

TABELAS HASH

Profa. Taiane C. Ramos
Estruturas de Dados e Seus
Algoritmos
Turma de Verão 2023

MOTIVAÇÃO

Alternativas para acelerar buscas em grandes volumes de dados:

Busca em um hash “ideal” sem colisões se dá em 1 passo.

□ $O(1)$ → **Tabelas Hash**

EXEMPLO DIDÁTICO

Funciona como se fosse um escaninho compartilhado.

Você recebe correspondência no escaninho com a primeira letra do seu nome.

Mais de uma pessoa pode usar o mesmo escaninho.



HASHING: FUNCIONAMENTO

n - quantidade de chaves
m - comprimento da tabela
[0, m-1] endereços possíveis.

Cada compartimento da tabela pode armazenar 1 ou mais registros dependendo da implementação.

COMO DETERMINAR M ?

A tabela hash pode ser armazenada em disco, então, teoricamente, poderíamos usar o próprio valor da chave pra endereçar a tabela.

Mas é uma boa ideia?

HASHING: PRINCÍPIO DE FUNCIONAMENTO

Exemplo de endereçamento de hash usando diretamente o valor da chave.

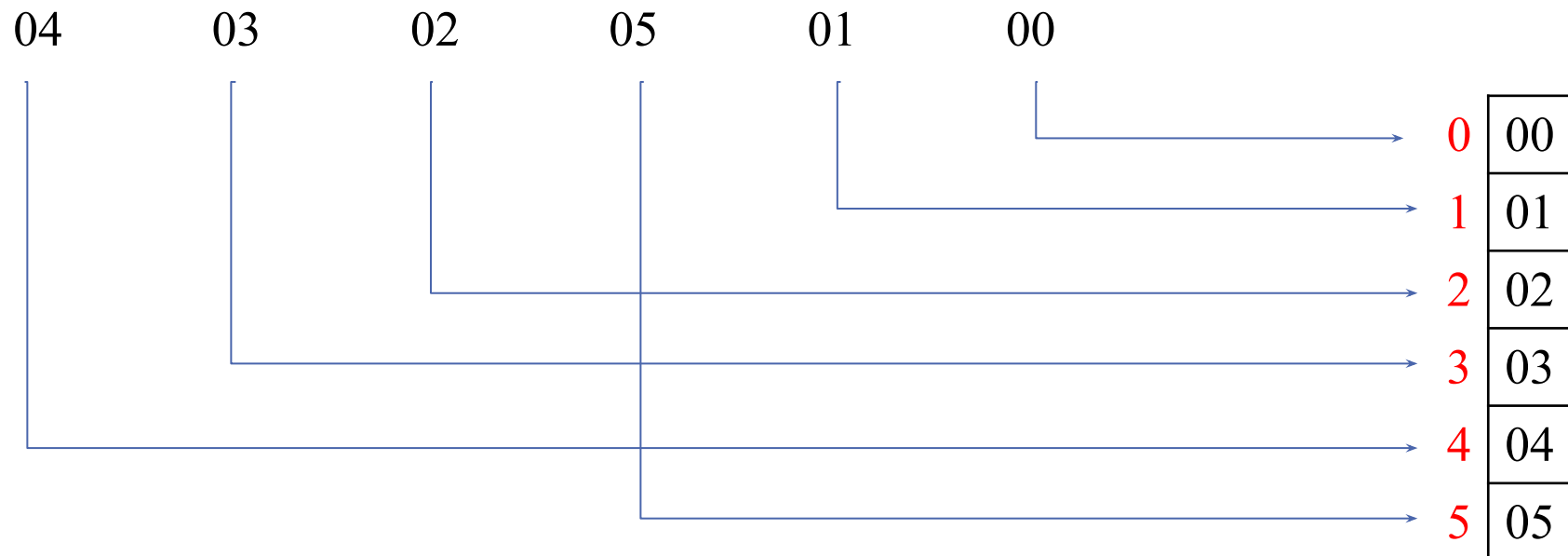
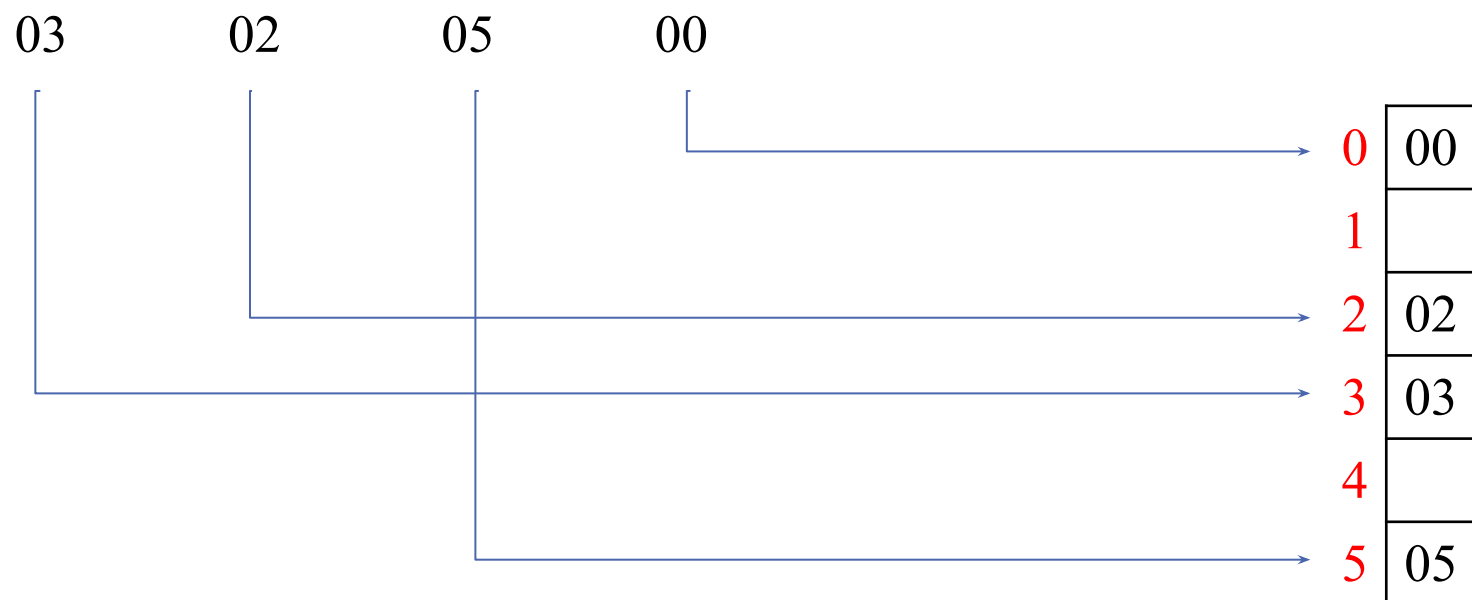


TABELA PODE TER ESPAÇOS VAZIOS

Mas e se temos valores esparsos de chave?



