Algoritmos e Estruturas de Dados: Tabela de Dispersão com Endereçamento Aberto

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 Endereçamento Aberto – Introdução 1/4

- Todos os elementos ficam na própria tabela
- Útil quando a tabela precisa ter um tamanho fixo
- No caso de colisão:
  Outras entradas da tabela são testadas até encontrar um lugar
- Fator de colisão precisa ser menor que 1
- Qual posição tentar ? hash(k, i) = x
  - **k** é a chave
  - **i** é o número de tentativas feitas até agora
  - **x** é a posição a ser testada
- A medida que i tende para M, é preciso ter certeza que todas as posições serão testadas

Tabela de Dispersão com Endereçamento Aberto – Introdução 2/4

- Suponha que serão feitas 3 inserções:
  - Hash("Ana",0) = 3
  - Hash("Renata",0) = 0
  - Hash("Paulo",0) = 3
  - Hash("Paulo",1) = 0
  - Hash("Paulo",2) = 5

0	Renata	dados da renata
1	---	
2	---	
3	Ana	dados da ana
4	---	
5	Paulo	dados do paulo

Tabela de Dispersão com Endereçamento Aberto – Introdução 3/4

- Linear Probing
  - Tenta posições sucessivas na tabela
    hash(k,i) = (hash*(k) + i) % M
  - hash* é uma função hash qualquer como as vistas antes
  - Simples
  - Qualquer M
  - Gera aglomeramentos (clustering) de posições ocupadas

Tabela de Dispersão com Endereçamento Aberto – Introdução 4/4

- Double Hashing

  hash(k, i) = (hash*(k) + i × hash**(k)) % M

  - hash* e hash** são funções hash auxiliares
- Como garantir que toda a tabela será percorrida ?
  - M potência de 2 e hash** impar
  - M primo e hash** positivo menor que M
- Exemplo:
  - hash*(k) = k % M
  - hash**(k) = 1 + (k % (M-1))

Tabela Hash com Endereçamento Aberto – Interface 1/6

- ***ohtbl_init***
- int ohtbl_init(OHTbl **htbl*, int *positions*,
  int (**h1*)(const void **key*), int (**h2*)(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.
- **Description**
  - Inicializa a tabela hash com endereçamento aberto especificada por *htbl*. Esta operação deve ser chamada para uma tabela hash com endereçamento aberto antes que a tabela hash possa ser usada com qualquer outra operação. O número de posições a serem alocadas na tabela hash é especificado por *positions*. Os pointers para função *h1* e *h2* especificam funções hash auxiliares definidas pelo usuário para hash duplo. O pointer para função *match* especifica uma função definida pelo usuário que determina se duas chaves combinam. Ela deve executar de maneira similar àquela descrita para *chtbl_init.*
  - O argumento *destroy* fornece uma maneira para liberar dados alocados dinamicamente quando *ohtbl_destroy* é chamada. Ele funciona de uma maneira similar àquela descrita para para *chtbl_destroy.* Para uma tabela hash com endereçamento aberto contendo dados que não devem ser liberados, *destroy* deverá ser NULL.

Tabela Hash com Endereçamento Aberto – Interface 2/6

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

Tabela Hash com Endereçamento Aberto – Interface 3/6

- ***ohtbl_insert***
- int ohtbl_insert(OHTbl *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 na tabela hash com endereçamento aberto especificada por *htbl*. O novo elemento contem um pointer para *data*, logo a memória referenciada por *data* deverá permanecer válida pelo tempo que o elemento permaneça na tabela hash. É responsabilidade do chamador gerenciar a memória associada com os dados.

Tabela Hash com Endereçamento Aberto – Interface 4/6

- ***ohtbl_remove***
- int ohtbl_remove(OHTbl *htbl, void **data);
- **Retorno**
  - 0 se a remoção do elemento teve sucesso, ou -1 caso contrário.
- **Descrição**
  - Remove o elemento correspondente a *data* da tabela hash com endereçamento aberto 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 Endereçamento Aberto – Interface 5/6

- ***ohtbl_lookup***
- int ohtbl_lookup(const OHTbl *htbl, void **data);
- **Retorno**
  - 0 se o elemento foi encontrado na tabela hash, ou -1 caso contrário.
- **Descrição**
  - Determina se um elemento combina com *data* na tabela hash com endereçamento aberto especificada por *htbl*. Se uma combiñação é encontrada, após o retorno *data* aponta para os dados encontrados na tabela hash.

