Tabelas de Dispersão

(Hashing)

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

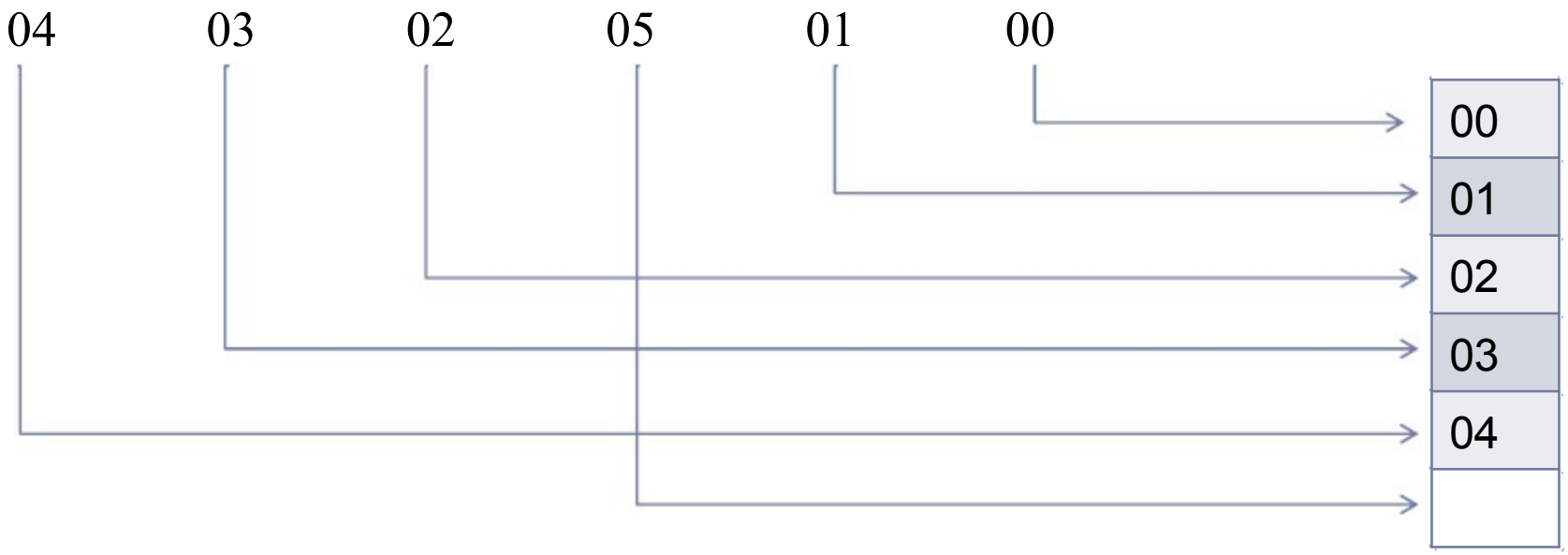
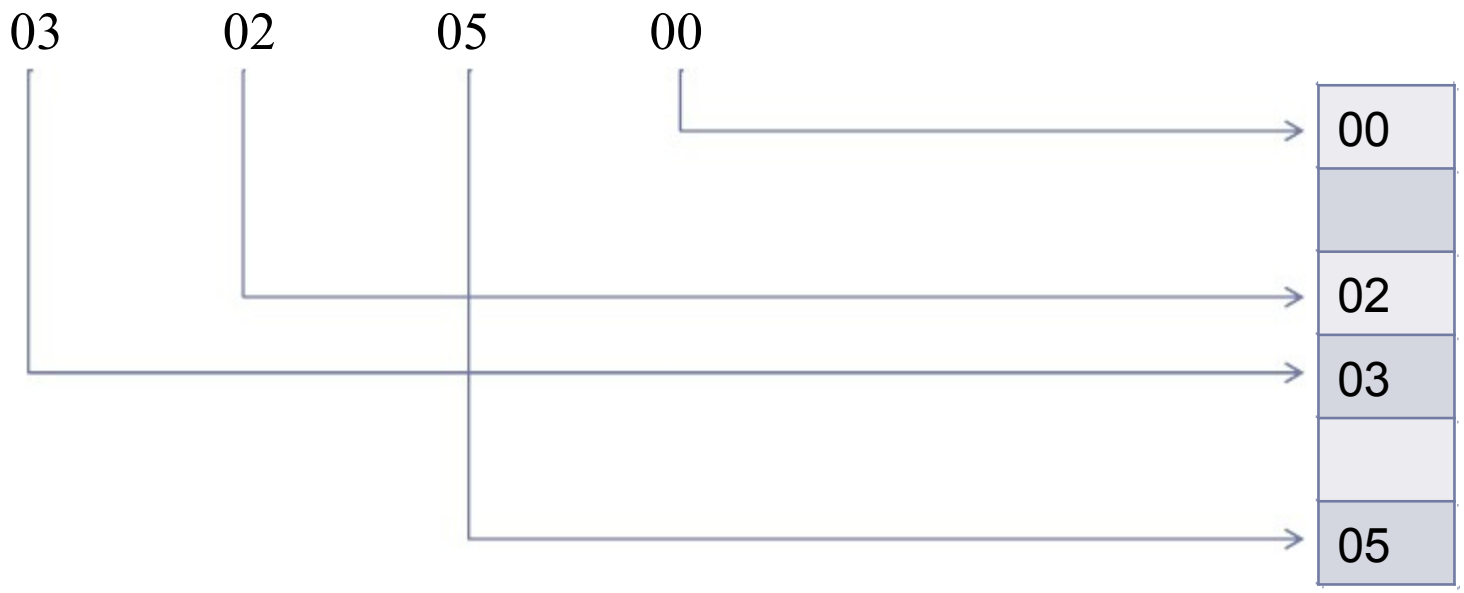