MAS...

Se armazenarmos 2 registros com chaves 0 e 999.999

□ tabela teria 999.998 compartimentos vazios

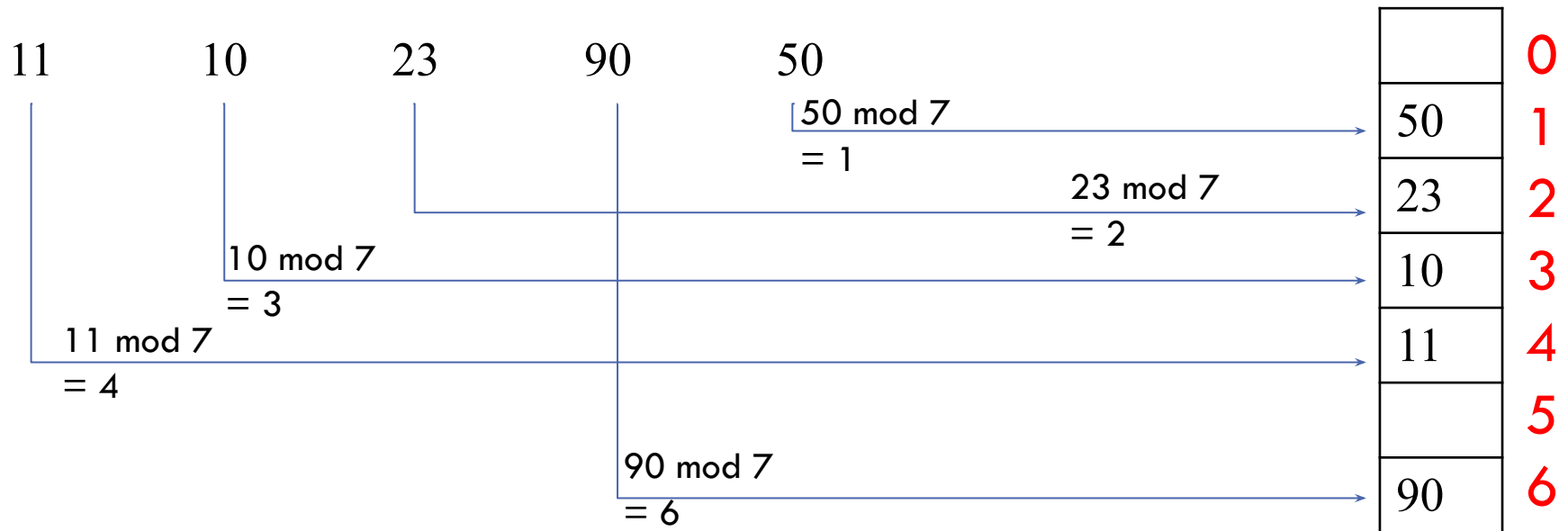
SOLUÇÃO

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

EXEMPLO

$$h(x) = x \bmod 7$$



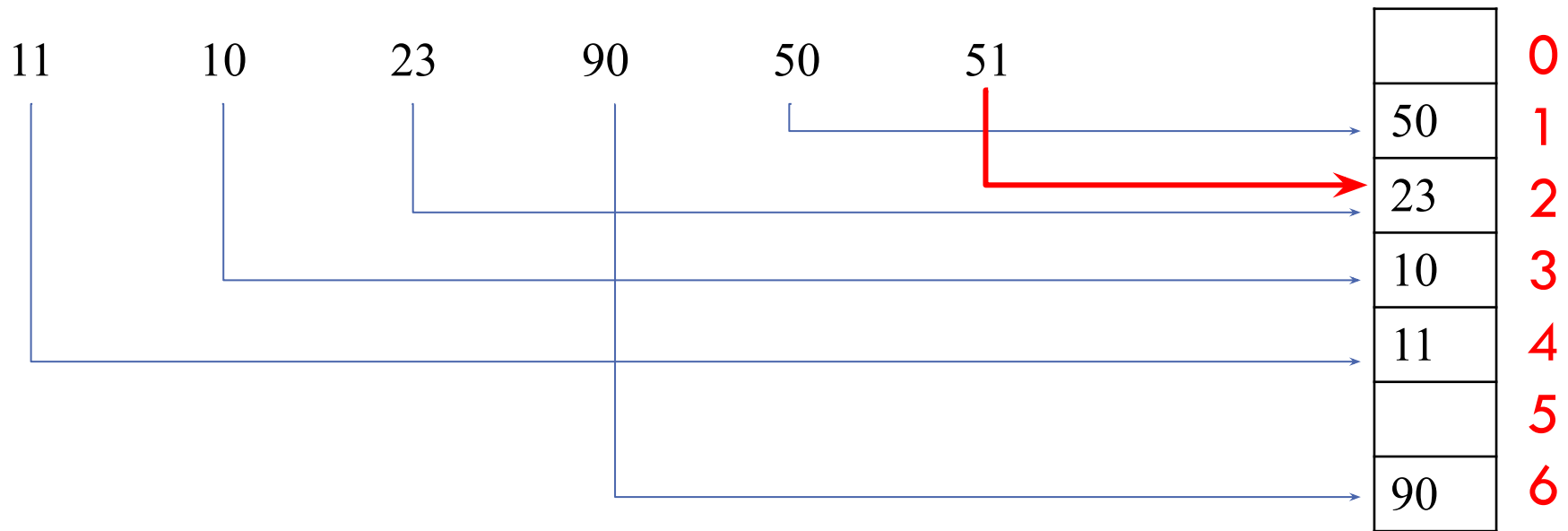
FUNÇÃO HASH H

Pode ser que duas chaves resultem no mesmo valor de hash
 $x \neq y$ e $h(x) = h(y)$

Colisão: Quando o endereço obtido por $h(x)$ já está ocupado.

EXEMPLO: COLISÃO

$$h(x) = x \bmod 7$$



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

CARACTERÍSTICAS DESEJÁVEIS DAS FUNÇÕES DE HASH

Produzir um número baixo de colisões

Ser facilmente computável

Ser uniforme (mesma chance de produzir todos os endereços disponíveis).

