TABELAS HASH

Vanessa Braganholo

Estruturas de Dados e Seus Algoritmos

MOTIVAÇÃO

Alternativas para acelerar buscas em grandes volumes de dados:

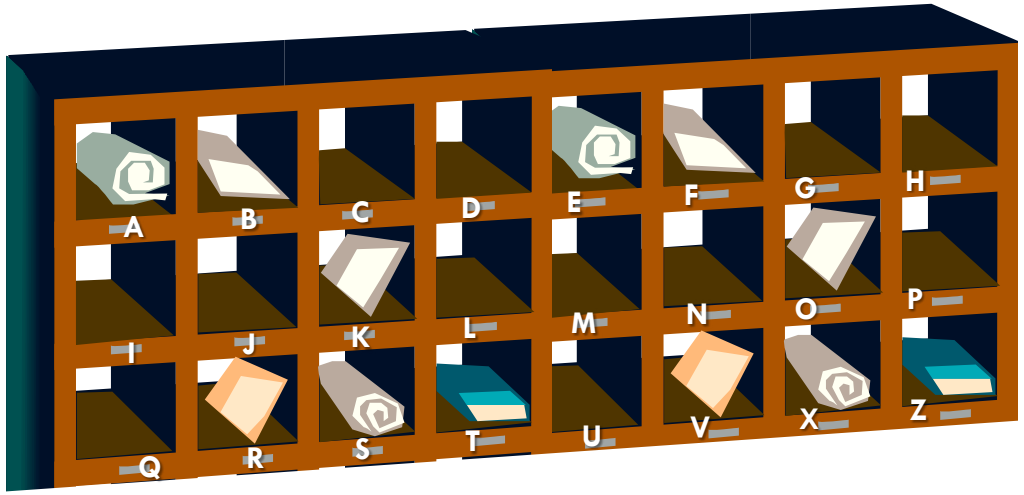
- Usar um índice (ex. Árvore B, Árvore B+)
- Usar cálculo de endereço para acessar diretamente o registro procurado em O(1) ➔ **Tabelas Hash**

EXEMPLO MOTIVADOR

Distribuição de correspondências de funcionários numa empresa

- Um escaninho para cada inicial de sobrenome
- Todos os funcionários com a mesma inicial de sobrenome procuram sua correspondência dentro do mesmo escaninho
  - Pode haver mais de uma correspondência

dentro do mesmo escaninho



HASHING: PRINCÍPIO DE FUNCIONAMENTO

Suponha que existem **n** chaves a serem armazenadas numa tabela de comprimento **m**

- Em outras palavras, a tabela tem **m** compartimentos
- Endereços possíveis: [**0, m-1**]

- Situações possíveis: cada compartimento da tabela pode armazenar **x** registros

- Para simplificar, assumimos que **x = 1** (cada compartimento armazena apenas **1** registro)

COMO DETERMINAR M?

Uma opção é determinar **m** em função do número de valores possíves das chaves a serem armazenadas

HASHING: PRINCÍPIO DE FUNCIONAMENTO

Se os valores das chaves variam de [0, m-1], então podemos usar o valor da chave para definir o endereço do compartimento onde o registro será armazenado

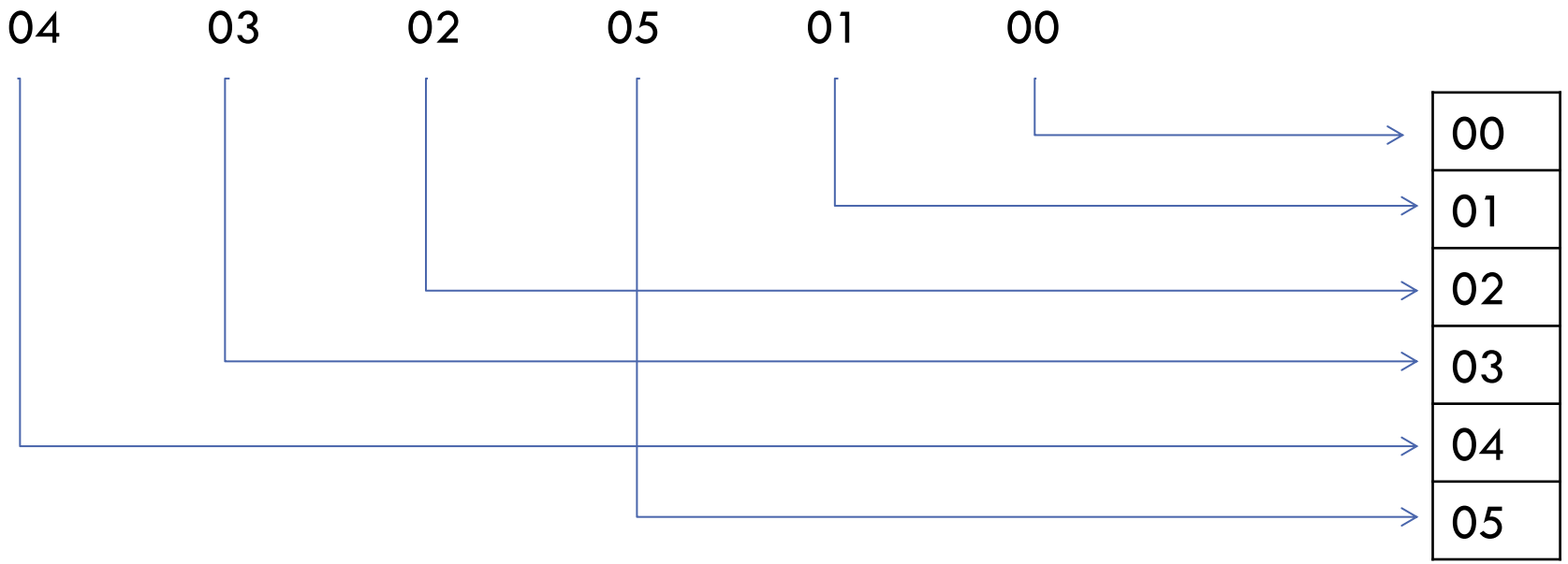
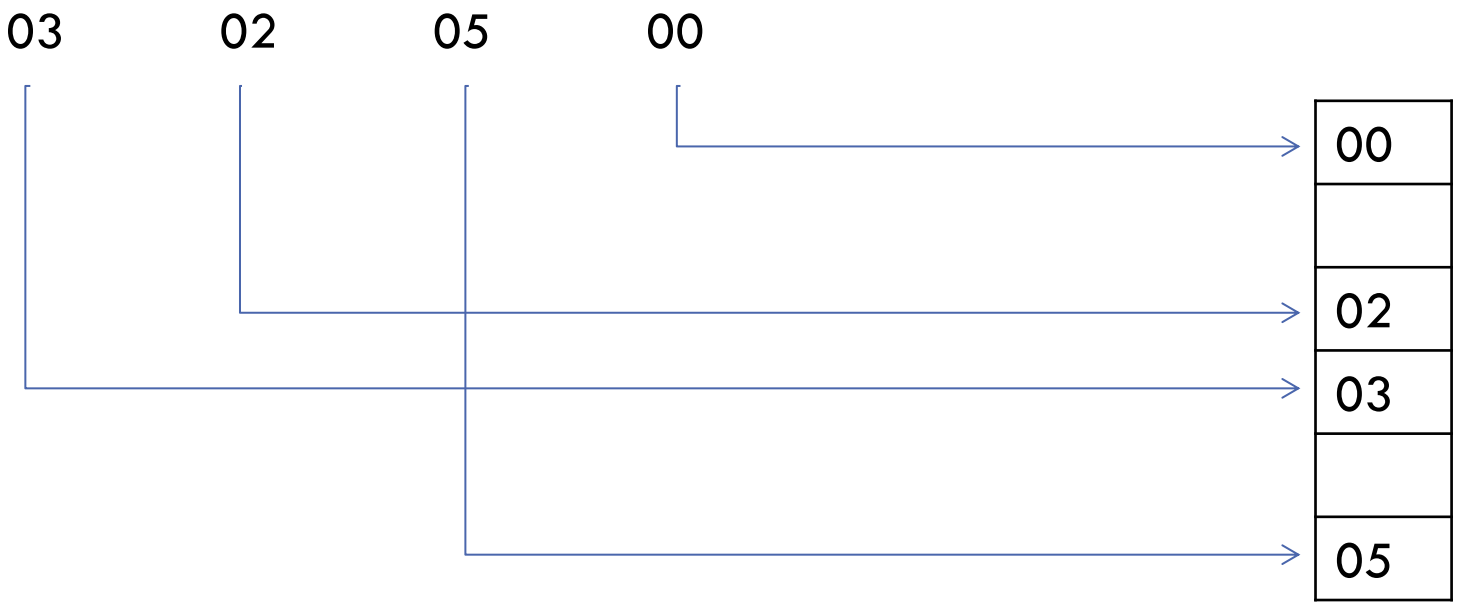


TABELA PODE TER ESPAÇOS VAZIOS

Se o número **n** de chaves a armazenar é menor que o número de compartimentos **m** da tabela



MAS...

