Algoritmos e Estruturas de Dados: Tabela de Dispersão com Encadeamento

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Maio/2011

Referências

- Mastering Algorithms with C
  Kyle Loudon
  O´Reilly, 1999
- Livros de algoritmos e estruturas de dados em geral

Tabela de Dispersão com Encadeamento – Introdução 1/5

- Cada elemento composto por chave+dados
- Chave é a parte dos dados usada para localizar o elemento
- Exemplo: Nome do funcionário em seu registro
- Elementos são armazenados em um array
- Como localizar o elemento com a chave procurada ?
- Função de dispersão (Função hash)
- Mapeia valor de chave em endereço do array

Tabela de Dispersão com Encadeamento – Introdução 2/5

- Chave pode ser de qualquer tipo
  - Número, String, etc
- Endereço é tipicamente um inteiro
- Função hash tem
  - Domínio: todos os valores possíveis para chave
  - Imagem: todos os endereços possíveis do array
- **hash("Ana") retorna 3**

0	Renata	dados da renata
1	Maria	dados da maria
2	José	dados do josé
3	Ana	dados da ana
4	Pedro	dados do pedro
5	Paulo	dados do paulo

Tabela de Dispersão com Encadeamento – Introdução 3/5

- Problema:
  Conjunto das chaves possíveis é muito maior que o conjunto dos endereços válidos
- Exemplo: uma palavra com 8 letras gera 26^8 possíveis combinações
  - Mais que 10^{11} possibilidades
- Um nome com 30 letras gera 26^{30} possíveis combinações
- O número de endereços válidos está associado com o tamanho do array
- Exemplo: Empresa tem 100 funcionários
  - Array com tamanho 100 será usado
  - Mas existem 26^{30} nomes possíveis
- Quando duas chaves mapeiam para o mesmo endereço: COLISÃO
- Objetivo da função hash é minimizar colisões

Tabela de Dispersão com Encadeamento – Introdução 4/5

- Forma possível para **tratar colisões**:
  Usar um array de listas encadeadas
- Cada lista recebe todos os elementos cuja chave resultou no mesmo valor de hash
- Inserção: Aplica a hash e coloca na frente da lista apropriada
- Consulta/Remoção: Aplica a hash e percorre a lista apropriada

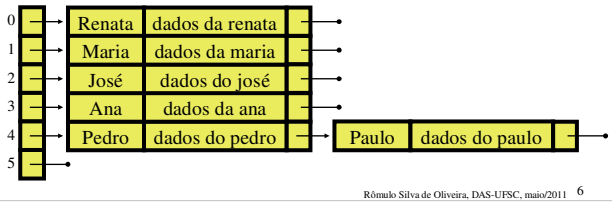

Tabela de Dispersão com Encadeamento – Introdução 5/5

- Número excessivo de colisões faz a tabela hash perder desempenho
- Objetivo é distribuir os elementos por toda a tabela
- Ideal seria ter um elemento por entrada da tabela

- Fator de Carga (load factor):
  $\alpha = N / M$
  - N é o número de elementos na tabela
  - M é o número de entradas na tabela

- Com hash uniforme e $\alpha \leq 1$ temos no máximo 1 elemento por entrada
  - 1 elemento por lista encadeada
- Bom chute: $\alpha = 0.75$

Função Hash 1/4

- Uma boa função hash espalha os elementos uniformemente e aleatoriamente pela tabela

- Informações qualitativas sobre as chaves ajuda a escolher uma boa função hash
  - Exemplo: número de matrícula do tipo 12345-2011/1

Função Hash 2/4

- **Método da divisão**
  - Simples, popular, eficaz

- Hash(k) = k % M
  - M é o número de entradas na tabela
  - M é escolhido entre os números primos longe de potência de 2

- Exemplo: N = 4500 M = 1699
  - M é primo entre 2^{10} e 2^{11}
  - $\alpha = 4500 / 1699 \approx 2,6$
  - Teremos 2 a 3 elementos por lista encadeada

Função Hash 3/4

- **Método da Multiplicação**

- Multiplica por constante entre 0 e 1
- Pega a parte fracionária
- Multiplica por M

- Hash(k) = floor[M ((k×A) % 1)]
- $A \approx 0,618$

- O tamanho da tabela não é tão crítico como no método da divisão
  - Não precisa ser primo
  - Livre para escolher a vontade o fator de carga (o M é consequência)

