

# Estruturas de Dados

## Módulo 18 - Tabelas de Dispersão

PONTIFÍCIA UNIVERSIDADE CATÓLICA  
DO RIO DE JANEIRO



# Exercícios Adicionais

- P3 e P4 de 2001 a 2003:

<http://www.inf.puc-rio.br/~inf1620/P&L/links.html>

# Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,  
*Introdução a Estruturas de Dados*, Editora Campus  
(2004)

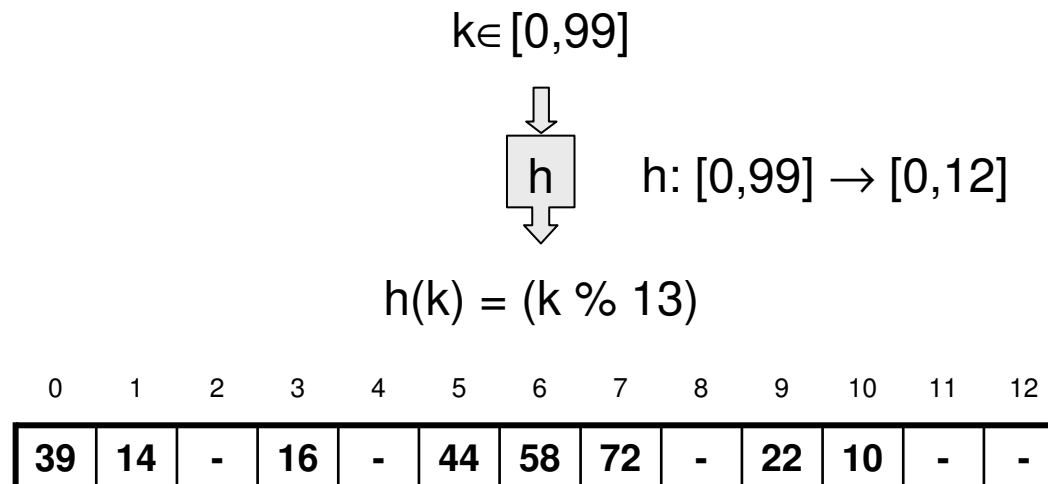
Capítulo 18 – Tabelas de dispersão

# Tópicos

- Tabelas de dispersão
- Tratamento de colisão
- Exemplo: número de ocorrências de palavras

# Tabelas de Dispersão

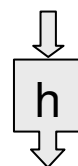
- Tabelas de dispersão (hash tables):
  - utilizadas para buscar um elemento em ordem constante  $O(1)$
  - necessitam de mais memória, proporcional ao número de elementos armazenado



# Tabelas de Dispersão

- Função de dispersão (função de *hash*):
  - mapeia uma chave de busca em um índice da tabela
  - colisão = duas ou mais chaves de busca são mapeadas para um mesmo índice da tabela de *hash*
  - deve apresentar as seguintes propriedades:
    - ser eficientemente avaliada (para acesso rápido)
    - espalhar bem as chaves de busca (para minimizarmos colisões)

$k \in [0,99]$



$h: [0,99] \rightarrow [0,12]$

$$h(k) = (k \% 13)$$

# Tabelas de Dispersão

- Dimensão da tabela:
  - deve ser escolhida para diminuir o número de colisões
  - costuma ser um valor primo
  - a *taxa de ocupação* não deve ser muito alta:
    - a taxa de ocupação não deve ser superior a 75%
    - uma taxa de 50% em geral traz bons resultados
    - uma taxa menor que 25% pode representar um gasto excessivo de memória

# Tabelas de Dispersão

- Exemplo:
  - tipo Aluno: define o tipo da estrutura de dados de interesse
  - tipo Hash: define o tipo dos vetores de ponteiros para Aluno

```
struct aluno {  
    int mat;  
    char nome[81];  
    char email[41];  
    char turma;  
};  
typedef struct aluno Aluno;  
  
#define N 127  
typedef Aluno* Hash[N];
```