Se o intervalo de valores de chave é muito grande, **m** é muito grande

Pode haver um número proibitivo de espaços vazios na tabela se houver poucos registros

Exemplo: armazenar 2 registros com chaves 0 e 999.999 respectivamente

- **m** = 1.000.000
- tabela teria 999.998 compartimentos vazios

SOLUÇÃO

Definir um valor de **m** menor que os valores de chaves possíveis

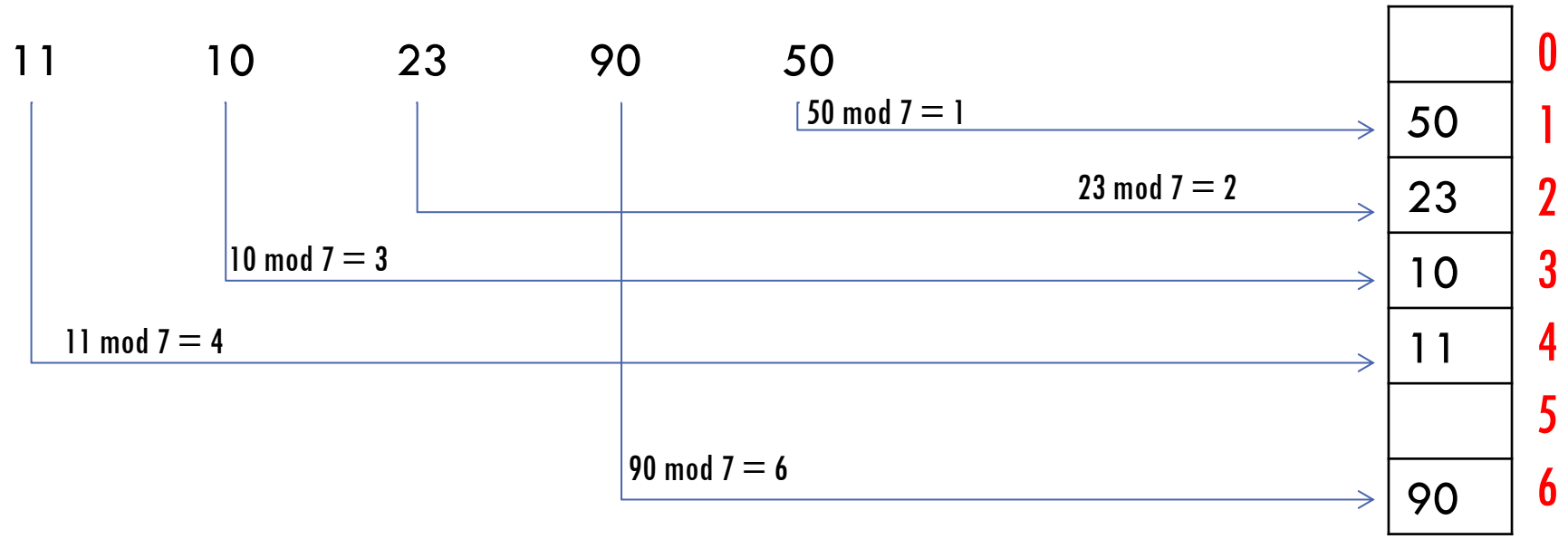
Usar uma função hash **h** que mapeia um valor de chave **x** para um endereço da tabela

Se o endereço **h(x)** estiver livre, o registro é armazenado no compartimento apontado por **h(x)**

Diz-se que **h(x)** produz um **endereço-base** para **x**

EXEMPLO

h(x) = x mod 7



FUNÇÃO HASH H

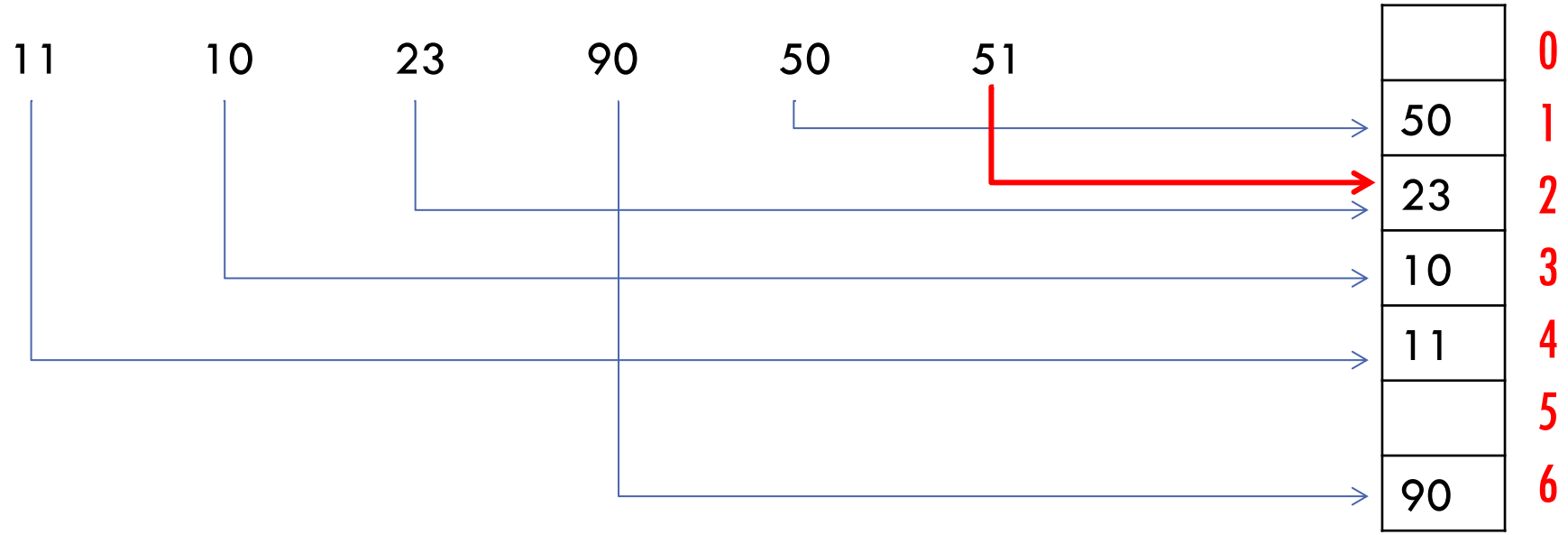
Infelizmente, a função pode não garantir injetividade, ou seja, é possível que **x ≠ y** e **h(x) = h(y)**

Se ao tentar inserir o registro de chave **x** o compartimento de endereço **h(x)** já estiver ocupado por **y**, ocorre uma **colisão**

- Diz-se que **x** e **y** são sinônimos em relação a **h**

EXEMPLO: COLISÃO

h(x) = x mod 7



A chave 51 colide com a chave 23 e não pode ser inserida no endereço 2!

Solução: uso de um procedimento especial para armazenar a chave 51 (tratamento de colisões)

CARACTERÍSTICAS DESEJÁVEIS DAS FUNÇÕES DE HASH

Produzir um número baixo de colisões

Ser facilmente computável

Ser uniforme

CARACTERÍSTICAS DESEJÁVEIS DAS FUNÇÕES DE HASH

Produzir um **número baixo de colisões**

- Difícil, pois depende da distribuição dos valores de chave
- Exemplo: Pedidos que usam o ano e mês do pedido como parte da chave
  - Se a função **h** realçar estes dados, haverá muita concentração de valores nas mesmas faixas

CARACTERÍSTICAS DESEJÁVEIS DAS FUNÇÕES DE HASH

Ser **facilmente computável**

- Se a tabela estiver armazenada em disco (nosso caso), isso não é tão crítico, pois a operação de I/O é muito custosa, e dilui este tempo
- Das 3 condições, é a mais fácil de ser garantida

Ser **uniforme**

- Idealmente, a função **h** deve ser tal que todos os compartimentos possuam a mesma probabilidade de serem escolhidos
- Difícil de testar na prática

EXEMPLOS DE FUNÇÕES DE HASH

Algumas funções de hash são bastante empregadas na prática por possuírem algumas das características anteriores:

- Método da Divisão
- Método da Dobra
- Método da Multiplicação

EXEMPLOS DE FUNÇÕES DE HASH

Método da Divisão

Método da Dobra

Método da Multiplicação

MÉTODO DA DIVISÃO

Uso da função mod:

h(x) = x mod m

onde **m** é a dimensão da tabela

Alguns valores de **m** são melhores do que outros

- Exemplo: se **m** for par, então **h(x)** será par quando **x** for par, e ímpar quando **x** for ímpar → indesejável