Tabela pode ter espaços vazios

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 houverem poucos registros

m = 1.000.000

tabela teria 999.998 compartimentos vazios

Exemplo: armazenar 2 registros com chaves 0 e 999.999 respectivamente

Solução

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

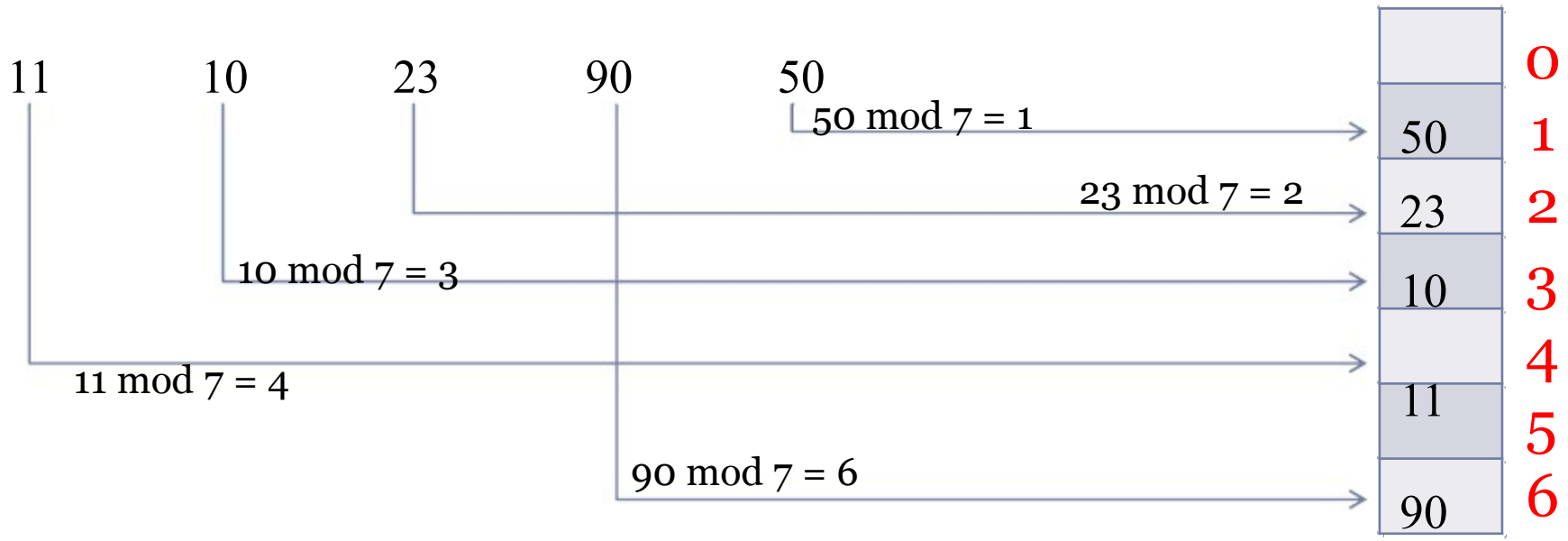
Usar uma função hash **h** que mapeia um valor de chave **x** para um endereço da tabela