# Tratamento de Colisão

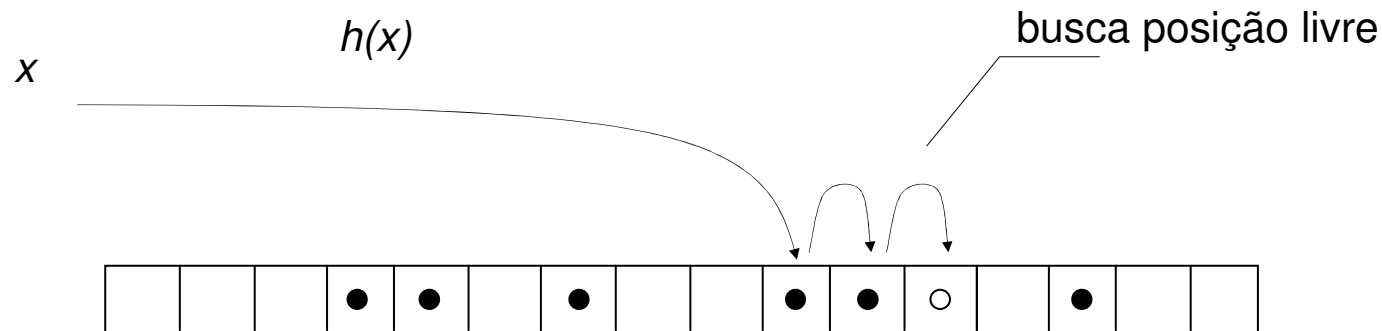
- Estratégias para tratamento de colisão:
  - uso da primeira posição consecutiva livre:
    - simples de implementar
    - tende a concentrar os lugares ocupados na tabela
  - uso de uma segunda função de dispersão:
    - evita a concentração de posições ocupadas na tabela
    - usa uma segunda função de dispersão para re-posicionar o elemento
  - uso de listas encadeadas:
    - simples de implementar
    - cada elemento da tabela hash representa um ponteiro para uma lista encadeada

# Tratamento de Colisão

- Uso da posição consecutiva livre:
  - estratégia geral:
    - armazene os elementos que colidem em outros índices, ainda não ocupados, da própria tabela
  - estratégias particulares:
    - diferem na escolha da posição ainda não ocupada para armazenar um elemento que colide

# Tratamento de Colisão

- Uso da posição consecutiva livre:
  - Estratégia 1:
    - se a função de dispersão mapeia a chave de busca para um índice já ocupado, procure o próximo índice livre da tabela (usando incremento circular) para armazenar o novo elemento



# Tratamento de Colisão

- Operação de busca
  - suponha que uma chave  $x$  for mapeada pela função de hash  $h$  para um determinado índice  $h(x)$
  - procure a ocorrência do elemento a partir de  $h(x)$ , até que o elemento seja encontrado ou que uma posição vazia seja encontrada

# Tratamento de Colisão

- Operação de busca
  - entrada: a tabela e a chave de busca
  - saída: o ponteiro do elemento, se encontrado  
NULL, se o elemento não for encontrado

```
Aluno* hsh_busca (Hash tab, int mat)
{
    int h = hash(mat);
    while (tab[h] != NULL) {
        if (tab[h]->mat == mat)
            return tab[h];
        h = (h+1) % N;
    }
    return NULL;
}
```

# Tratamento de Colisão

- Operação de inserção e modificação:
  - suponha que uma chave  $x$  for mapeada pela função de hash  $h$  para um determinado índice  $h(x)$
  - procure a ocorrência do elemento a partir de  $h(x)$ , até que o elemento seja encontrado ou que uma posição vazia seja encontrada
  - se o elemento existir, modifique o seu conteúdo
  - se não existir, insira um novo na primeira posição livre que encontrar na tabela, a partir do índice mapeado

# Tratamento de Colisão

```
Aluno* hsh_inserere (Hash tab, int mat, char* n, char* e, char t)
{
    int h = hash(mat);
    while (tab[h] != NULL) {
        if (tab[h]->mat == mat)
            break;
        h = (h+1) % N;
    }
    if (tab[h]==NULL) { /* não encontrou o elemento */
        tab[h] = (Aluno*) malloc(sizeof(Aluno));
        tab[h]->mat = mat;
    }
    /* atribui/modifica informação */
    strcpy(tab[h]->nome,n);
    strcpy(tab[h]->email,e);
    tab[h]->turma = t;
    return tab[h];
}
```