Atenção: na pag. 235 do livro, a fórmula contém um pequeno erro

MÉTODO DA DIVISÃO

Estudos apontam bons valores de **m**:

- Escolher **m** de modo que seja um número primo não próximo a uma potência de 2; ou
- Escolher **m** tal que não possua divisores primos menores do que 20

EXEMPLOS DE FUNÇÕES DE HASH

Método da Divisão

Método da Dobra

Método da Multiplicação

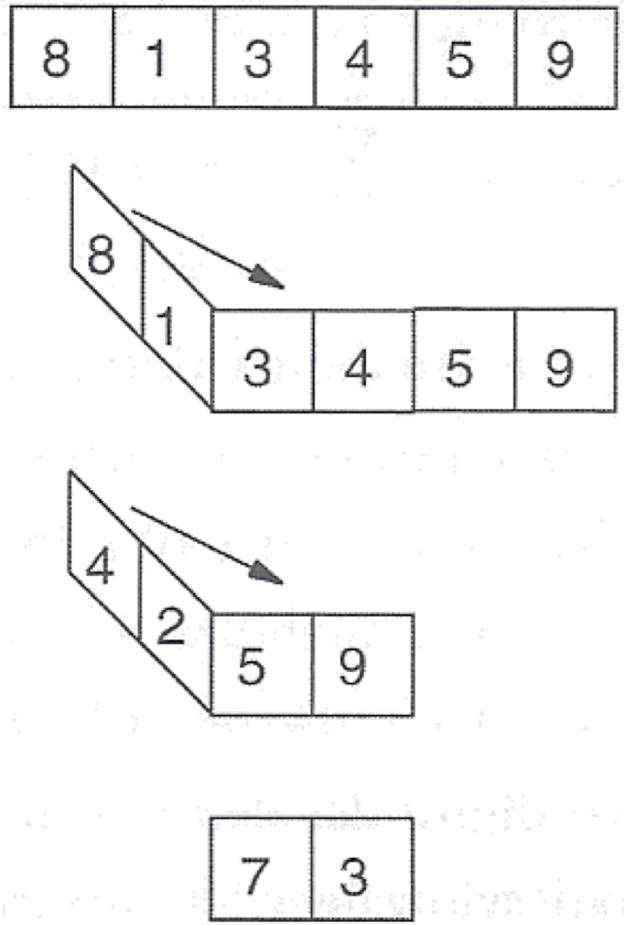
MÉTODO DA DOBRA

Suponha a chave como uma sequencia de dígitos escritos em um pedaço de papel

O método da dobra consiste em “dobrar” este papel, de maneira que os dígitos se superponham

Os dígitos então devem ser somados, sem levar em consideração o “vai-um”

EXEMPLO: MÉTODO DA DOBRA



8+4=12

1+3=4

4+9=13

2+5=7

Fonte: Fig. 10.4, pag 237

MÉTODO DA DOBRA

A posição onde a dobra será realizada, e quantas dobras serão realizadas, depende de quantos dígitos são necessários para formar o endereço base

O tamanho da dobra normalmente é do tamanho do endereço que se deseja obter

EXERCÍCIO

Escreva uma função em C que implementa o método da dobra, de forma a obter endereços de 2 dígitos

Assuma que as chaves possuem 6 dígitos

EXEMPLOS DE FUNÇÕES DE HASH

Método da Divisão

Método da Dobra

Método da Multiplicação

MÉTODO DA MULTIPLICAÇÃO

Multiplicar a chave por ela mesma

Armazenar o resultado numa palavra de **b** bits

Descartar os bits das extremidades direita e esquerda, um a um, até que o resultado tenha o tamanho de endereço desejado

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

- 12 x 12 = 144
- 144 representado em binário: 10010000
- Armazenar em 10 bits: 0010010000
- Obter endereço de 6 bits (endereços entre 0 e 63)

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

- 12 x 12 = 144
- 144 representado em binário: 10010000
- Armazenar em 10 bits: 0010010000
- Obter endereço de 6 bits (endereços entre 0 e 63)

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

- 12 x 12 = 144
- 144 representado em binário: 10010000
- Armazenar em 10 bits: 0010010000
- Obter endereço de 6 bits (endereços entre 0 e 63)

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

- 12 x 12 = 144
- 144 representado em binário: 10010000
- Armazenar em 10 bits: 0010010000
- Obter endereço de 6 bits (endereços entre 0 e 63)

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

- 12 x 12 = 144
- 144 representado em binário: 10010000
- Armazenar em 10 bits: 0010010000
- Obter endereço de 6 bits (endereços entre 0 e 63)

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

- 12 x 12 = 144
- 144 representado em binário: 10010000
- Armazenar em 10 bits: 0010010000
- Obter endereço de 6 bits (endereços entre 0 e 63)

USO DA FUNÇÃO DE HASH

A mesma função de hash usada para inserir os registros é usada para buscar os registros

EXEMPLO: BUSCA DE REGISTRO POR CHAVE

h(x) = x mod 7

Encontrar o registro de chave 90

- 90 mod 7 = 6

Encontrar o registro de chave 7

- 7 mod 7 = 0
- Compartimento 0 está vazio: registro não está armazenado na tabela

Encontrar o registro de chave 8

- 8 mod 7 = 1
- Compartimento 1 tem um registro com chave diferente da chave buscada, e não existem registros adicionais: registro não está armazenado na tabela

0	
1	50
2	23
3	10
4	11
5	
6	90

IMPLEMENTAÇÃO BÁSICA EM MEMÓRIA PRINCIPAL

Ver código da implementação básica no site da disciplina

Observações:

- Caso compartimento já esteja ocupado, **inserção é cancelada (não faz sentido na prática!!)**
- Para evitar isso, é necessário **tratar colisões**

TRATAMENTO DE COLISÕES

FATOR DE CARGA

O fator de carga de uma tabela hash é $\alpha = n/m$, onde **n** é o número de registros armazenados na tabela

- O número de colisões cresce rapidamente quando o fator de carga aumenta
- Uma forma de diminuir as colisões é diminuir o fator de carga
- Mas **isso não resolve o problema**: colisões sempre podem ocorrer

Como tratar as colisões?

TRATAMENTO DE COLISÕES

Por Encadeamento

Por Endereçamento Aberto

TRATAMENTO DE COLISÕES

Por Encadeamento

Por Endereçamento Aberto

TRATAMENTO DE COLISÕES POR ENCADEAMENTO

Encadeamento Exterior

Encadeamento Interior

ENCADEAMENTO EXTERIOR

Manter **m** listas encadeadas, uma para cada possível endereço base

A tabela base não possui nenhum registro, apenas os ponteiros para as listas encadeadas

Por isso chamamos de encadeamento **exterior**: a tabela base não armazena nenhum registro

