. `public TabelaHash(int N)`: construtor da classe TabelaHash, deve instanciar o *array* `tabela`. O valor N corresponde ao tamanho da tabela. Note que obtém-se melhores resultados se N for um numero primo.

2. `private int hash(int k)`: função de dispersão. Calcula um índice na tabela, a partir da chave k. No caso deste exercício, k é o número de matrícula de um aluno.

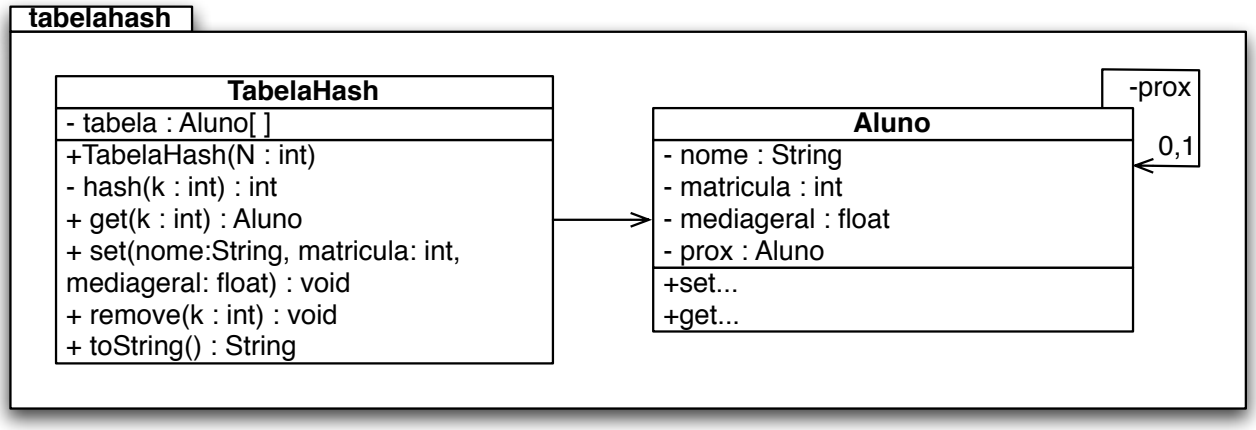

Figura 2: Diagrama de classes do pacote tabelahash

3. `public Aluno get(int k)`: retorna referência para o objeto Aluno com número de matrícula k. Se o aluno não for encontrado na tabela, retorna `null`.

4. `public void set(String nome, int matricula, float mediageral)`: insere os dados de um novo aluno. Se já existir o cadastro do aluno na tabela, altere os dados inserindo os valores passados para o método.

5. `public void remove(int k)`: remove da tabela o registro do aluno com matrícula k. Caso este aluno não seja encontrado, o método não faz qualquer alteração na tabela.

6. `public String toString()`:imprime o conteúdo de toda a tabela hash.

Implemente também uma classe contendo método `main` para testar sua implementação da tabela hash.

3 Instruções – implementação em ANSI C

1. Inicialmente crie os seguintes arquivos fonte:

   - `principal.c`: função main com demonstração do funcionamento da tabela.
   - `tabela_hash.h`: declarações de estruturas, tipos de dados e protótipos das funções.
   - `tabela_hash.c`: implementação das funções da tabela.

2. Arquivo `tabela_hash.h`:

   Abaixo, temos a estrutura de dados que deve ser implementada para o cadastro de alunos:

```
struct aluno {
   int matricula;
   char nome[81];
   char turma;
   char email[41];
   struct aluno* prox;
};

typedef struct aluno Aluno;
```

Podemos definir a tabela hash como um **tipo de dados** composto por um array de ponteiros para alunos, com uma quantidade de posições que seja um número primo, por exemplo:

```
#define N 127
typedef Aluno* Hash[N];
```

Desta forma, quando declaramos uma variável do tipo `Hash`, já estamos criando o array com quantidade prima de posições.

3. Funções a serem implementadas no arquivo `tabela_hash.c`:

   (a) `static int hash(int k)`: função de dispersão. Calcula um êndice na tabela, a partir da chave k. No caso deste exercêcio, k é o número de matrêcula de um aluno. A função deve ser declarada estática para que seu esc opo seja local (restrito ao módulo `hash.c`);

   (b) `Aluno* hsh_get(Hash tab, int mat)`: operação para buscar e retornar a referência para um Aluno com número de matrêcula `mat`. Se o aluno não for encontrado na tabela `tab`, retorna `null`.

   (c) `Aluno* hsh_set(Hash tab, int mat, char* n, char *e, char t)`: insere os dados de um novo aluno na tabela `tab`. Se já existir o cadastro do aluno na tabela, altere os dados inserindo os valores passados para a função. A função deve retornar o ponteiro para o novo aluno inserido, ou para o aluno com dados modificados;

   (d) `void hsh_remove(Hash tab, int mat)`: remove da tabela `tab` o registro do aluno com matrêcula `mat`. Caso este aluno não seja encontrado, a função não faz qualquer alteração na tabela;

   (e) `void hsh_imprime(Hash tab)`:imprime o conteúdo de toda a tabela hash.

Observação: após implementar a tabela, implemente um programa principal para testar e demonstrar o funcionamento da estrutura de dados implementada.