# Tratamento de Colisão

- Uso de uma segunda função de dispersão:

- exemplo:

- primeira função de hash:  $h(x) = x \% N$

- segunda função de hash:  $h'(x) = N - 2 - x \% (N - 2)$

onde  $x$  representa a chave de busca e  $N$  a dimensão da tabela

- se houver colisão, procure uma posição livre na tabela com incrementos dados por  $h'(x)$

- em lugar de tentar  $(h(x)+1)\%N$ , tente  $(h(x)+h'(x))\%N$

# Tratamento de Colisão

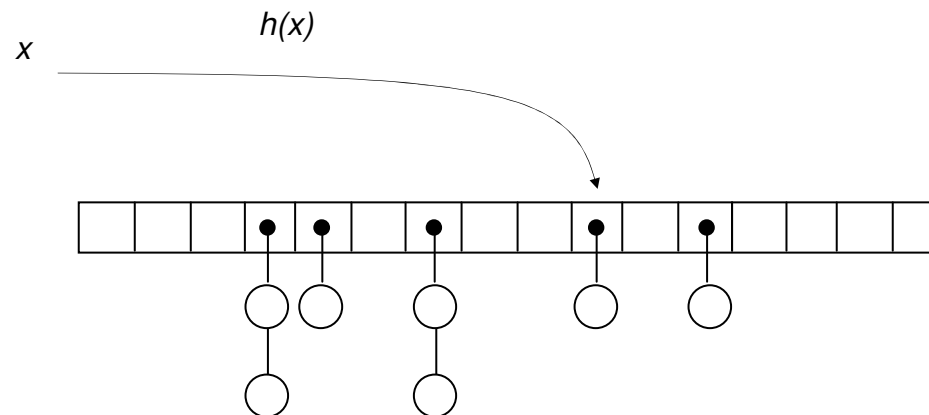
- Uso de uma segunda função de dispersão (cont):
  - cuidados na escolha da segunda função de dispersão:
    - nunca pode retornar zero
      - pois isso não faria com que o índice fosse incrementado
    - não deve retornar um número divisor da dimensão da tabela
      - pois isso limitaria a procura de uma posição livre a um sub-conjunto restrito dos índices da tabela
      - se a dimensão da tabela for um número primo, garante-se automaticamente que o resultado da função não será um divisor

# Tratamento de Colisão

```
static int hash2 (int mat)
{
    return N - 2 - mat%(N-2);
}
Aluno* hsh_busca (Hash tab, int mat)
{
    int h = hash(mat);
    int h2 = hash2(mat);
    while (tab[h] != NULL) {
        if (tab[h]->mat == mat)
            return tab[h];
        h = (h+h2) % N;
    }
    return NULL;
}
```

# Tratamento de Colisão

- Uso de listas encadeadas
  - cada elemento da tabela hash representa um ponteiro para uma lista encadeada
    - todos os elementos mapeados para um mesmo índice são armazenados na lista encadeada
    - os índices da tabela que não têm elementos associados representam listas vazias



# Tratamento de Colisão

- Exemplo:
  - cada elemento armazenado na tabela será um elemento de uma lista encadeada
  - a estrutura da informação deve prever um ponteiro adicional para o próximo elemento da lista

```
struct aluno {  
    int mat;  
    char nome[81];  
    char turma;  
    char email[41];  
    struct aluno* prox; /* encadeamento na lista de colisão */  
};  
typedef struct aluno Aluno;
```

```

/* função para inserir ou modificar um determinado elemento */
Aluno* hsh_inserere (Hash tab, int mat, char* n, char* e, char t)
{
    int h = hash(mat);
    Aluno* a = tab[h];
    while (a != NULL) {
        if (a->mat == mat)
            break;
        a = a->prox;
    }
    if (a==NULL) {      /* não encontrou o elemento */
        /* insere novo elemento no início da lista */
        a = (Aluno*) malloc(sizeof(Aluno));
        a->mat = mat;
        a->prox = tab[h];
        tab[h] = a;
    }
    /* atribui ou modifica informação */
    strcpy(a->nome,n);
    strcpy(a->email,e);
    a->turma = t;
    return a;
}