EXEMPLOS DE FUNÇÕES DE HASH

Exemplos de funções de hash bastante usadas:

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:

$$h(x) = x \bmod m$$

onde m é a dimensão da tabela

MÉTODO DA DIVISÃO

Possíveis bons valores de m (literatura):

- Número primo não próximo a uma potência de 2; ou
- Um número que não possui divisores primos menores do que 20

EXEMPLOS DE FUNÇÕES DE HASH

Método da
Divisão

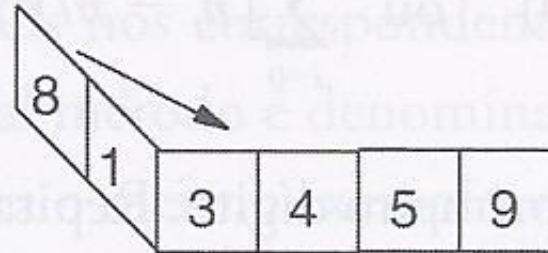
Método da
Dobra

Método da
Multiplicação

EXEMPLO: MÉTODO DA DOBRA

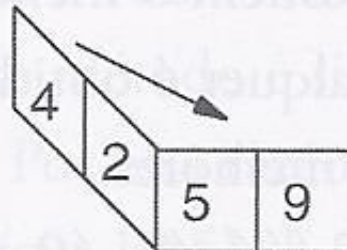
Chave

8	1	3	4	5	9
---	---	---	---	---	---



$$8+4=12$$

$$1+3=4$$



$$4+9=13$$

$$2+5=7$$

Dobras do tamanho que você quer o endereço final.

7	3
---	---

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 bits até obter o tamanho de endereço desejado

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

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

0	0	1	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

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

0	0	1	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---

MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

- $12 \times 12 = 144$
- 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

- $12 \times 12 = 144$
- 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

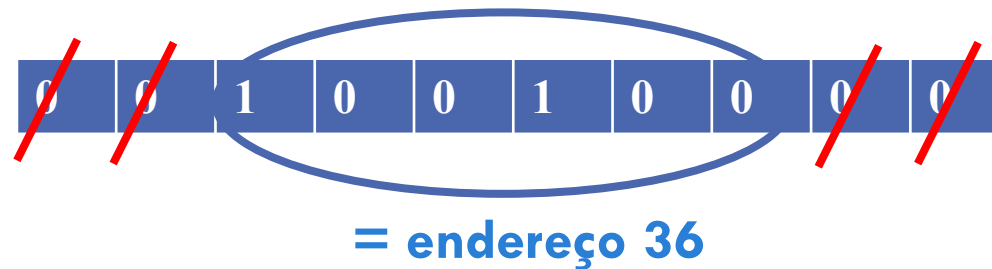
- $12 \times 12 = 144$
- 144 representado em binário: 10010000
- Armazenar em 10 bits: 0010010000
- Obter endereço de 6 bits (endereços entre 0 e 63)



MÉTODO DA MULTIPLICAÇÃO

Exemplo: chave 12

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



BUSCA

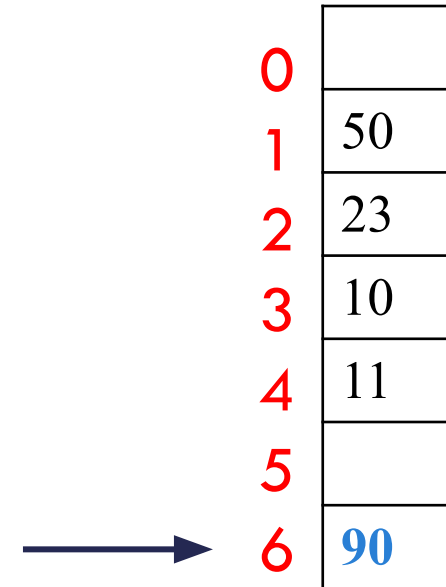
Para fazer a busca usamos a mesma função de hash.

