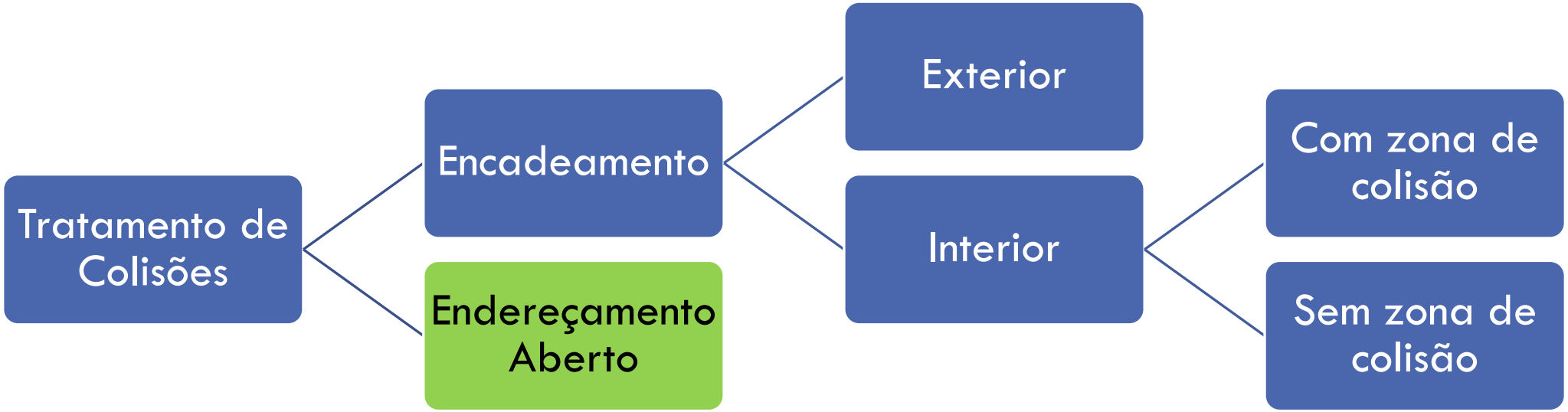
TABELAS HASH
TRATAMENTO DE COLISÕES POR ENDEREÇAMENTO ABERTO

Vanessa Braganholo

Estruturas de Dados e Seus Algoritmos

TIPOS DE TRATAMENTO DE COLISÕES



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

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

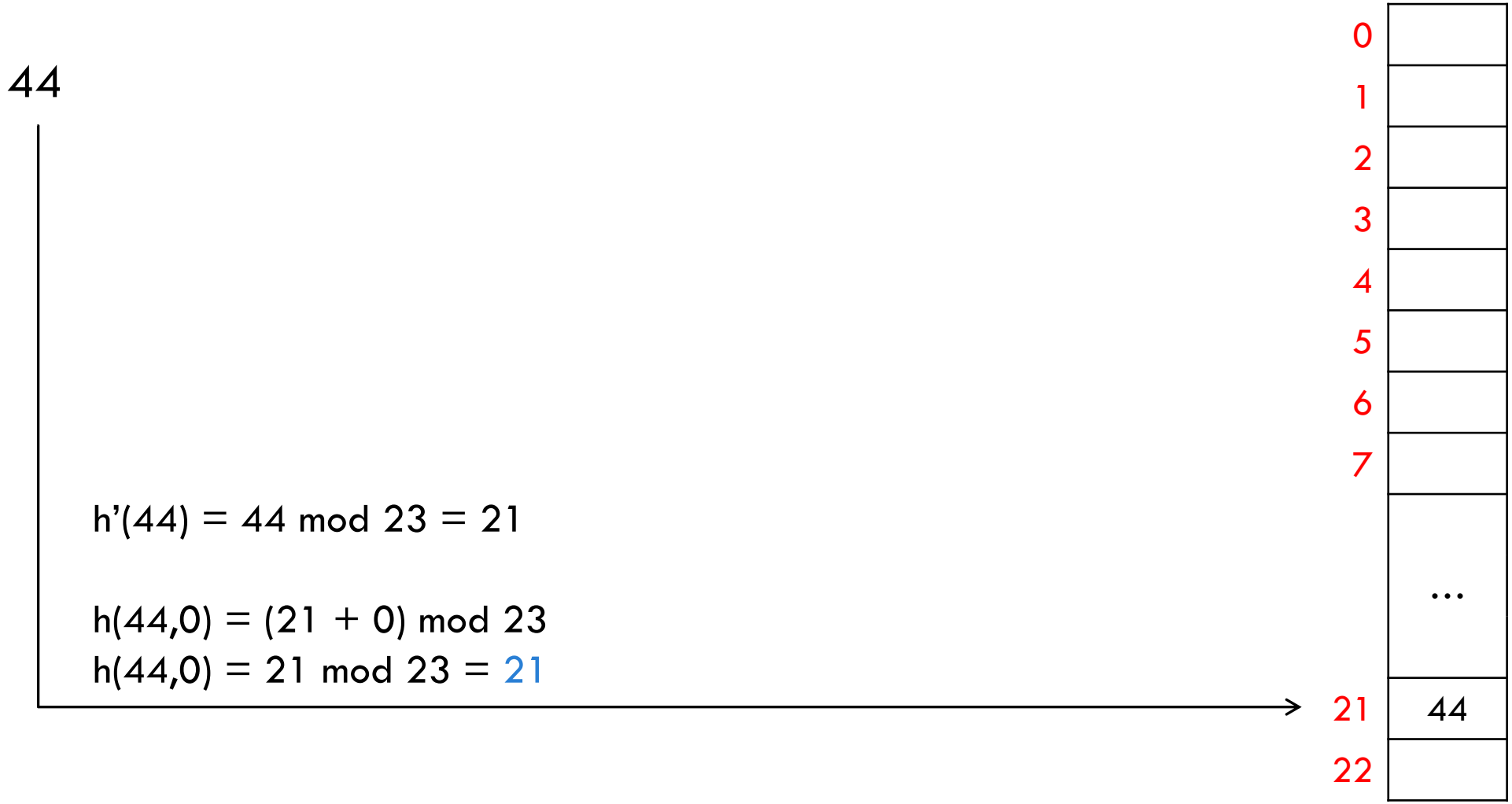
EXEMPLO TENTATIVA LINEAR

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