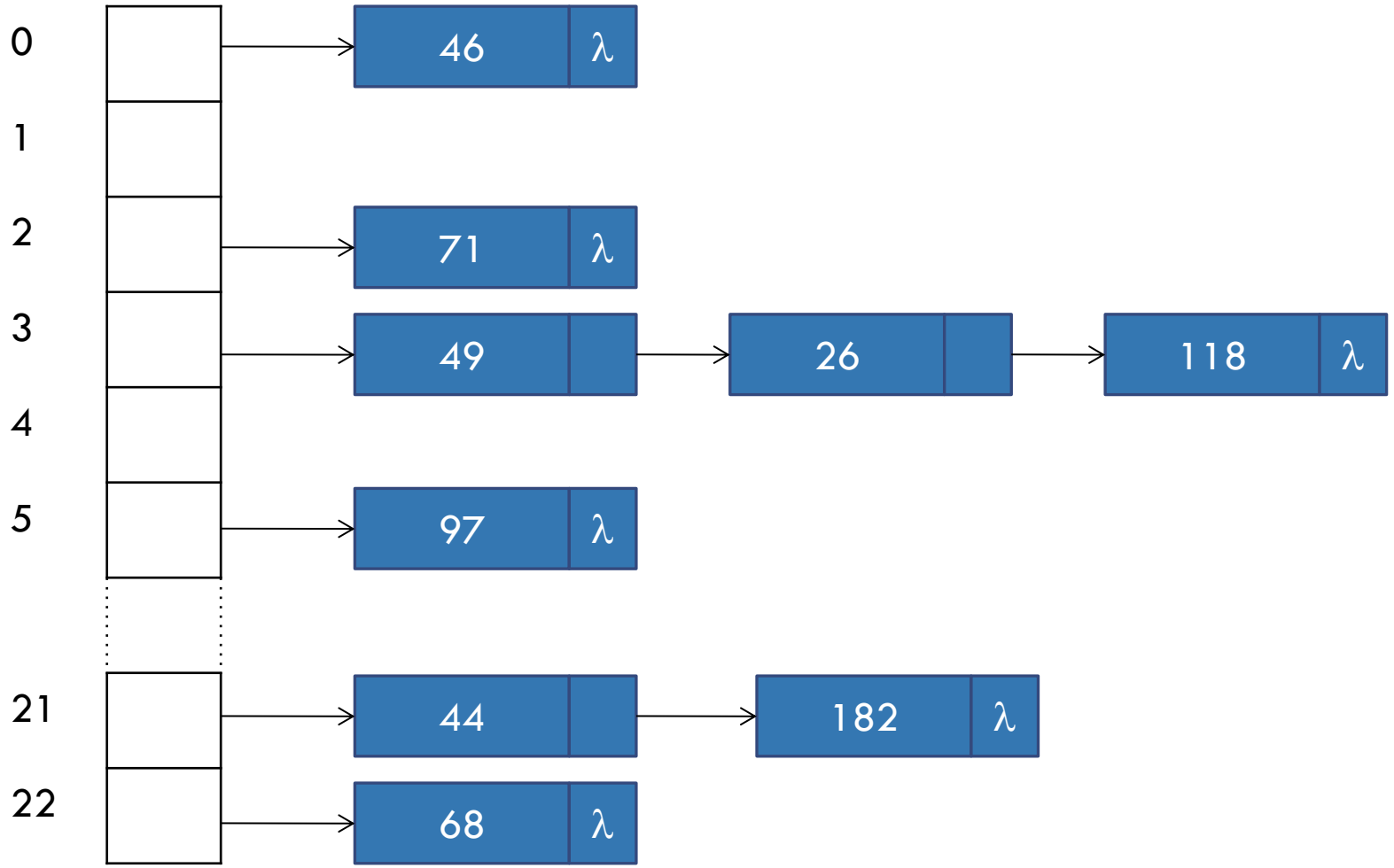
NÓS DA LISTA ENCADEADA

Cada nó da lista encadeada contém:

- um registro
- um ponteiro para o próximo nó

EXEMPLO: ENCADEAMENTO EXTERIOR

h(x) = x mod 23



Fonte: Fig. 10.5, pag 240

BUSCA EM TABELA HASH COM ENCADEAMENTO EXTERIOR

Busca por um registro de chave x:

1. Calcular o endereço aplicando a função h(x)
2. Percorrer a lista encadeada associada ao endereço
3. Comparar a chave de cada nó da lista encadeada com a chave x, até encontrar o nó desejado
4. Se final da lista for atingido, registro não está lá

INSERÇÃO EM TABELA HASH COM ENCADEAMENTO EXTERIOR

Inserção de um registro de chave x

1. Calcular o endereço aplicando a função h(x)
2. Buscar registro na lista associada ao endereço h(x)
3. Se registro for encontrado, sinalizar erro
4. Se o registro não for encontrado, inserir no final da lista

EXCLUSÃO EM TABELA HASH COM ENCADEAMENTO EXTERIOR

Exclusão de um registro de chave x

1. Calcular o endereço aplicando a função h(x)
2. Buscar registro na lista associada ao endereço h(x)
3. Se registro for encontrado, excluir registro
4. Se o registro não for encontrado, sinalizar erro

COMPLEXIDADE NO PIOR CASO

É necessário percorrer uma lista encadeada até o final para concluir que a chave não está na tabela

Comprimento de uma lista encadeada pode ser O(n)

Complexidade no pior caso: **O(n)**

COMPLEXIDADE NO CASO MÉDIO

Assume que função hash é uniforme

Número médio de comparações feitas na **busca sem sucesso** é igual ao fator de carga da tabela $\alpha = n/m$

Número médio de comparações feitas na **busca com sucesso** também é igual a $\alpha = n/m$

Se assumirmos que o número de chaves **n** é proporcional ao tamanho da tabela **m**

- $\alpha = n/m = O(1)$
- **Complexidade constante!**

IMPLEMENTAÇÃO EM MEMÓRIA PRINCIPAL

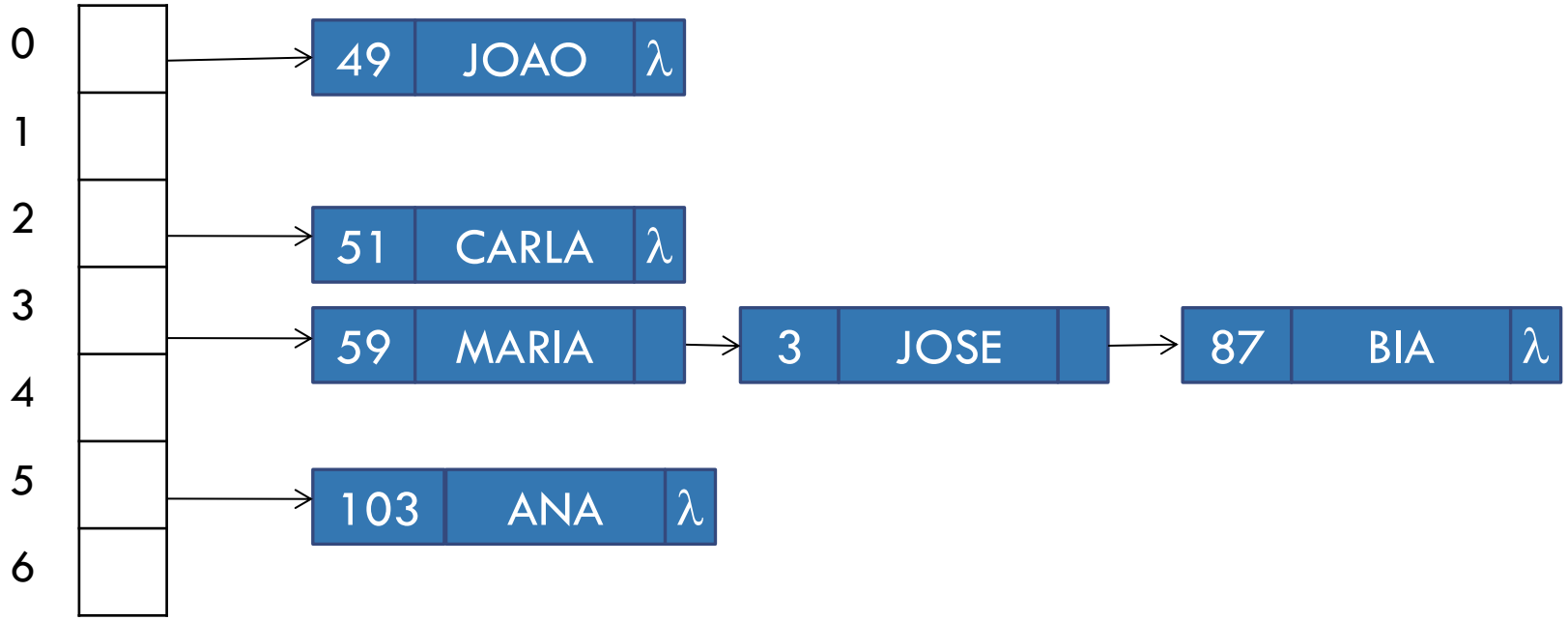
Ver implementação no site da disciplina

IMPLEMENTAÇÃO EM DISCO

Normalmente, usa-se um arquivo para armazenar os compartimentos da tabela, e outro para armazenar as listas encadeadas

Ponteiros para NULL são representados por -1

EXEMPLO



ESTRUTURA DOS ARQUIVOS

Arquivo tabHash.dat
(compartimento_hash)

0	0
1	-1
2	4
3	1
4	-1
5	2
6	-1

m = 7

Arquivo clientes.dat (cliente)

	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	3	TRUE
2	103	ANA	-1	TRUE
3	3	JOSE	5	TRUE
4	51	CARLA	-1	TRUE
5	87	BIA	-1	TRUE
6				
7				
8				
...				

USO DE FLAG INDICADOR DE STATUS

Para facilitar a manutenção da lista encadeada, pode-se adicionar um flag indicador de **status** a cada registro

No exemplo do slide anterior, esse flag é chamado **ocupado**

O flag **ocupado** pode ter os seguintes valores:

- TRUE: quando o compartimento tem um registro
- FALSE: quando o registro que estava no compartimento foi excluído

REFLEXÃO:

Como seriam os procedimentos para inclusão e exclusão?