Função Hash 4/4

```
int hashpjw(const void *key) {
    const char *ptr;
    int val;

    val = 0;
    ptr = key;

    while (*ptr != '\0') {
        int tmp;
        val = (val << 4) + (*ptr);

        if (tmp = (val & 0xf0000000)) {
            val = val ^ (tmp >> 24);
            val = val ^ tmp;
        }
        ptr++;
    }
    return val % PRIME_TBLSIZ;
}
```


Tabela Hash com Encadeamento – Interface 1/6

- ***chtbl_init***
- int chtbl_init(CHTbl *htbl, int buckets, int (*h)(const void *key),
  int (*match)(const void *key1, const void *key2), void (*destroy)(void *data));
- **Retorno**
  - 0 se a inicialização da tabela hash teve sucesso, ou -1 caso contrário.
- **Descrição**
  - Inicializa a tabela hash encadeada especificada por *htbl*. Esta operação deve ser chamada para a tabela hash encadeada antes que a tabela hash seja usada em qualquer outra operação. O número de entradas (buckets) alocados para a tabela é especificado por *buckets*. O pointer para função *h* especifica a função hash definida pelo usuário a ser usada. O pointer para função *match* especifica a função definida pelo usuário que determina se duas chaves combinam ou não. Ela deve retornar 1 se *key1* é igual a *key2*, e 0 caso contrário.
  - O argumento *destroy* provê uma maneira de liberar dados alocados dinamicamente quando *chtbl_destroy* é chamada. Por exemplo, se a tabela hash contem dados dinamicamente alocados via *malloc*, *destroy* deveria ser associada com *free* para liberar os dados quando a tabela hash for destruida. Para dados estruturados contendo vários membros alocados dinamicamente, *destroy* deveria estar associada com uma função definida pelo usuário que chama *free* para cada membro alocado dinamicamente e para a estrutura ela própria. Para uma tabela hash contendo dados que não devem ser liberados, *destroy* deve ser NULL.

Tabela Hash com Encadeamento – Interface 2/6

- ***chtbl_destroy***
- void chtbl_destroy(CHTbl *htbl);
- **Retorno**
  - Nenhum.
- **Descrição**
  - Destrói a tabela hash encadeada especificada por *htbl*. Nenhuma outra operação é permitida após a chamada de *chtbl_destroy* a não ser que *chtbl_init* seja chamada novamente. A operação *chtbl_destroy* remove todos os elementos da tabela hash e chama a função passada como *destroy* para *chtbl_init* uma vez para cada elemento quando o mesmo é removido, desde que *destroy* não seja NULL.

Tabela Hash com Encadeamento – Interface 3/6

- ***chtbl_insert***
- int chtbl_insert(CHTbl *htbl, const void *data);
- **Retorno**
  - 0 se a inserção do elemento teve sucesso, 1 se o elemento já está na tabela hash, ou -1 caso contrário.
- **Descrição**
  - Insere um elemento em uma tabela hash encadeada especificada por *htbl*. O novo elemento contem um pointer para *data*, logo a memória referenciada por *data* deve permanecer válida pelo tempo que o elemento permanecer na tabela hash. É responsabilidade do chamador gerenciar a memória associada com os dados.

Tabela Hash com Encadeamento – Interface 4/6

- ***chtbl_remove***
- int chtbl_remove(CHTbl *htbl, void **data);
- **Retorno**
  - 0 se a remoção do elemento teve sucesso, ou -1 caso contrário.
- **Descrição**
  - Remove o elemento correspondente aos dados indicados por *data* da tabela hash encadeada especificada por *htbl*. Após o retorno, *data* aponta para os dados armazenados no elemento que foi removido. É responsabilidade do chamador gerenciar a memória associada com os dados.

Tabela Hash com Encadeamento – Interface 5/6

- ***chtbl_lookup***
- int chtbl_lookup(const CHTbl *htbl, void **data);
- **Retorno**
  - 0 se o elemento for encontrado na tabela hash, ou -1 caso contrário.
- **Descrição**
  - Determina se um elemento combina com dados na tabela hash encadeada especificada por *htbl*. Se um elemento for encontrado, após o retorno *data* aponta para os dados encontrados na tabela hash.