```

# Exemplo: número de ocorrências de palavras

- Especificação:
  - programa para exibir quantas vezes cada palavra ocorre em um dado texto
    - entrada: um texto T
    - saída: uma lista de palavras, em ordem decrescente do número de vezes que cada palavra ocorre em T
  - observações:
    - uma palavra é uma seqüência de uma ou mais letras (maiúsculas ou minúsculas)
    - por simplicidade, caracteres acentuados não são considerados

# Exemplo: número de ocorrências de palavras

- Armazenamento das palavras lidas e da sua freqüência:
  - tabela de dispersão
  - usa a própria palavra como chave de busca
- Função para obter a freqüência das palavras:
  - dada uma palavra, tente encontrá-la na tabela
  - se não existir, armazene a palavra na tabela
  - se existir, incremente o número de ocorrências da palavra
- Função para exibir as ocorrências em ordem decrescente:
  - crie um vetor armazenando as palavras da tabela de dispersão
  - ordene o vetor
  - exiba o conteúdo do vetor

# Exemplo: número de ocorrências de palavras

- Tabela de dispersão:
  - usa a lista encadeada para o tratamento de colisões

```
#define NPAL 64          /* dimensão máxima de cada palavra */
#define NTAB 127        /* dimensão da tabela de dispersão */
/* tipo que representa cada palavra */
struct palavra {
    char pal[NPAL];
    int n;
    struct palavra* prox; /* tratamento de colisão com listas */
};
typedef struct palavra Palavra;
/* tipo que representa a tabela de dispersão */
typedef Palavra* Hash[NTAB];
```

# Exemplo: número de ocorrências de palavras

- Leitura de palavras:
  - captura a próxima seqüência de letras do arquivo texto
  - entrada: ponteiro para o arquivo de entrada  
cadeia de caracteres armazenando a palavra capturada
  - saída: inteiro, indicando leitura bem sucedida (1) ou não (0)
  - processamento:
    - capture uma palavra, pulando os caracteres que não são letras e armazenando a seqüência de letras a partir da posição do cursor do arquivo
    - para identificar se um caractere é letra ou não, use a função `isalpha` disponibilizada pela interface `ctype.h`

```

static int le_palavra (FILE* fp, char* s)
{
    int i = 0;
    int c;
    /* pula caracteres que não são letras */
    while ((c = fgetc(fp)) != EOF) {
        if (isalpha(c))
            break;
    };
    if (c == EOF)
        return 0;
    else
        s[i++] = c;    /* primeira letra já foi capturada */
    /* lê os próximos caracteres que são letras */
    while ( i<NPAL-1 && (c = fgetc(fp)) != EOF && isalpha(c))
        s[i++] = c;
    s[i] = '\0';
    return 1;
}

```

# Exemplo: número de ocorrências de palavras

- Função para inicializar a tabela:
  - atribui NULL a cada elemento

```
static void inicializa (Hash tab)
{
    int i;
    for (i=0; i<NTAB; i++)
        tab[i] = NULL;
}
```

# Exemplo: número de ocorrências de palavras

- Função de dispersão:
  - mapeia a chave de busca (uma cadeia de caracteres) em um índice da tabela
  - soma os códigos dos caracteres que compõem a cadeia e tira o módulo dessa soma para se obter o índice da tabela

```
static int hash (char* s)
{
    int i;
    int total = 0;
    for (i=0; s[i]!='\0'; i++)
        total += s[i];
    return total % NTAB;
}
```

# Exemplo: número de ocorrências de palavras

- Função para acessar os elementos na tabela:
  - entrada: uma palavra (chave de busca)
  - saída: o ponteiro da estrutura Palavra associada
  - processamento:
    - se a palavra ainda não existir na tabela, crie uma nova palavra e forneça como retorno essa nova palavra criada

```

static Palavra *acessa (Hash tab, char* s)
{
    Palavra* p;
    int h = hash(s);
    for (p=tab[h]; p!=NULL; p=p->prox) {
        if (strcmp(p->pal,s) == 0)
            return p;
    }
    /* insere nova palavra no inicio da lista */
    p = (Palavra*) malloc(sizeof(Palavra));
    strcpy(p->pal,s);
    p->n = 0;
    p->prox = tab[h];
    tab[h] = p;
    return p;
}