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

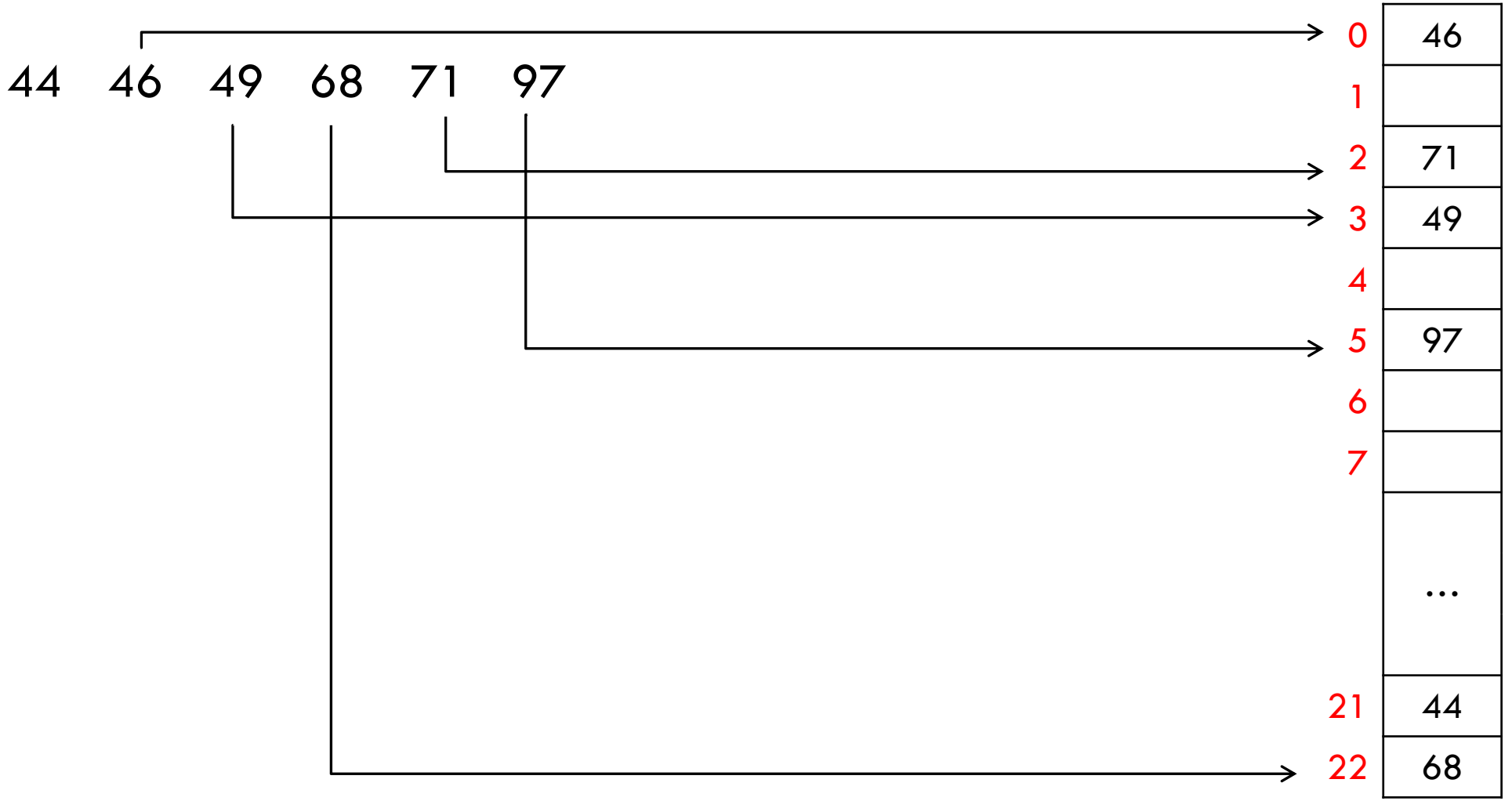
EXEMPLO TENTATIVA LINEAR

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



EXEMPLO TENTATIVA LINEAR

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



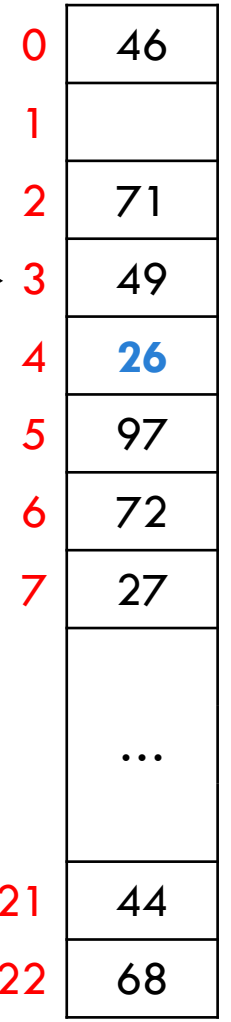
EXEMPLO TENTATIVA LINEAR

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

44 46 49 68 71 97 **26**