Tabela Hash com Encadeamento – Interface 6/6

- ***chtbl_size***
- int chtbl_size(CHTbl *htbl);
- **Retorno**
  - Número de elementos na tabela hash.
- **Descrição**
  - Macro que obtem o número de elementos na tabela hash encadeada especificada por *htbl*.

Tabela Hash com Encadeamento – Header 1/2

```
#ifndef CHTBL_H
#define CHTBL_H

#include <stdlib.h>

#include "list.h"

typedef struct CHTbl_ {

    int             buckets;

    int             (*h)(const void *key);
    int             (*match)(const void *key1, const void *key2);
    void            (*destroy)(void *data);

    int             size;
    List            *table;

} CHTbl;
```

Tabela Hash com Encadeamento – Header 1/2

```
int chtbl_init(CHTbl *htbl, int buckets, int (*h)(const void *key),
    int(*match)(const void *key1, const void *key2), void (*destroy)(void *data));

void chtbl_destroy(CHTbl *htbl);

int chtbl_insert(CHTbl *htbl, const void *data);

int chtbl_remove(CHTbl *htbl, void **data);

int chtbl_lookup(const CHTbl *htbl, void **data);

#define chtbl_size(htbl) ((htbl)->size)

#endif
```


Tabela Hash com Encadeamento – Implementação 1/6

```
#include <stdlib.h>
#include <string.h>

#include "list.h"
#include "chtbl.h"
```


Tabela Hash com Encadeamento – Implementação 2/6

```
int chtbl_init(CHTbl *htbl, int buckets, int (*h)(const void *key),
    int(*match)(const void *key1, const void *key2), void (*destroy)(void*data)) {

    int i;

    if ((htbl->table = (List *)malloc(buckets * sizeof(List))) == NULL)        return -1;

    htbl->buckets = buckets;

    for (i = 0; i < htbl->buckets; i++)
        list_init(&htbl->table[i], destroy);

    htbl->h = h;
    htbl->match = match;
    htbl->destroy = destroy;

    htbl->size = 0;
    return 0;
}
```


Tabela Hash com Encadeamento – Implementação 3/6

```
void chtbl_destroy(CHTbl *htbl) {

    int i;

    for (i = 0; i < htbl->buckets; i++) {
        list_destroy(&htbl->table[i]);
    }

    free(htbl->table);
    memset(htbl, 0, sizeof(CHTbl));
    return;
}
```


Tabela Hash com Encadeamento – Implementação 4/6

```
int chtbl_insert(CHTbl *htbl, const void *data) {

    void  *temp;
    int bucket, retval;

    temp = (void *)data;

    if (chtbl_lookup(htbl, &temp) == 0)
        return 1;

    bucket = htbl->h(data) % htbl->buckets;

    if ((retval = list_ins_next(&htbl->table[bucket], NULL, data)) == 0)
        htbl->size++;

    return retval;
}
```


Tabela Hash com Encadeamento – Implementação 5/6

```
int chtbl_remove(CHTbl *htbl, void **data) {
    ListElmt        *element, *prev;
    int bucket;

    bucket = htbl->h(*data) % htbl->buckets;

    prev = NULL;
    for (element = list_head(&htbl->table[bucket]); element != NULL; element = list_next(element)) {
        if ( htbl->match(*data, list_data(element) ) ) {
            if (list_rem_next(&htbl->table[bucket], prev, data) == 0) {
                htbl->size--;
                return 0;
            }
            else {
                return -1;
            }
        }
        prev = element;
    }
    return -1;
}
```

Tabela Hash com Encadeamento – Implementação 6/6

```
int chtbl_lookup(const CHTbl *htbl, void **data) {

    ListElmt *element;
    int bucket;

    bucket = htbl->h(*data) % htbl->buckets;

    for (element = list_head(&htbl->table[bucket]); element != NULL; element =
    list_next(element)) {

    if (htbl->match(*data, list_data(element))) {
    *data = list_data(element);
    return 0;
    }
    }

    return -1;
}
```


Tabela Hash com Encadeamento – Sumário

- Estrutura de dados simples e extremamente eficiente
- Permite pesquisa rápida em grandes volumes de dados
- Requer cuidado com a função hash usada
- Não permite caminhamento ordenado sobre os dados
  - Árvores permitem isto
- Utilizada nas mais diversas situações