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

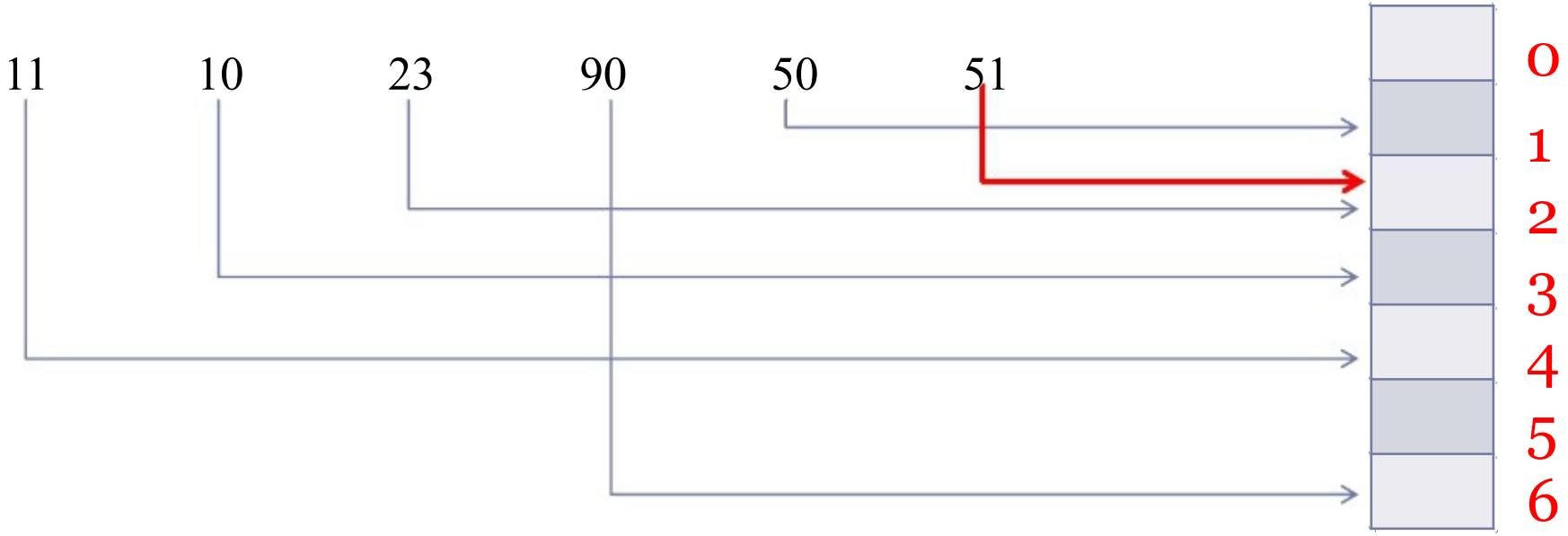
Infelizmente, a função pode não garantir injetividade, ou seja, é possível que **x ≠ y** e **h(x) = h(y)**

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



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 como parte da chave o ano e mês do pedido.

Se a função **h** realçar estes dados, haverá muita concentração de valores nas mesmas faixas.