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

$h(26,0) = (3 + 0) \bmod 23 = 3$ (OCUPADO)
$h(26,1) = (3 + 1) \bmod 23 = 4$



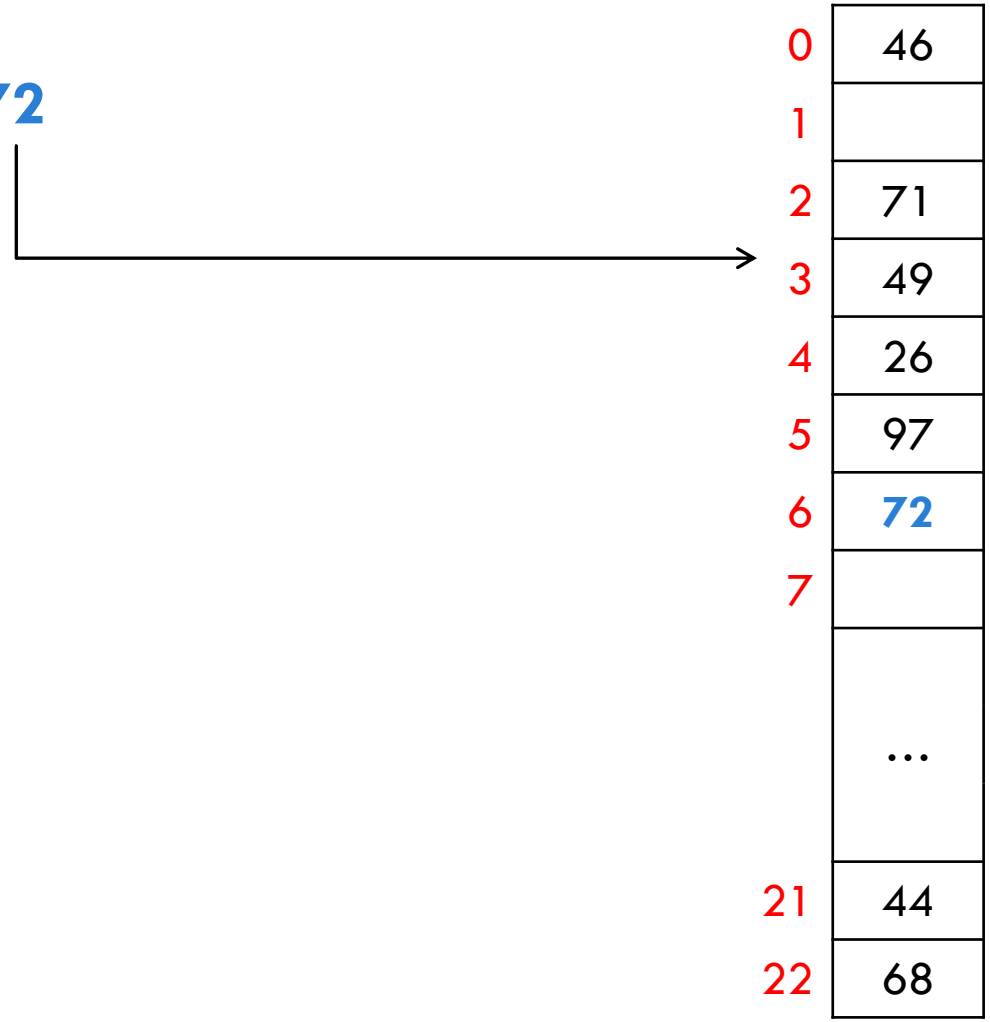
EXEMPLO TENTATIVA LINEAR

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

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

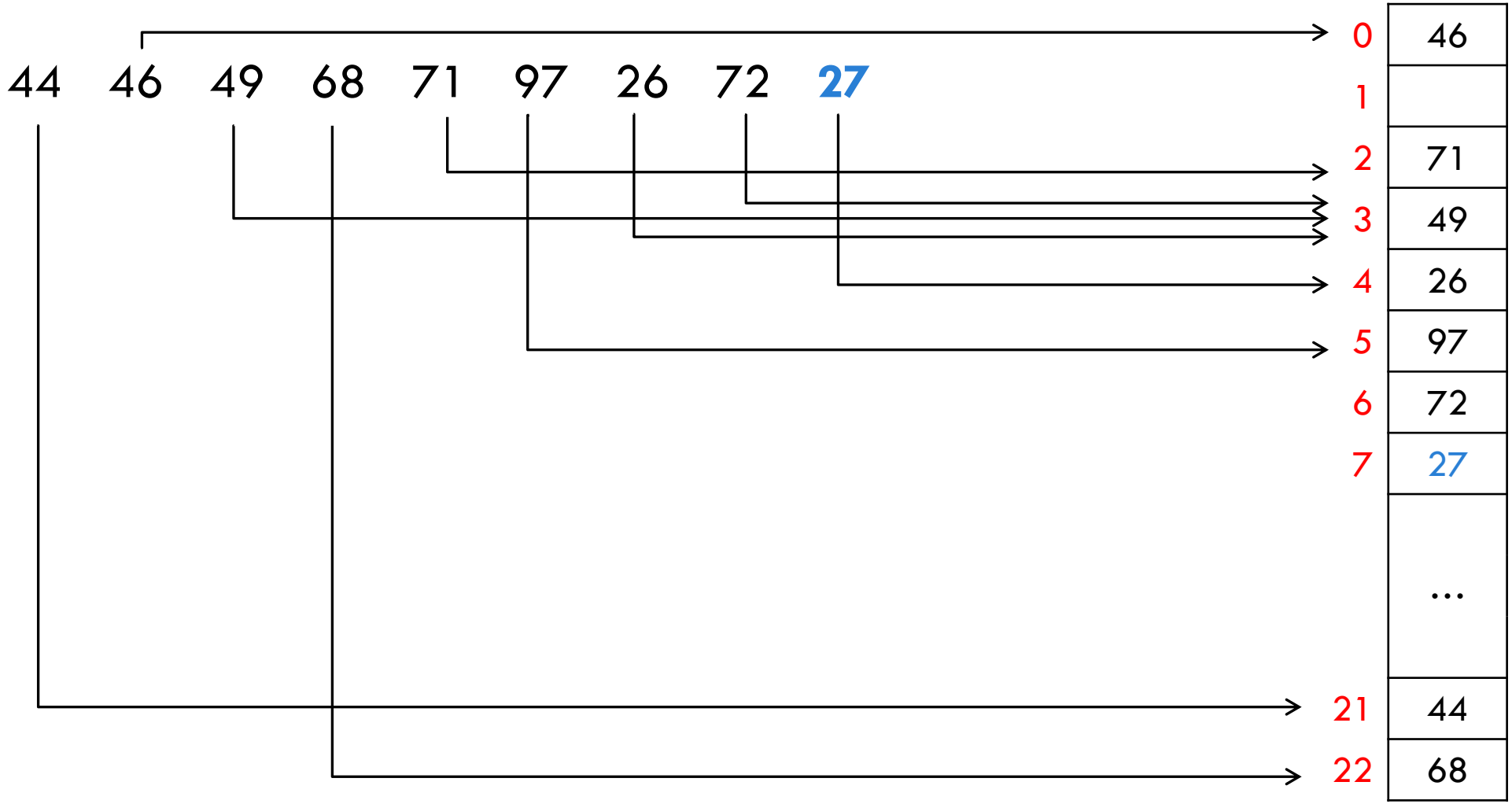
$h'(72) = 72 \bmod 23 = 3$

$h(72,0) = (3+ 0) \bmod 23 = 3$ (OCUPADO)