IMPLEMENTAÇÃO DE EXCLUSÃO

- Ao excluir um registro, marca-se o flag de ocupado como FALSE (ou seja, marca-se que o compartimento está liberado para nova inserção)

IMPLEMENTAÇÃO DE INSERÇÃO (OPÇÃO 1)

Para inserir novo registro

- Inserir o registro no final da lista encadeada, se ele já não estiver na lista

- De tempos em tempos, re-arrumar o arquivo para ocupar as posições onde o flag de ocupado é FALSE

IMPLEMENTAÇÃO DE INSERÇÃO (OPÇÃO 2)

Para inserir novo registro

- Ao passar pelos registros procurando pela chave, guardar o endereço **p** do primeiro nó marcado como LIBERADO (flag ocupado = FALSE)
- Se ao chegar ao final da lista encadeada, a chave não for encontrada, gravar o registro na posição **p**
- Atualizar ponteiros
  - Nó anterior deve apontar para o registro inserido
  - Nó inserido deve apontar para nó que era apontado pelo nó anterior

EXERCÍCIO

Implementar o Encadeamento Exterior

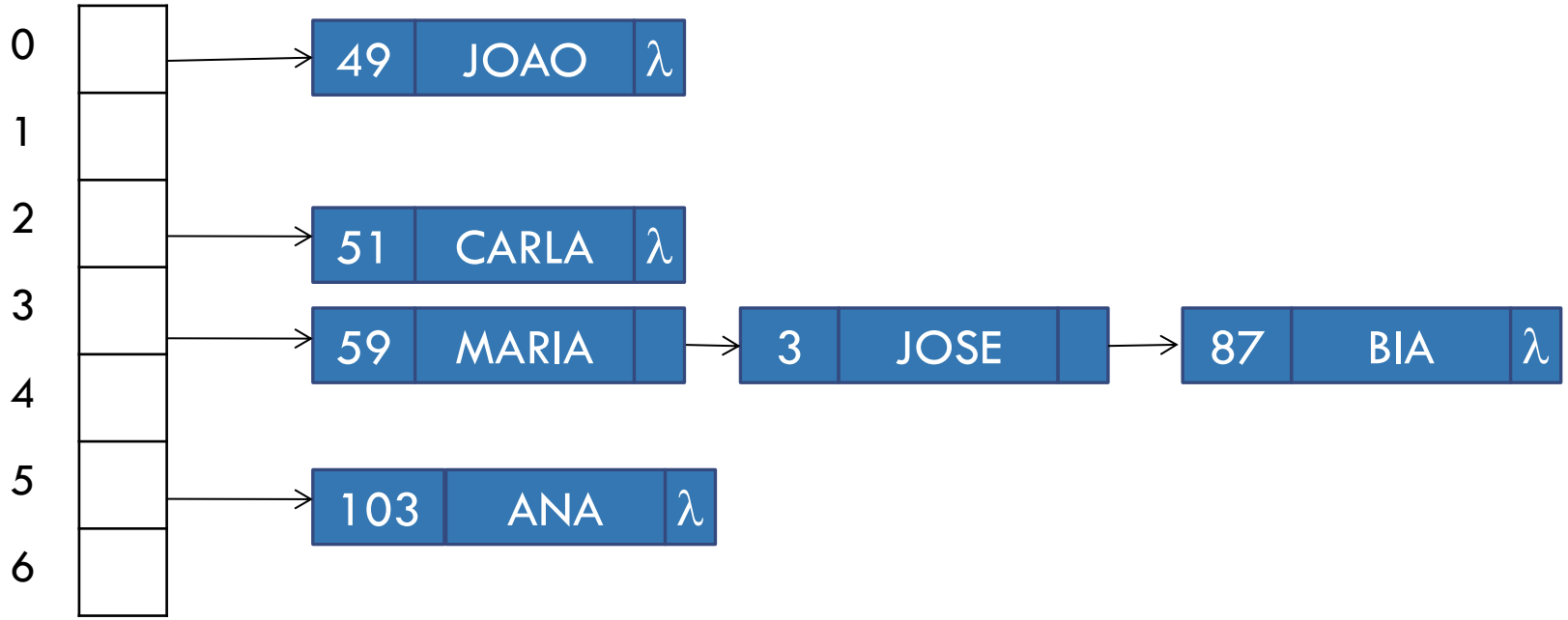
- Tamanho da tabela: m (recebido como parâmetro)
- Função de hash: h(x) = x mod 7
- Registros a inserir: Clientes (codCliente (inteiro) e nome (String de 100 caracteres))

ESTRUTURA DA IMPLEMENTAÇÃO

Uso de dois arquivos:

- tabHash.dat (modelado por compartimento_hash.h)
- clientes.dat (modelado por cliente.h)

EXEMPLO



ESTRUTURA DOS ARQUIVOS (M = 7)

Arquivo tabHash.dat
(compartimento_hash)

0	0
1	-1
2	4
3	1
4	-1
5	2
6	-1

m = 7

Arquivo clientes.dat (cliente)

	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	3	TRUE
2	103	ANA	-1	TRUE
3	3	JOSE	5	TRUE
4	51	CARLA	-1	TRUE
5	87	BIA	-1	TRUE
6				
7				
8				
...				

TRATAMENTO DE COLISÕES POR ENCADEAMENTO

Encadeamento Exterior

Encadeamento Interior

ENCADEAMENTO INTERIOR

Em algumas aplicações não é desejável manter uma estrutura externa à tabela hash, ou seja, não se pode permitir que o espaço de registros cresça indefinidamente

Nesse caso, ainda assim pode-se fazer tratamento de colisões

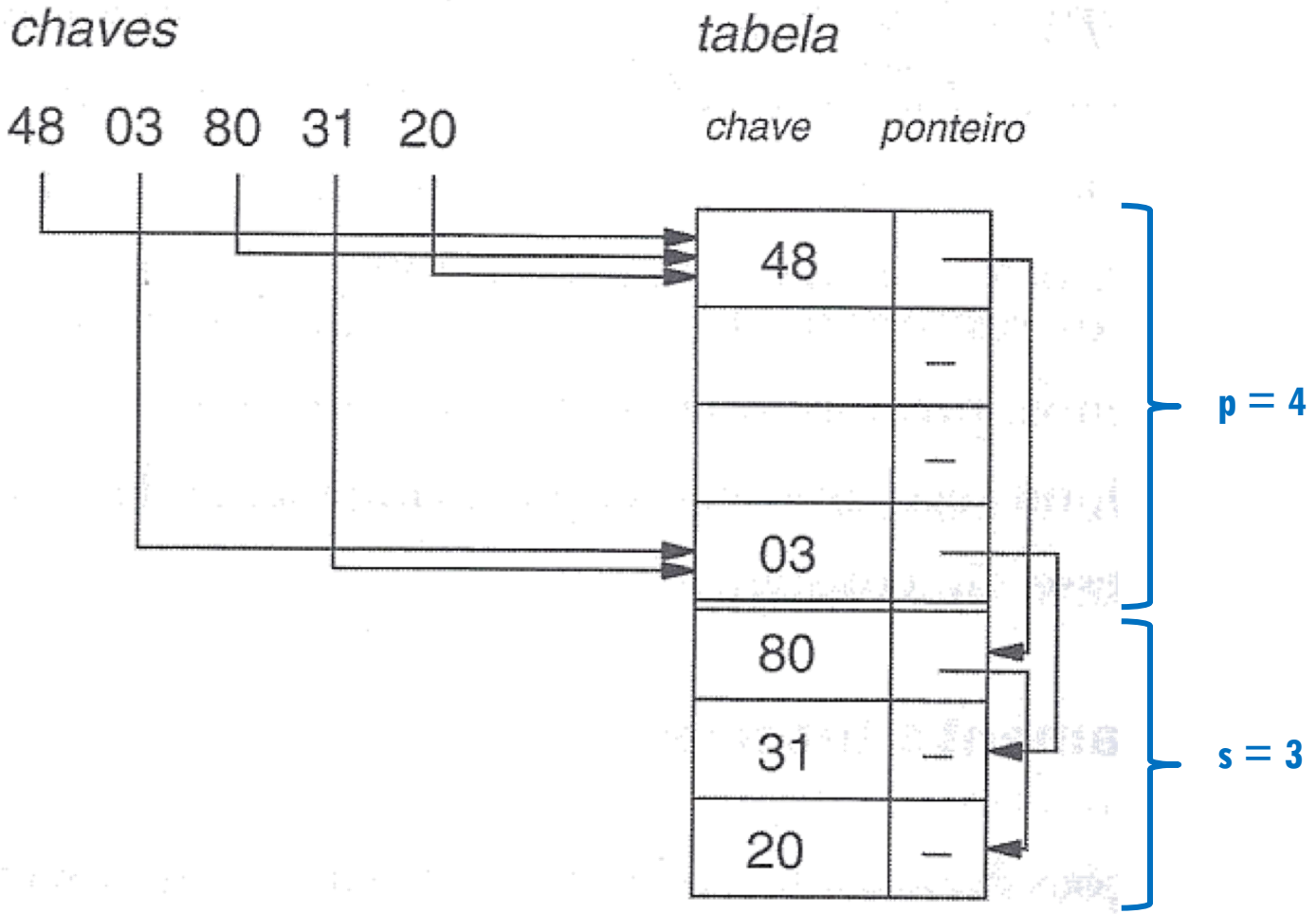
ENCADEAMENTO INTERIOR COM ZONA DE COLISÕES

Dividir a tabela em duas zonas

- Uma de endereços-base, de tamanho **p**
- Uma de colisão, de tamanho **s**
- **p + s = m**

- Função de hash deve gerar endereços no intervalo **[0, p-1]**
- Cada nó tem a mesma estrutura utilizada no Encadeamento Exterior (tabela de dados)

EXEMPLO: ENCADEAMENTO INTERIOR COM ZONA DE COLISÕES



h(x) = x mod 4

Fonte: Fig. 10.6, pag 242

OVERFLOW

Em um dado momento, pode acontecer de não haver mais espaço para inserir um novo registro

REFLEXÕES

Qual deve ser a relação entre o tamanho de **p** e **s**?