Características desejáveis das funções de hash

Ser facilmente computável

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

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

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

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

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

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

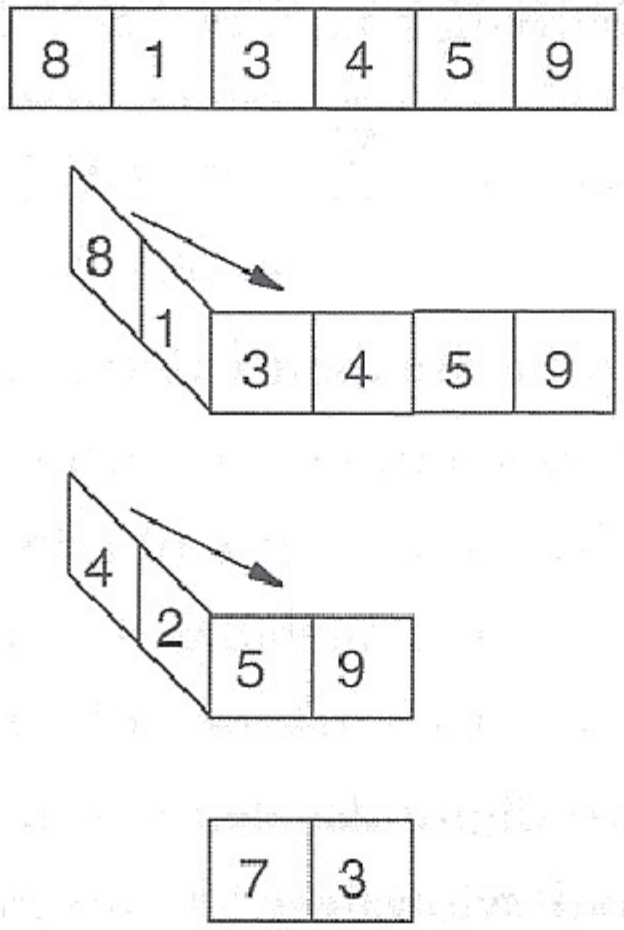
Método da Dobra

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



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

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**

Método da Multiplicação

Multiplicar a chave por ela mesma

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

Descartar os bits das extremidadades 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

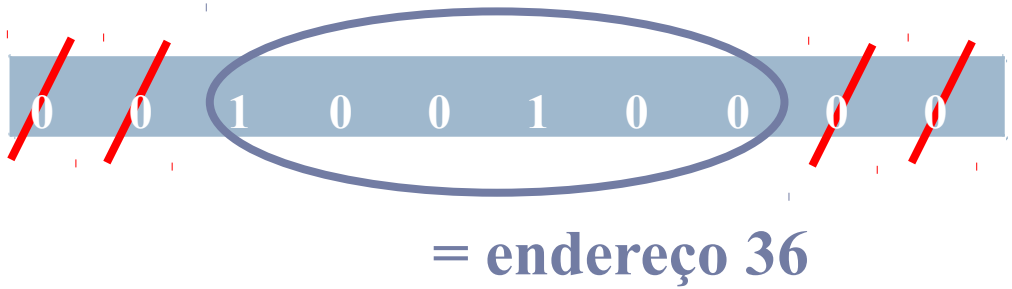
Exemplo: chave 12

12 x 12 = 144

144 representado em binário: 10010000

Armazenar em 10 bits: 0010010000

Obter endereço de 6 bits (endereços entre 0 e 63)