Tabela Hash com Endereçamento Aberto – Interface 6/6

- ***ohtbl_size***
- int ohtbl_size(const OHTbl **htbl*);
- **Retorno**
  - Número de elementos na tabela hash.
- **Descrição**
  - Macro que avalia o número de elementos na tabela hash com endereçamento aberto especificada por *htbl.*

Tabela Hash com Endereçamento Aberto – Header 1/2

```
#ifndef OHTBL_H
#define OHTBL_H

#include <stdlib.h>


typedef struct OHTbl_ {

    int         positions;
    void        *vacated;

    int         (*h1)(const void *key);
    int         (*h2)(const void *key);
    int         (*match)(const void *key1, const void *key2);
    void        (*destroy)(void *data);

    int         size;
    void        **table;

} OHTbl;
```


Tabela Hash com Endereçamento Aberto – Header 2/2

```
int ohtbl_init(OHTbl *htbl, int positions,
        int (*h1)(const void *key), int (*h2)(const void *key),
        int (*match)(const void *key1, const void *key2),
        void (*destroy)(void *data));

void ohtbl_destroy(OHTbl *htbl);

int ohtbl_insert(OHTbl *htbl, const void *data);

int ohtbl_remove(OHTbl *htbl, void **data);

int ohtbl_lookup(const OHTbl *htbl, void **data);

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

#endif
```


Tabela Hash com Endereçamento Aberto – Implementação 1/6

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

#include "ohtbl.h"

// Reserve a sentinel memory address for vacated elements.

static char     vacated;
```


Tabela Hash com Endereçamento Aberto – Implementação 2/6

```
int ohtbl_init( OHTbl *htbl, int positions,
        int (*h1)(const void *key), int(*h2)(const void *key),
        int (*match)(const void *key1, const void *key2),   void (*destroy)(void *data)) {

    int i;

    if (   (htbl->table = (void **)malloc(positions * sizeof(void *))   ) == NULL)          return -1;

    htbl->positions = positions;

    for (i = 0; i < htbl->positions; i++)
        htbl->table[i] = NULL;

    htbl->vacated = &vacated;
    htbl->h1 = h1;
    htbl->h2 = h2;
    htbl->match = match;
    htbl->destroy = destroy;
    htbl->size = 0;

    return 0;
}
```


Tabela Hash com Endereçamento Aberto – Implementação 3/6

```
void ohtbl_destroy(OHTbl *htbl) {

    int  i;

    if (htbl->destroy != NULL) {
        for (i = 0; i < htbl->positions; i++) {

            if (htbl->table[i] != NULL && htbl->table[i] != htbl->vacated)
                htbl->destroy(htbl->table[i]);
        }
    }

    free(htbl->table);

    memset(htbl, 0, sizeof(OHTbl));

    return;
}
```


Tabela Hash com Endereçamento Aberto – Implementação 4/6

```
int ohtbl_insert(OHTbl *htbl, const void *data) {

    void *temp;
    int position, i;

    if (htbl->size == htbl->positions)          return -1;

    temp = (void *)data;
    if (ohtbl_lookup(htbl, &temp) == 0)       return 1;

    for (i = 0; i < htbl->positions; i++) {

        position = (htbl->h1(data) + (i * htbl->h2(data))) % htbl->positions;

        if (htbl->table[position] == NULL || htbl->table[position] == htbl->vacated) {
            htbl->table[position] = (void *)data;
            htbl->size++;
            return 0;
        }
    }
return -1;
}
```

Tabela Hash com Endereçamento Aberto – Implementação 5/6

```
int ohtbl_remove(OHTbl *htbl, void **data) {

    int position, i;

    for (i = 0; i < htbl->positions; i++) {
        position = (htbl->h1(*data) + (i * htbl->h2(*data))) % htbl->positions;
        if (htbl->table[position] == NULL) {
            return -1;
        }
        else    if (htbl->table[position] == htbl->vacated) {
                    continue;
                }

                else    if (htbl->match(htbl->table[position], *data)) {
                            *data = htbl->table[position];
                            htbl->table[position] = htbl->vacated;
                            htbl->size--;
                            return 0;
                        }
    }

    return -1;

}
```


Tabela Hash com Endereçamento Aberto – Implementação 6/6

```
int ohtbl_lookup(const OHTbl *htbl, void **data) {

    int position, i;

    for (i = 0; i < htbl->positions; i++) {

        position = (htbl->h1(*data) + (i * htbl->h2(*data))) % htbl->positions;

        if (htbl->table[position] == NULL) {
            return -1;
        }
        else    if (htbl->match(htbl->table[position], *data)) {
                    *data = htbl->table[position];
                    return 0;
                }
    }

    return -1;

}
```


Tabela Hash com Endereçamento Aberto – Sumário

- Estrutura de dados simples e eficiente
- Código mais complexo e potencialmente mais lento que a tabela hash com encadeamento
- Precisa de duas funções hash
- Economiza a memória dos pointers
- Tabela pode ter um tamanho fixo

- Não permite caminhamento ordenado sobre os dados
  - Árvores permitem isto