EXEMPLO: BUSCA DE REGISTRO POR CHAVE

$$h(x) = x \bmod 7$$

chave 90

$$\square 90 \bmod 7 = 6$$




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

EXEMPLO: BUSCA DE REGISTRO POR CHAVE

$$h(x) = x \bmod 7$$

chave 7

$$7 \bmod 7 = 0$$



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

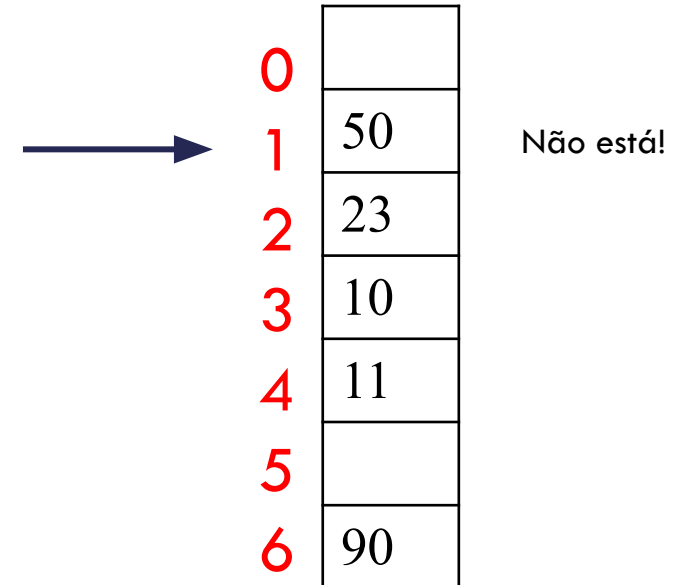
EXEMPLO: BUSCA DE REGISTRO POR CHAVE

$$h(x) = x \bmod 7$$

Encontrar o registro de chave 8

$$\square 8 \bmod 7 = 1$$

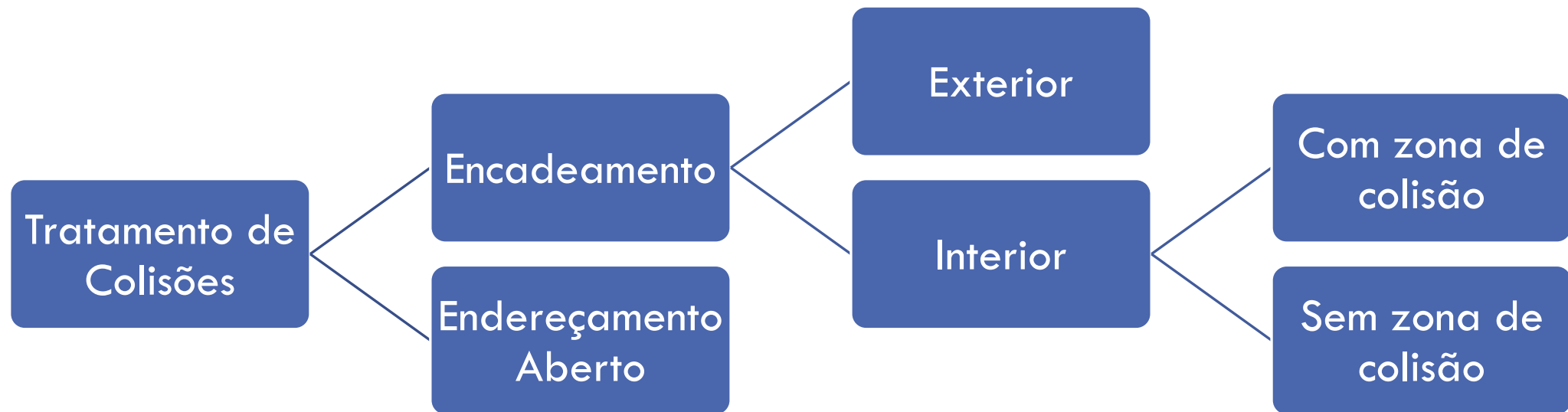
\square



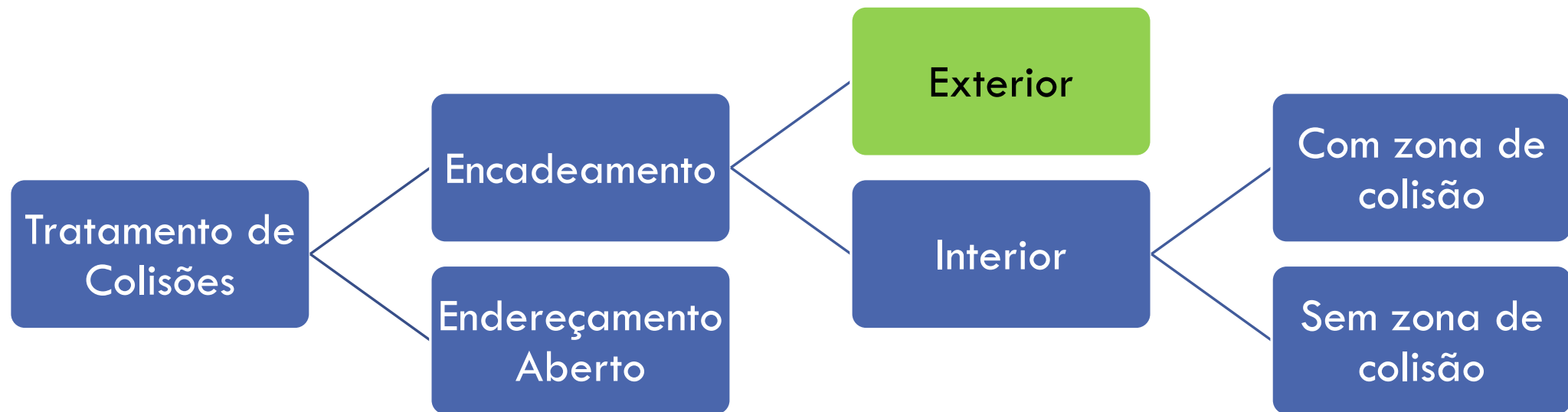
0	
1	50
2	23
3	10
4	11
5	
6	90

Não está!

TIPOS DE TRATAMENTO DE COLISÕES



TIPOS DE TRATAMENTO DE COLISÕES



ENCADEAMENTO EXTERIOR

Cada endereço da tabela hash contém uma lista encadeada.

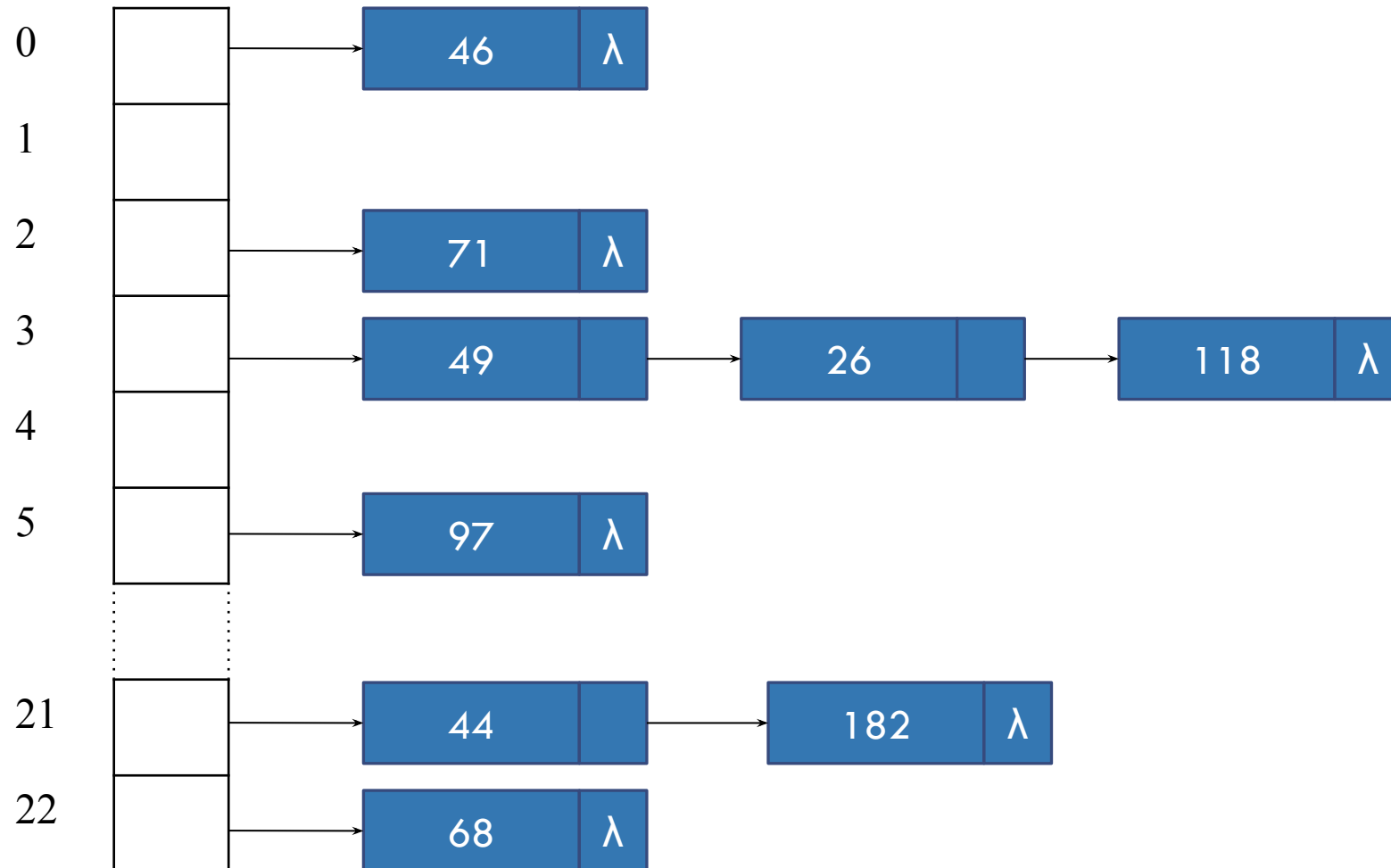
NÓS DA LISTA ENCADEADA

Cada nó da lista contém:

- um registro
- um ponteiro para o próximo nó

EXEMPLO: ENCADEAMENTO EXTERIOR

$$h(x) = x \bmod 23$$



BUSCA

Busca por um registro de chave x:

1. Calcular o endereço base
2. Percorrer a lista encadeada
3. Comparar sequencialmente as chaves até encontrar
4. Se final da lista for atingido, registro não está lá

INSERÇÃO

Inserção de um registro de chave x

1. Calcular o endereço base
2. Buscar registro na lista
3. Se registro for encontrado, não insere
4. Se o registro não for encontrado, inserir no final da lista

EXCLUSÃO

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

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

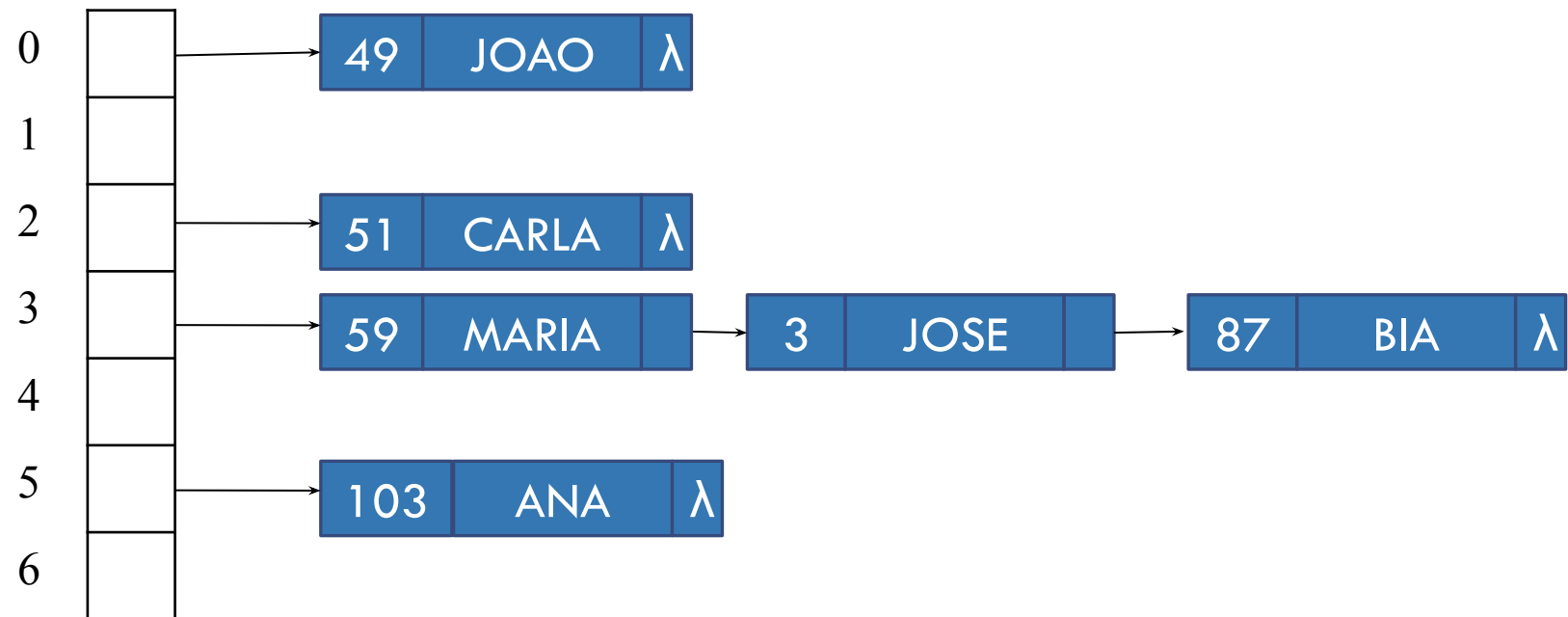
USO DE FLAG INDICADOR DE STATUS

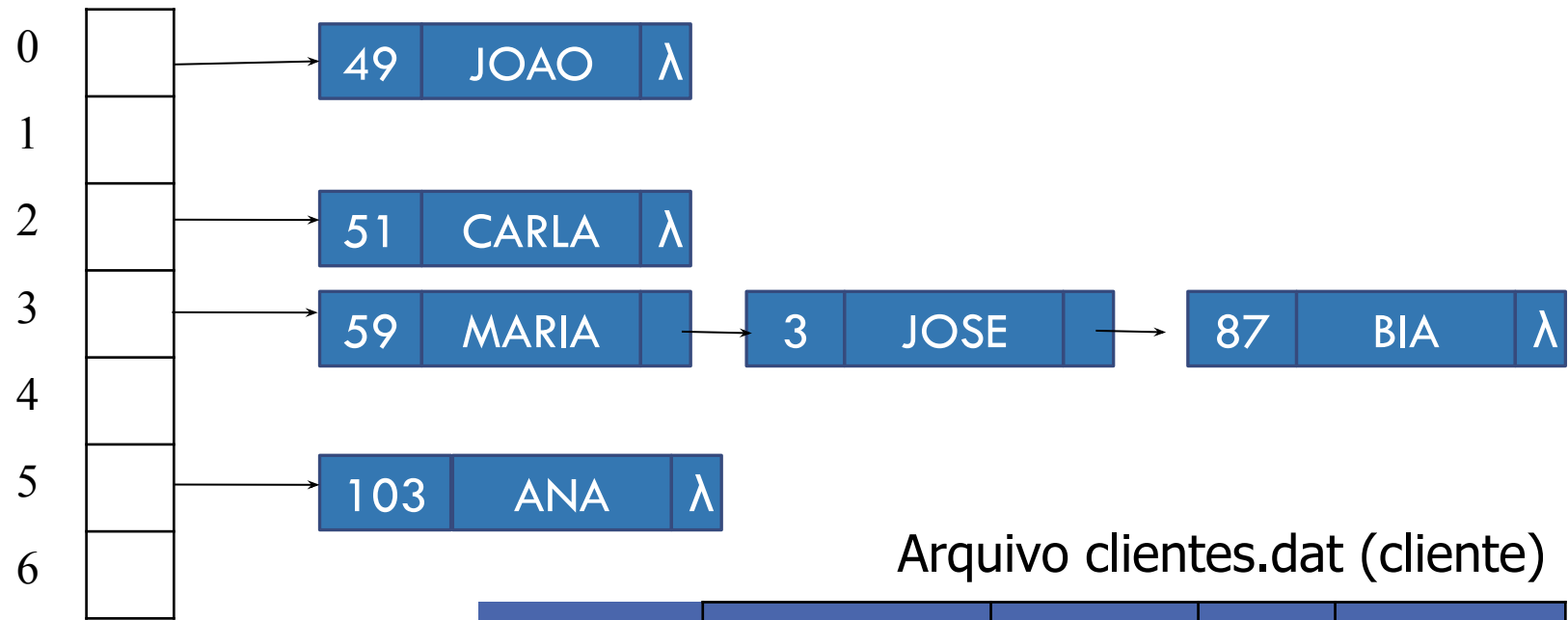
Para facilitar a manutenção da lista encadeada, pode-se adicionar um flag indicador de **status** a cada registro (chamaremos esse flag de **ocupado**)