- O que acontece quando **p** é muito grande, e **s** muito pequeno?
- O que acontece quando **p** é muito pequeno, e **s** muito grande?

- Pensem nos casos extremos:
  - p = 1; s = m – 1
  - p = m-1; s = 1

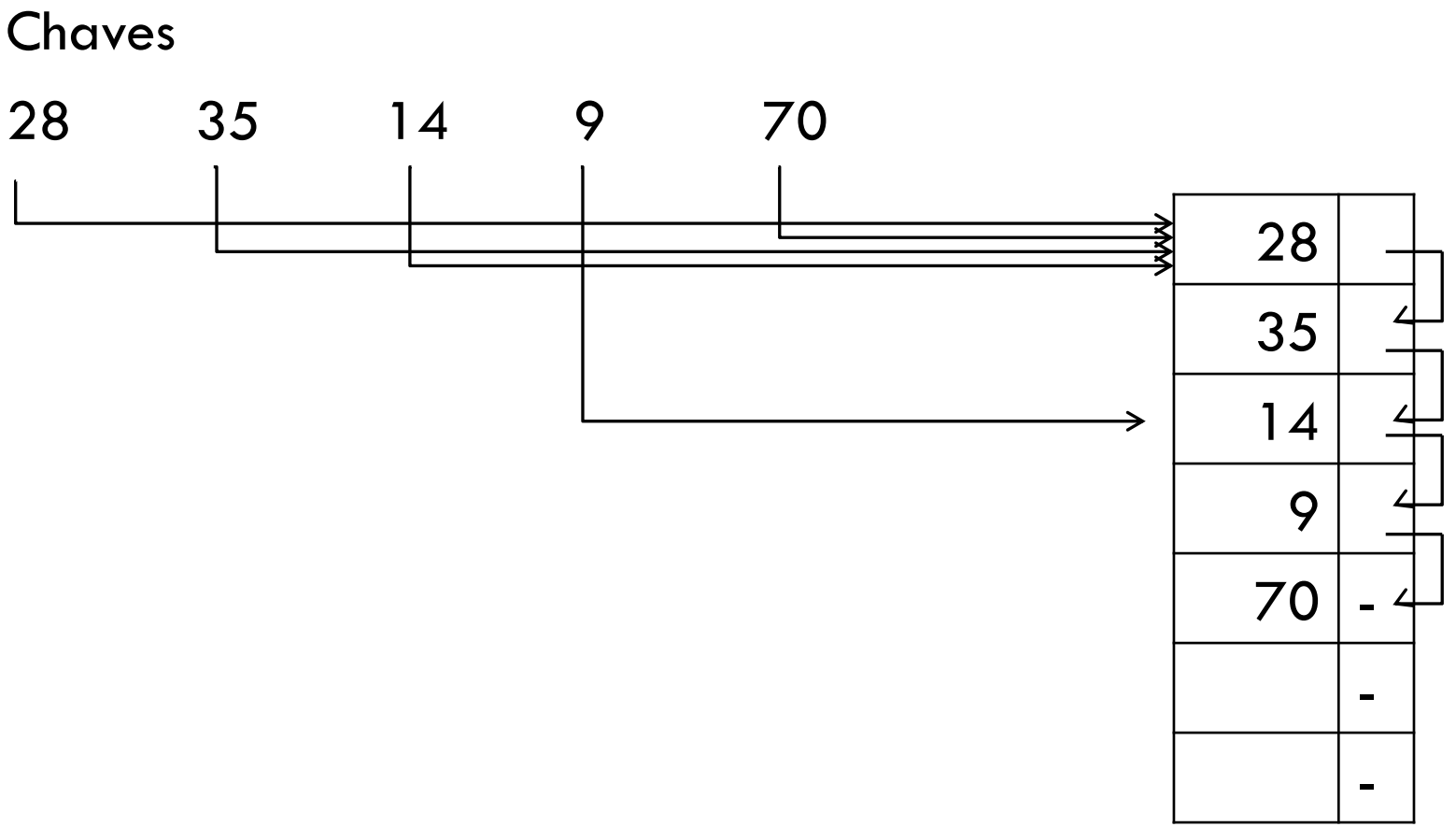
ENCADEAMENTO INTERIOR **SEM** ZONA DE COLISÕES

Outra opção de solução é não separar uma zona específica para colisões

- Qualquer endereço da tabela pode ser de base ou de colisão
- Quando ocorre colisão a chave é inserida no **primeiro compartimento vazio** a partir do compartimento em que ocorreu a colisão
- Efeito indesejado: **colisões secundárias**
  - Colisões secundárias são provenientes da coincidência de endereços para chaves que não são sinônimas

EXEMPLO: ENCADEAMENTO INTERIOR **SEM** ZONA DE COLISÕES

$h(x) = x \bmod 7$



Note que a Fig. 10.7, pag 243 do livro busca compartimentos livres de baixo para cima

IMPLEMENTAÇÃO EM MEMÓRIA PRINCIPAL

```
#define LIBERADO 0
#define OCUPADO 1


typedef struct aluno {
    int matricula;
    float cr;
    int prox;
    int ocupado;
} TAluno;


//Hash é um vetor que será alocado dinamicamente
typedef TAluno *Hash;
```

INICIALIZAÇÃO

```
TAluno *aloca(int mat, float cr, int status, int prox) {
    TAluno *novo = (TAluno *) malloc(sizeof(TAluno));
    novo->matricula = mat;
    novo->cr = cr;
    novo->ocupado = status;
    novo->prox = prox;
    return novo;
}

void inicializa(Hash *tab, int m) {
    int i;
    for (i = 0; i < m; i++) {
        tab[i] = aloca(-1, -1, LIBERADO, -1);
    }
}
```

BUSCA EM ENCADEAMENTO INTERIOR

```
/*
  Função busca assume que a tabela tenha sido inicializada
  da seguinte maneira:
    T[i].ocupado = LIBERADO, e
    T[i].pont = -1, para 0 < i < m-1

  RETORNO:
     Se chave x for encontrada, achou = 1,
     função retorna endereço onde x foi encontrada

     Se chave x não for encontrada, achou = 0, e há duas
     possibilidades para valor retornado pela função:
         endereço de algum compartimento livre, encontrado
                na lista encadeada associada a h(mat)
         -1 se não for encontrado endereço livre
*/
```

Fonte: Implementação baseada no algoritmo 10.1, pag 244 (algoritmo no livro contém pequeno erro)

```
int busca(Hash *tab, int m, int mat, int *achou) {
    *achou = -1;
    int temp = -1;
    int end = hash(mat, m);
    while (*achou == -1) {
        TAluno *aluno = tab[end];
        if (!aluno->ocupado) {//achou compartimento livre -- guarda para
 retorná-lo caso chave não seja encontrada
            temp = end;
        }
        if (aluno->matricula == mat && aluno->ocupado) {
            //achou chave procurada
            *achou = 1;
        } else {
            if (aluno->prox == -1) {
                //chegou no final da lista encadeada
                *achou = 0;
                end = temp;
            } else {
                //avança para o próximo
                end = aluno->prox;
            }
        }
    }
    return end;
}
```

INSERÇÃO EM ENCADEAMENTO INTERIOR

```
/* Função assume que pos é o endereço onde
   será efetuada a inserção. Para efeitos de
   escolha de pos, a tabela foi considerada
   como circular, isto é, o compartimento 0 é
   o seguinte ao m-1
*/
```

Ver implementação no site da disciplina

Fonte: Implementação baseada no algoritmo 10.2, pag 244

EXCLUSÃO EM ENCADEAMENTO INTERIOR

```
void exclui(Hash *tab, int m, int mat) {
    int achou;
    int end = busca(tab, m, mat, &achou);
    if (achou) {
        //remove marcando flag para liberado
         tab[end]->ocupado = LIBERADO;
    } else {
        printf("Matrícula não encontrada. Remoção não realizada!");
    }
}
```

Fonte: Implementação baseada no algoritmo 10.3, pag 245

EXERCÍCIO

Implementar o Encadeamento Interior em Disco

- Registros a inserir: Clientes (codCliente (inteiro) e nome (String de 100 caracteres))

Uso de um arquivo

- tabHash.dat (cliente.h)

ESTRUTURA DO ARQUIVO (M = 7)

h(x) = x mod 7

Arquivo tabHash.dat

	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	-1		-1	FALSE
2	51	ANA	-1	TRUE
3	59	MARIA	4	TRUE
4	10	JANIO	-1	TRUE
5	103	PEDRO	-1	TRUE
6	-1		-1	FALSE

m = 7

EXERCÍCIOS

1. Desenhe a tabela hash (em disco) resultante das seguintes operações (cumulativas) usando o algoritmo de inserção em **Tabela Hash com Encadeamento Interior SEM zona de colisão**. Considere que a tabela tem tamanho 7 e a função de hash usa o método da divisão.