$h(72,1) = (3+ 1) \bmod 23 = 4$ (OCUPADO)
$h(72,2) = (3+ 2) \bmod 23 = 5$ (OCUPADO)
$h(72,3) = (3 + 3) \bmod 23 = 6$



EXEMPLO TENTATIVA LINEAR

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



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
                  pos_livre = end;

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

Fonte: Algoritmo 10.4, pag 247.

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?

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**

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

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

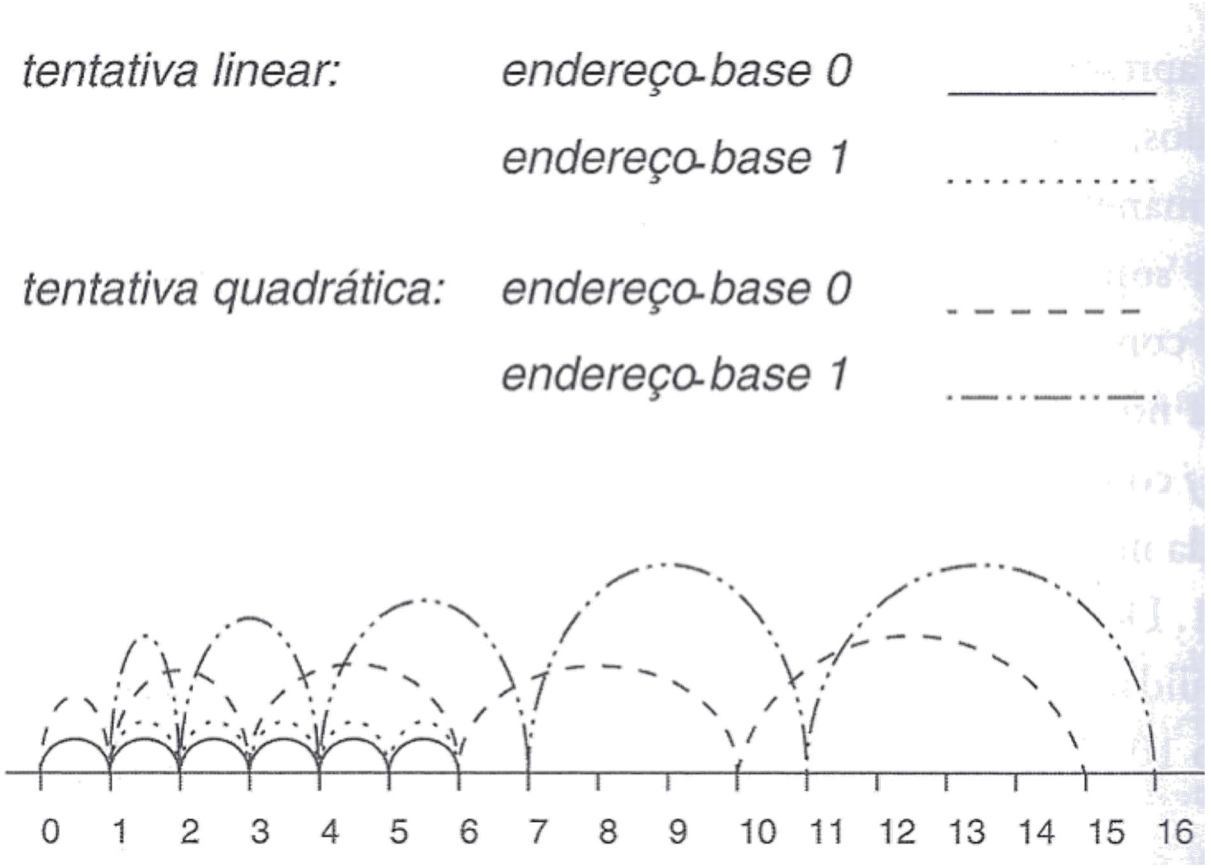
Valores de **m**, c_1 e c_2 precisam ser escolhidos de forma a garantir que todos os endereços-base serão percorridos

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