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

EXEMPLO





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				

Arquivo tabHash.dat
(compartimento_hash)

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

$m = 7$

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**, ou no final da lista, caso não tenha sido encontrado compartimento livre

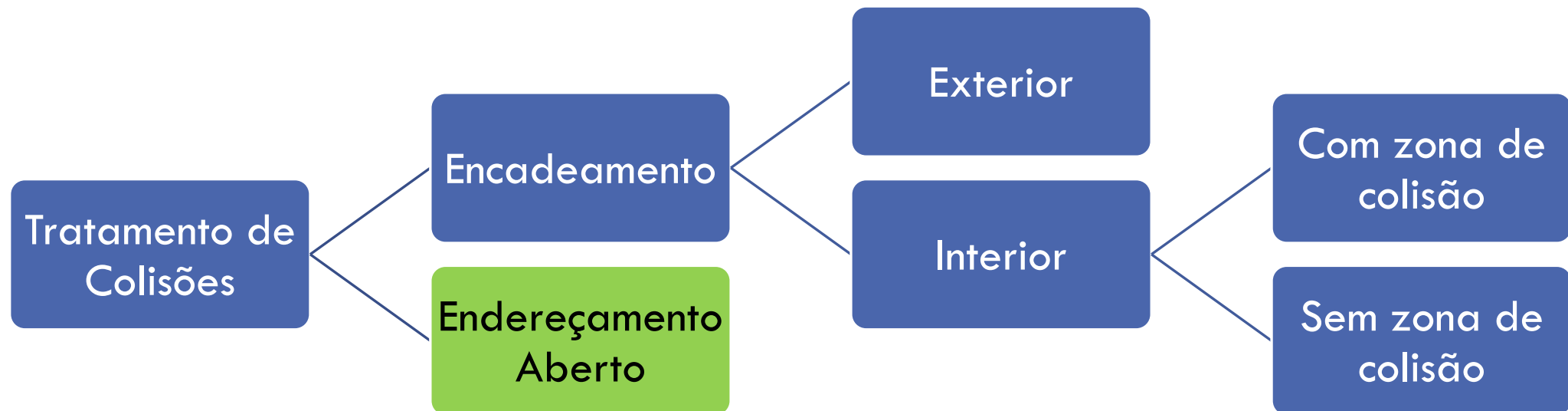
EXERCÍCIO

Desenhe a tabela hash (em disco) resultante das seguintes operações (cumulativas) usando o algoritmo de inserção em **Tabela Hash com Encadeamento Exterior**.

Considere que a tabela tem **tamanho 7** e a função de hash usa o **método da divisão**.

Inserir as chaves 10, 3, 5, 7, 12, 6, 14, 4, 8

TIPOS DE TRATAMENTO DE COLISÕES



TRATAMENTO DE COLISÕES POR ENDEREÇAMENTO ABERTO

Nesta abordagem, não precisamos armazenar ponteiros (menos consumo de espaço).

Quando houver colisão, “tentamos novamente”.

FUNIONAMENTO

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 $k = 0, \dots, m-1$

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 $h(x,0), h(x,1), \dots, h(x, m-1)$ é denominada **sequência de tentativas**

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

FUNÇÃO HASH

Exemplos de funções hash para gerar sequências de tentativas

- Tentativa Linear
- Tentativa Quadrática
- Dispersão Dupla

FUNÇÃO HASH

Exemplos de funções hash para gerar sequência de tentativas

- **Tentativa Linear**

- Tentativa Quadrática

- Dispersão Dupla

TENTATIVA LINEAR

Se o endereço obtido por $h(x)$ já estiver ocupado, tentamos o próximo.