(a) Inserir as chaves 10, 3, 5, 7, 12, 6, 14

(b) Inserir as chaves 4, 8

2. Repita o exercício anterior usando **Tabela Hash com Encadeamento Interior COM zona de colisão**. Considere que a zona de colisão tem tamanho 3.

3. Repita o exercício 1 usando **Tabela Hash com Encadeamento Exterior**.

TRATAMENTO DE COLISÕES

Por Encadeamento

Por Endereçamento Aberto

TRATAMENTO DE COLISÕES POR ENDEREÇAMENTO ABERTO

Motivação: as abordagens anteriores utilizam ponteiros nas listas encadeadas

- Aumento no consumo de espaço

Alternativa: armazenar apenas os registros, sem os ponteiros

Quando houver colisão, determina-se, por cálculo de novo endereço, o próximo compartimento a ser examinado

FUNCIONAMENTO

Para cada chave **x**, é necessário que todos os compartimentos possam ser examinados

A função **h(x)** deve fornecer, ao invés de um único endereço, um conjunto de **m** endereços base

Nova forma da função: **h(x,k)**, onde **k = 0, …, m-1**

Para encontrar a chave **x** deve-se tentar o endereço base **h(x,0)**

Se estiver ocupado com outra chave, tentar **h(x,1)**, e assim sucessivamente

SEQUÊNCIA DE TENTATIVAS

A sequência **h(x,0), h(x,1), …, h(x, m-1)** é denominada **sequencia de tentativas**

A sequencia de tentativas é uma **permutação** do conjunto {0, m-1}

Portanto: para cada chave **x** a função **h** deve ser capaz de fornecer uma permutação de endereços base

FUNÇÃO HASH

Exemplos de funções hash p/ gerar sequência de tentativas

- Tentativa Linear
- Tentativa Quadrática
- Dispersão Dupla

FUNÇÃO HASH

Exemplos de funções hash p/ gerar sequência de tentativas

- **Tentativa Linear**
- Tentativa Quadrática
- Dispersão Dupla

TENTATIVA LINEAR

Suponha que o endereço base de uma chave **x** é **h'(x)**

Suponha que já existe uma chave **y** ocupando o endereço **h'(x)**

Ideia: tentar armazenar **x** no endereço consecutivo a **h'(x)**. Se já estiver ocupado, tenta-se o próximo e assim sucessivamente

Considera-se uma tabela circular

h(x, k) = (h'(x) + k) mod m, $0 \leq k \leq m-1$

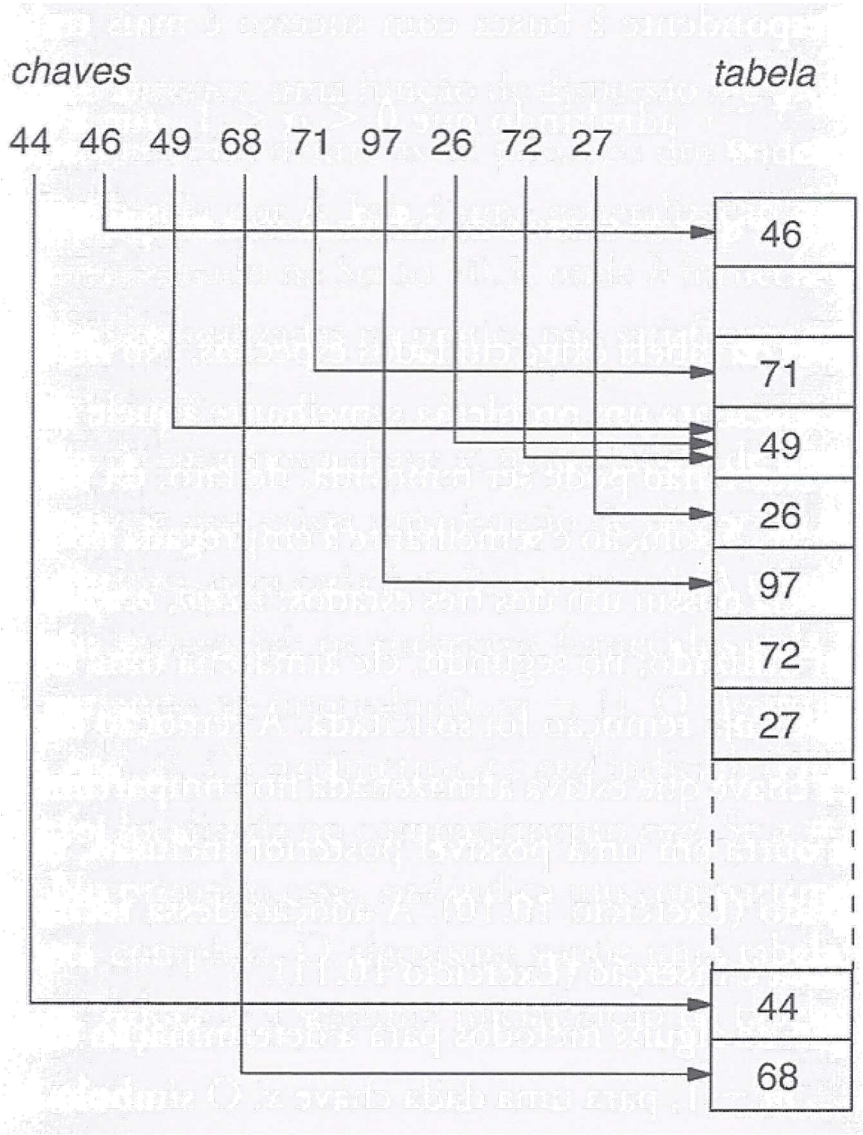
EXEMPLO TENTATIVA LINEAR

Observem a tentativa de inserir chave 26

Endereço já está ocupado: inserir no próximo endereço livre

h(x, k) = (h'(x) + k) mod m

h'(x) = x mod 23



IMPLEMENTAÇÃO ENDEREÇAMENTO ABERTO (EM MEMÓRIA PRINCIPAL)

```
typedef struct aluno {
    int matricula;
    float cr;
} TAluno;

typedef TAluno *Hash; //Hash é um vetor que será alocado dinamicamente


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

BUSCA POR ENDEREÇAMENTO ABERTO

```
int hash_linha(int mat, int m) {
    return mat % m;
}

int hash(int mat, int m, int k) {
    return (hash_linha(mat, m) + k) % m;
}

/*
 * Função busca

   RETORNO:
     Se chave mat for encontrada, achou = 1,
     função retorna endereço onde mat foi encontrada

     Se chave mat não for encontrada, achou = 0, e há duas
     possibilidades para valor retornado pela função:
      endereço de algum compartimento livre encontrado durante a busca
      -1 se não for encontrado endereço livre (tabela foi percorrida até o final)
 */
```

```
int busca(Hash *tab, int m, int mat, int *achou) {
    *achou = 0;
    int end = -1;
    int pos_livre = -1;
    int k = 0;
    while (k < m) {
        end = hash(mat, m, k);
        if (tab[end] != NULL && tab[end]->matricula == mat) {//encontrou chave
            *achou = 1;
            k = m; //força saída do loop
        }
        else {
            if (tab[end] == NULL) {//encontrou endereço livre
                //se for o primeiro, registra isso
                if (pos_livre == -1)
                    pos_livre = end;
            }
            k = k + 1; //continua procurando
        }
    }
    if (*achou)
        return end;
    else
        return pos_livre;
}
```

Fonte: Algoritmo 10.4, pag 247. No livro, a busca termina assim que um compartimento livre é encontrado. Mas isso não trata o caso de exclusão.

INSERÇÃO EM ENDEREÇAMENTO ABERTO