```

# Exemplo: número de ocorrências de palavras

- Trecho da função principal:
  - acessa cada palavra
  - incrementa o seu número de ocorrências

```
...
inicializa(tab);
while (le_palavra(fp,s)) {
    Palavra* p = acessa(tab,s);
    p->n++;
}
...
```

# Exemplo: número de ocorrências de palavras

- Exibição do resultado ordenado:
  - crie dinamicamente um vetor para armazenar as palavras
    - vetor de ponteiros para a estrutura Palavra
    - tamanho do vetor = número de palavras armazenadas na tabela
  - coloque o vetor em ordem decrescente do número de ocorrências de cada palavra
    - se duas palavras tiverem o mesmo número de ocorrências, use a ordem alfabética como critério de desempate

```
/*  
    função para percorrer a tabela e contar o número de palavras  
    entrada: tabela de dispersão  
*/  
static int conta_elems (Hash tab)  
{  
    int i;  
    int total = 0;  
    Palavra* p;  
    for (i=0; i<NTAB; i++) {  
        for (p=tab[i]; p!=NULL; p=p->prox)  
            total++;  
    }  
    return total;  
}
```

```

/*
    função para criar dinamicamente o vetor de ponteiros
    entrada: número de elementos
           tabela de dispersão
*/
static Palavra** cria_vetor (int n, Hash tab)
{
    int i, j=0;
    Palavra* p;
    Palavra** vet = (Palavra**) malloc(n*sizeof(Palavra*));
    /* percorre tabela preenchendo vetor */
    for (i=0; i<NTAB; i++) {
        for (p=tab[i]; p!=NULL; p=p->prox)
            vet[j++] = p;
    }
    return vet;
}

```

# Exemplo: número de ocorrências de palavras

- Ordenação do vetor (de ponteiros para Palavra):
  - utilize a função *qsort* da biblioteca padrão
  - defina a função de comparação apropriadamente

```
static int compara (const void* v1, const void* v2)
{
    Palavra** p1 = (Palavra**)v1;
    Palavra** p2 = (Palavra**)v2;
    if ((*p1)->n > (*p2)->n) return -1;
    else if ((*p1)->n < (*p2)->n) return 1;
    else return strcmp((*p1)->pal,(*p2)->pal);
}
```

# Exemplo: número de ocorrências de palavras

- Impressão da tabela:

```
static void imprime (Hash tab)
{
    int i;
    int n;
    Palavra** vet;
    /* cria e ordena vetor */
    n = conta_elems(tab);
    vet = cria_vetor(n,tab);
    qsort(vet,n,sizeof(Palavra*),compara);
    /* imprime ocorrências */
    for (i=0; i<n; i++)
        printf("%s = %d\n",vet[i]->pal,vet[i]->n);
    /* libera vetor */
    free(vet);
}
```

# Exemplo: número de ocorrências de palavras

- Função Principal:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
... /* funções auxiliares mostradas acima */
```

```

int main (int argc, char** argv)
{
    FILE* fp;
    Hash tab;
    char s[NPAL];
    if (argc != 2) {
        printf("Arquivo de entrada nao fornecido.\n");
        return 0; }
    /* abre arquivo para leitura */
    fp = fopen(argv[1],"rt");
    if (fp == NULL) {
        printf("Erro na abertura do arquivo.\n");
        return 0; }
    /* conta ocorrência das palavras */
    inicializa(tab);
    while (le_palavra(fp,s)) {
        Palavra* p = acessa(tab,s);
        p->n++; }
    /* imprime ordenado */
    imprime (tab);
    return 0;
}

```

# Resumo

- Tabelas de dispersão (hash tables):
  - utilizadas para buscar um elemento em ordem constante  $O(1)$
- Função de dispersão (função de hash):
  - mapeia uma chave de busca em um índice da tabela
- Estratégias para tratamento de colisão:
  - uso da primeira posição consecutiva livre
  - uso de uma segunda função de dispersão
  - uso de listas encadeadas