EXEMPLO TENTATIVA LINEAR

$$h(x, k) = (h'(x) + k) \bmod m$$
$$h'(x) = x \bmod 23$$

44

$$h'(44) = 44 \bmod 23 = 21$$

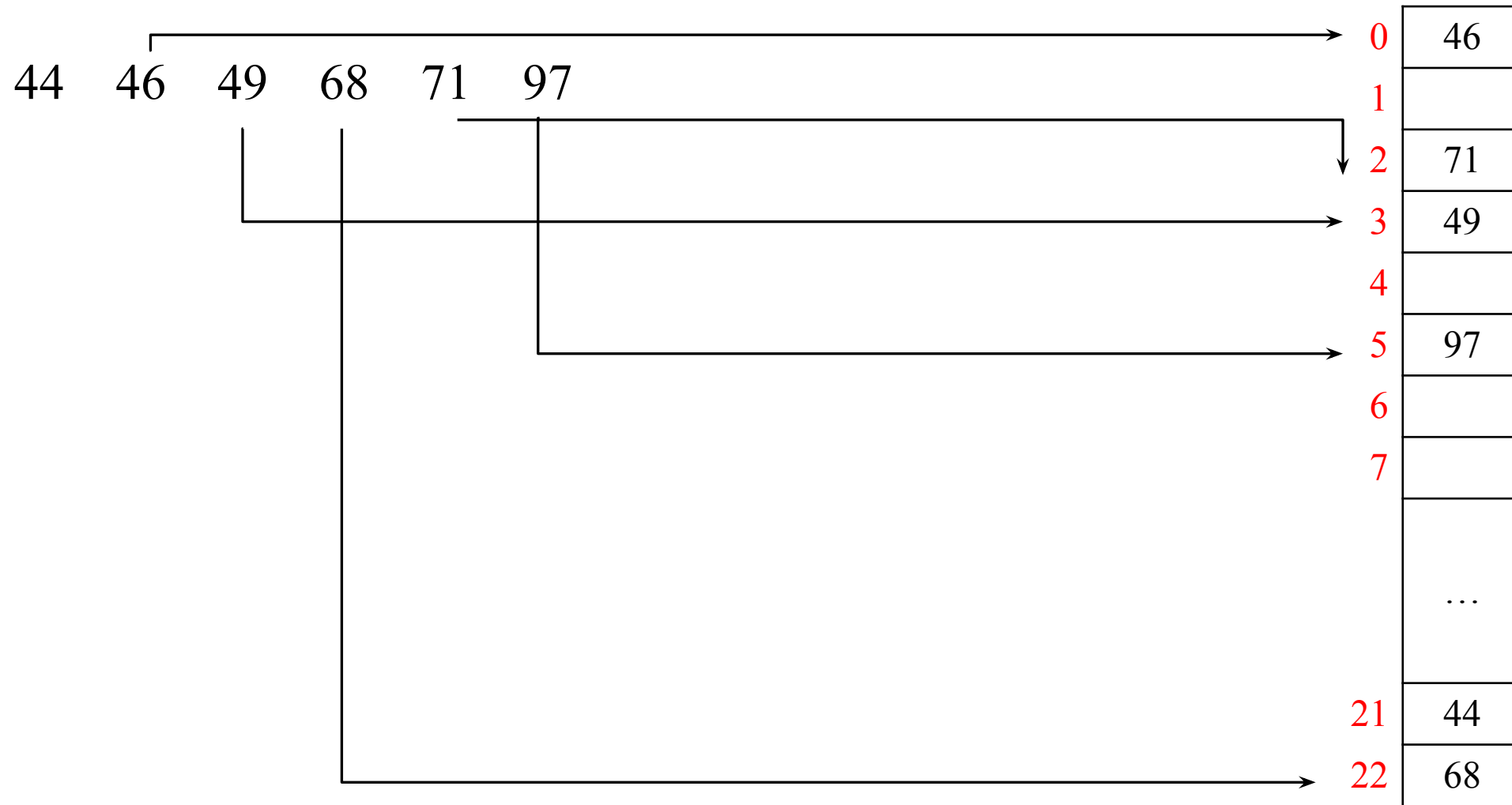
$$h(44, 0) = (21 + 0) \bmod 23$$

$$h(44, 0) = 21 \bmod 23 = 21$$

0	
1	
2	
3	
4	
5	
6	
7	
	...
21	44
22	

$$h(x, k) = (h'(x) + k) \bmod m$$
$$h'(x) = x \bmod 23$$

EXEMPLO TENTATIVA LINEAR



$$h(x, k) = (h'(x) + k) \bmod m$$

$$h'(x) = x \bmod 23$$

EXEMPLO TENTATIVA LINEAR

44 46 49 68 71 97 **26**



$$h'(26) = 26 \bmod 23 = 3$$

$$h(26, 0) = (3 + 0) \bmod 23 = 3 \text{ (OCUPADO)}$$

$$h(26, 1) = (3 + 1) \bmod 23 = 4$$

0	46
1	
2	71
3	49
4	26
5	97
6	72
7	27
	...
21	44
22	68

$$h(x, k) = (h'(x) + k) \bmod m$$

$$h'(x) = x \bmod 23$$

EXEMPLO TENTATIVA LINEAR

44 46 49 68 71 97 26 **72**

$$h'(72) = 72 \bmod 23 = 3$$

$$h(72, 0) = (3 + 0) \bmod 23 = 3 \text{ (OCUPADO)}$$

$$h(72, 1) = (3 + 1) \bmod 23 = 4 \text{ (OCUPADO)}$$

$$h(72, 2) = (3 + 2) \bmod 23 = 5 \text{ (OCUPADO)}$$

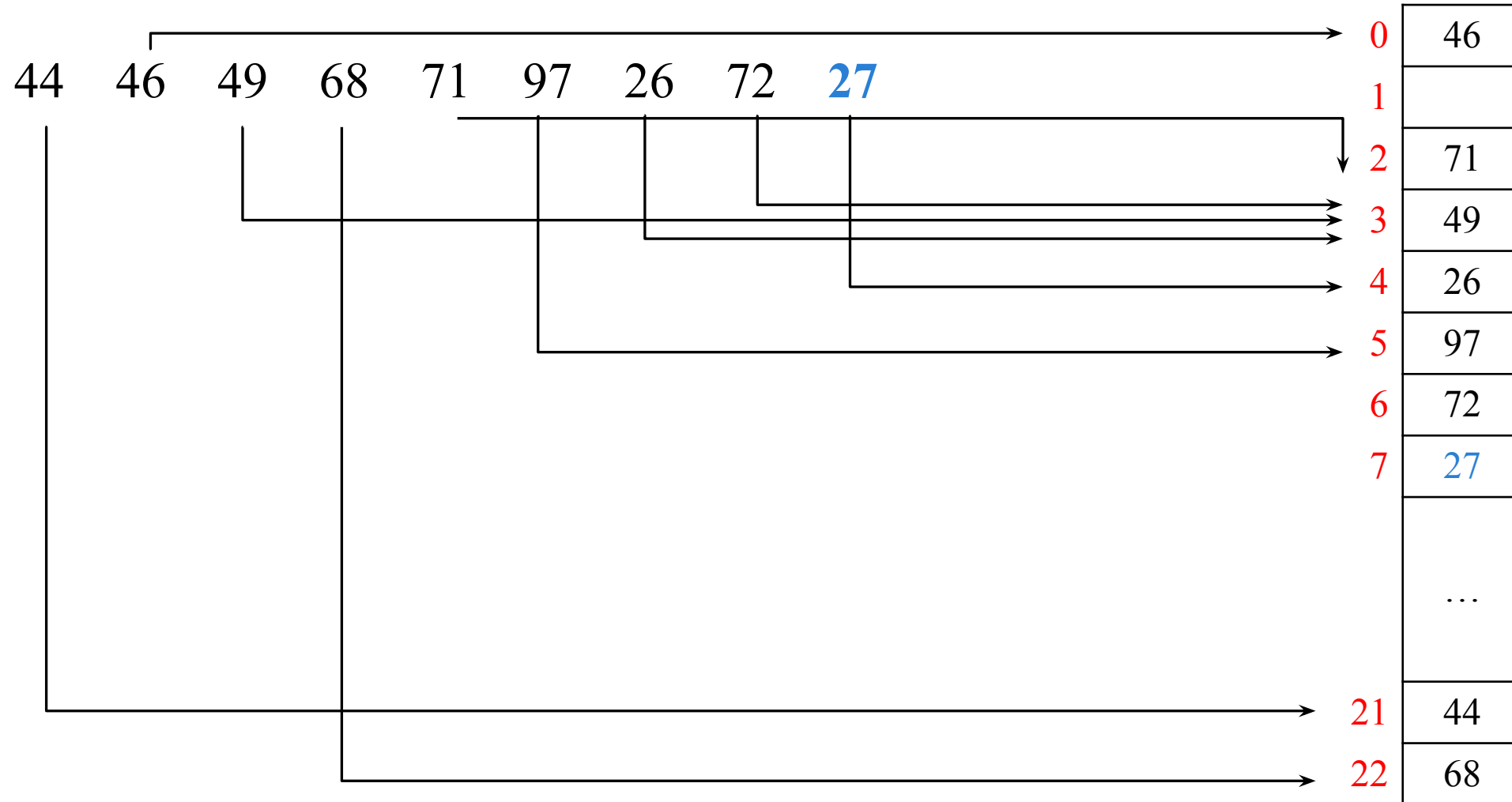
$$h(72, 3) = (3 + 3) \bmod 23 = 6$$

0	46
1	
2	71
3	49
4	26
5	97
6	72
7	
...	
21	44
22	68

$$h(x, k) = (h'(x) + k) \bmod m$$

$$h'(x) = x \bmod 23$$

EXEMPLO TENTATIVA LINEAR



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
                pos_livre = end;

                k = m; //força saída do loop
            }