```
// Função insere assume que end é o endereço onde será efetuada a inserção
void insere(Hash *tab, int m, int mat, float cr) {
    int achou;
    int end = busca(tab, m, mat, &achou);
    if (!achou) {
        if (end != -1) {//Não encontrou a chave, mas encontrou posição livre
            //Inserção será realizada nessa posição
            tab[end] = aloca(mat, cr);
        } else {
            //Não foi encontrada posição livre durante a busca: overflow
            printf("Ocorreu overflow. Inserção não realizada!\n");
        }
    } else {
        printf("Matricula já existe. Inserção inválida! \n");
    }
}
```

EXCLUSÃO EM ENDEREÇAMENTO ABERTO

```
void exclui(Hash *tab, int m, int mat) {
    int achou;
    int end = busca(tab, m, mat, &achou);
    if (achou) {
        //remove
        free(tab[end]);
        tab[end] = NULL;
    } else {
        printf("Matricula não encontrada. Remoção não realizada!");
    }
}
```

DISCUSSÃO DO ALGORITMO

Na presença de remoções, a inserção precisa que a busca percorra toda a tabela até ter certeza de que o registro procurado não existe

Em situações onde não há remoção, a busca pode parar assim que encontrar um compartimento livre (se a chave existisse, ela estaria ali)

QUAIS SÃO AS DESVANTAGENS DA TENTATIVA LINEAR?

QUAIS SÃO AS DESVANTAGENS DA TENTATIVA LINEAR?

Suponha um trecho de **j** compartimentos consecutivos ocupados (chama-se **agrupamento primário**) e um compartimento **l** vazio imediatamente seguinte a esses

Suponha que uma chave **x** precisa ser inserida em um dos **j** compartimentos

- **x** será armazenada em **l**
- isso aumenta o tamanho do **agrupamento primário** para **j + 1**
- Quanto maior for o tamanho de um agrupamento primário, maior a probabilidade de aumentá-lo ainda mais mediante a inserção de uma nova chave

FUNÇÃO HASH

Exemplos de funções hash p/ gerar sequência de tentativas

- Tentativa Linear
- **Tentativa Quadrática**
- Dispersão Dupla

TENTATIVA QUADRÁTICA

Para mitigar a formação de agrupamentos primários, que aumentam muito o tempo de busca:

- Obter sequências de endereços para endereços-base próximos, porém diferentes
- Utilizar como incremento uma **função quadrática de k**

- **$h(x,k) = (h'(x) + c_1\ k + c_2\ k^2)\ \mathbf{mod}\ m$,**

onde c_1 e c_2 são constantes, $c_2 \neq 0$ e $k = 0, \ldots, m-1$

TENTATIVA QUADRÁTICA

Método evita agrupamentos primários

Mas… se duas chaves tiverem a mesma tentativa inicial, vão produzir sequências de tentativas idênticas: **agrupamento secundário**

TENTATIVA QUADRÁTICA

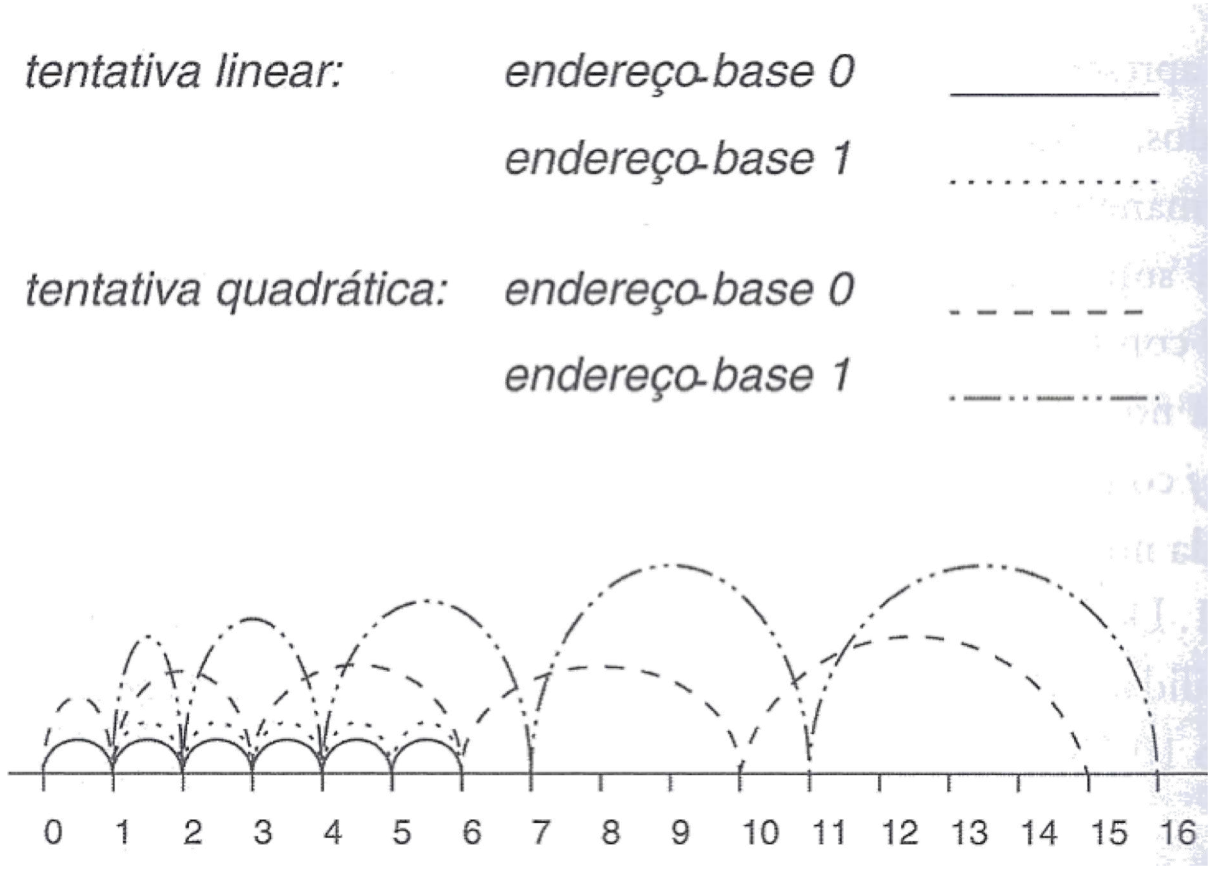
Valores de **m**, c_1 e c_2 precisam ser escolhidos de forma a garantir que todos os endereços-base serão percorridos

Exemplo:

- **h(x,0) = h'(x)**
- **h(x,k) = (h(x,k-1) + k) mod m**, para **0 < k < m**

- Essa função varre toda a tabela se **m** for potência de 2

TENTATIVA LINEAR X TENTATIVA QUADRÁTICA



FUNÇÃO HASH

Exemplos de funções hash p/ gerar sequência de tentativas

- Tentativa Linear
- Tentativa Quadrática
- **Dispersão Dupla**

DISPERSÃO DUPLA

Utiliza duas funções de hash, **h'(x)** e **h''(x)**

h(x,k) = (h'(x) + k.h''(x)) mod m, para **0 ≤ k < m**

Método distribui melhor as chaves do que os dois métodos anteriores

- Se duas chaves distintas **x** e **y** são sinônimas (**h'(x) = h'(y)**), os métodos anteriores produzem exatamente a mesma sequência de tentativas para **x** e **y**, ocasionando concentração de chaves em algumas áreas da tabela
- No método da dispersão dupla, isso só acontece se **h'(x) = h'(y)** e **h''(x) = h''(y)**

DISCUSSÃO

A técnica de hashing é mais utilizada nos casos em que existem muito mais buscas do que inserções de registros

EXERCÍCIO

1. Desenhe a tabela hash (em disco) resultante das seguintes operações (cumulativas) usando o algoritmo de inserção **Tabela Hash por Endereçamento Aberto**. A tabela tem tamanho 7.

(a) Inserir as chaves 10, 3, 5, 7, 12, 6, 14, 4, 8. Usar a função de tentativa linear $h(x, k) = (h'(x) + k) \bmod 7$, $0 \leq k \leq m-1$, e $h'(x) = x \bmod 7$

(b) Repita o exercício anterior, mas agora usando dispersão dupla $h(x,k) = (h'(x) + k.h''(x)) \bmod 7$, sendo $h'(x) = x \bmod 7$ e $h''(x) = x + 1$

REFERÊNCIA

Szwarcfiter, J.; Markezon, L. Estruturas de Dados e seus Algoritmos, 3a. ed. LTC. Cap. 10