            else k = k + 1; //continua procurando
        }
    }
    if (*achou)
        return end;
    else
        return pos_livre;
}

```

INSERÇÃO EM ENDEREÇAMENTO ABERTO

```
// Função 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 achou a chave, mas achou 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

O algoritmo de busca não prevê realização de exclusões, pois assume que chave não está na tabela quando encontra a primeira posição livre

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?

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 para gerar sequências 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**

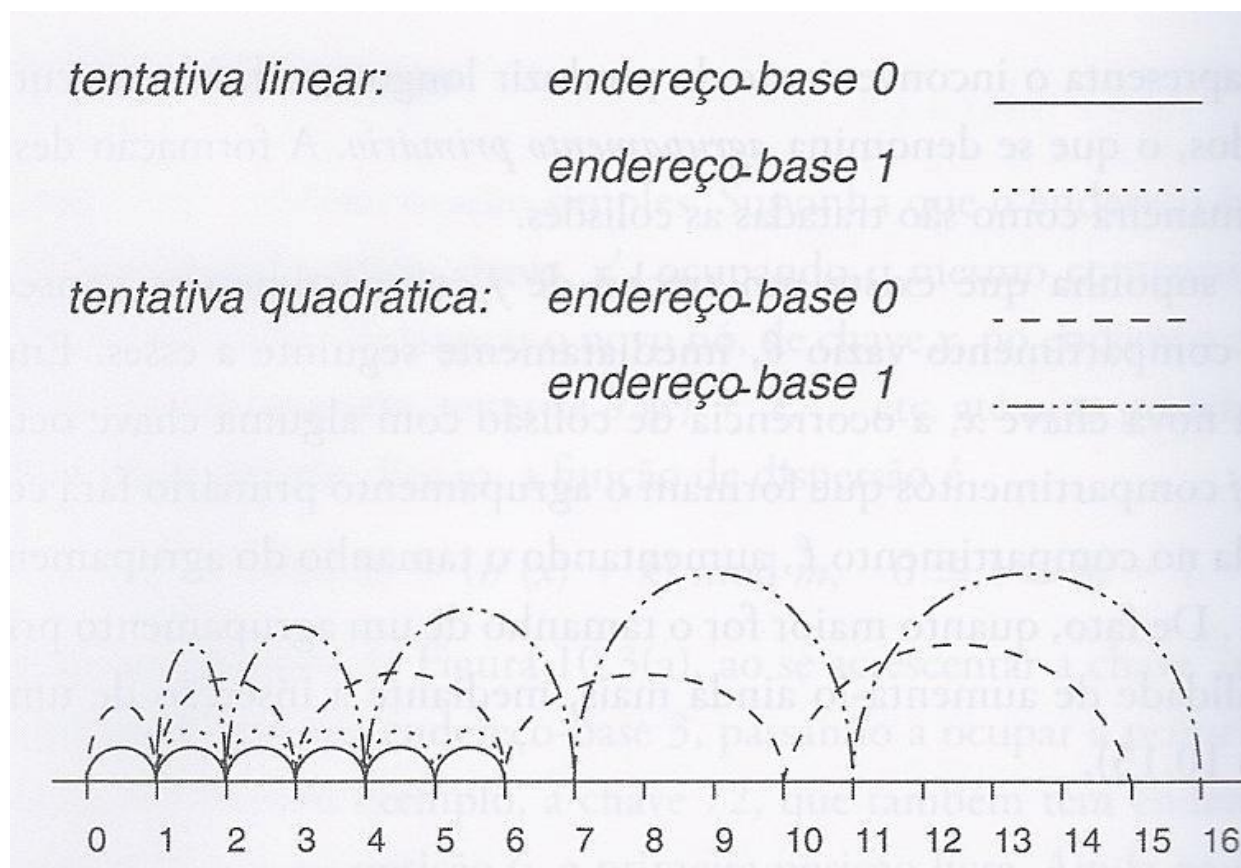
$$\square h(x,k) = (h'(x) + c_1 k + c_2 k^2) \bmod m,$$

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

TENTATIVA QUADRÁTICA

Método evita agrupamentos primários

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



FUNÇÃO HASH

Exemplos de funções hash para gerar sequência de tentativas

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

DISPERSÃO DUPLA

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

$$h(x,k) = (h'(x) + k \cdot h''(x)) \bmod m, \text{ para } 0 \leq k < m$$

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 $h(x, k) = (h'(x) + k \cdot h''(x)) \bmod 7$, sendo $h'(x) = x \bmod 7$ e $h''(x) = x - 1$