Uso da função de hash

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

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

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

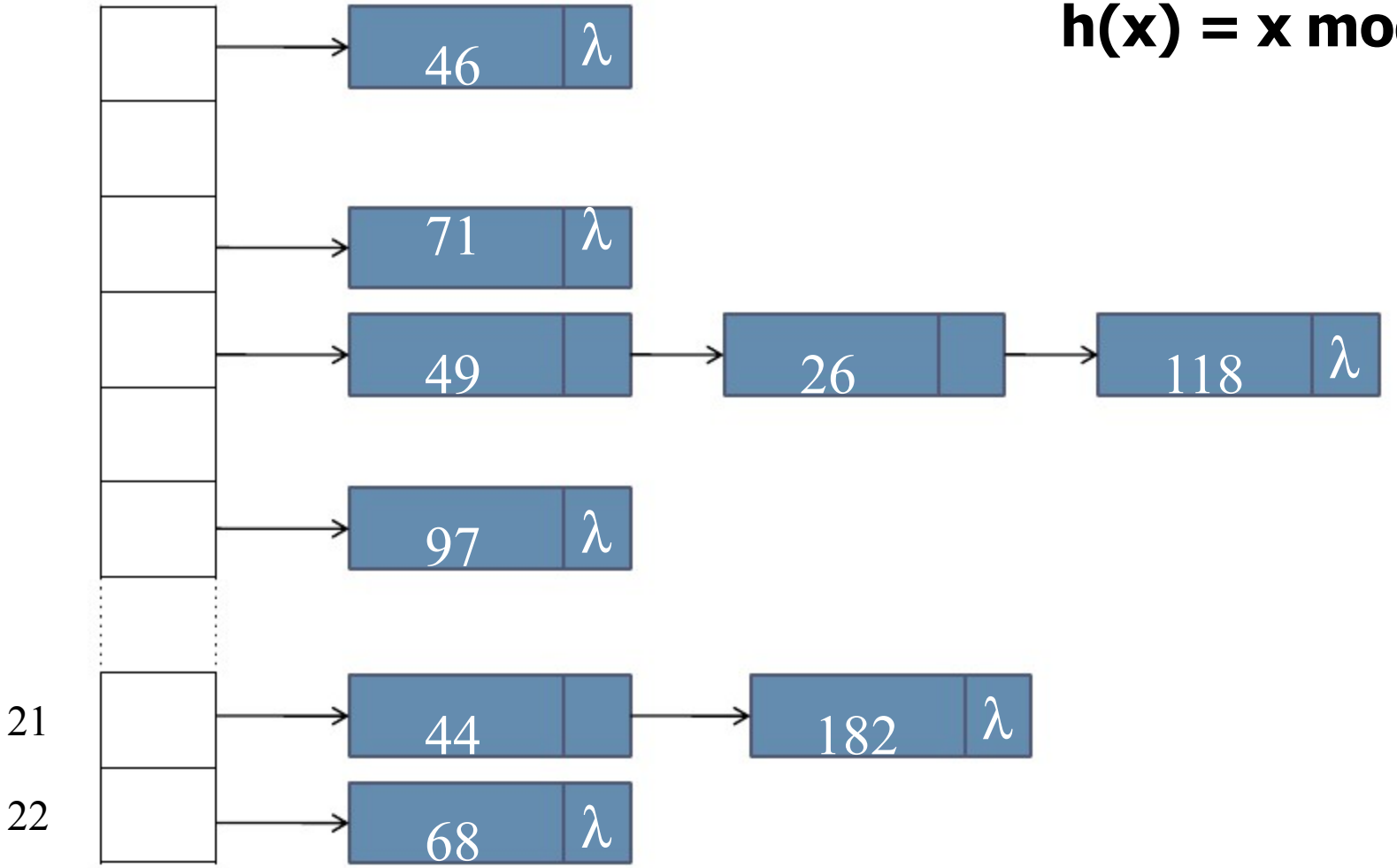
Cada nó da lista encadeada contém:

um registro

um ponteiro para o próximo nó

Exemplo: Encadeamento Exterior

h(x) = x mod 23



Encadeamento Exterior

Busca por um registro de chave x:

Calcular o endereço aplicando a função h(x)

Percorrer a lista encadeada associada ao endereço

Comparar a chave de cada nó da lista encadeada com a chave x, até encontrar o nó desejado

Se final da lista for atingido, registro não está lá

Encadeamento Exterior

Inserção de um registro de chave x

Calcular o endereço aplicando a função h(x)

Buscar registro na lista associada ao endereço h(x)

Se registro for encontrado, sinalizar erro

Se o registro não for encontrado, inserir no final da lista

Encadeamento Exterior

Exclusão de um registro de chave x

Calcular o endereço aplicando a função h(x)

Buscar registro na lista associada ao endereço h(x)

Se registro for encontrado, excluir registro

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!

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**

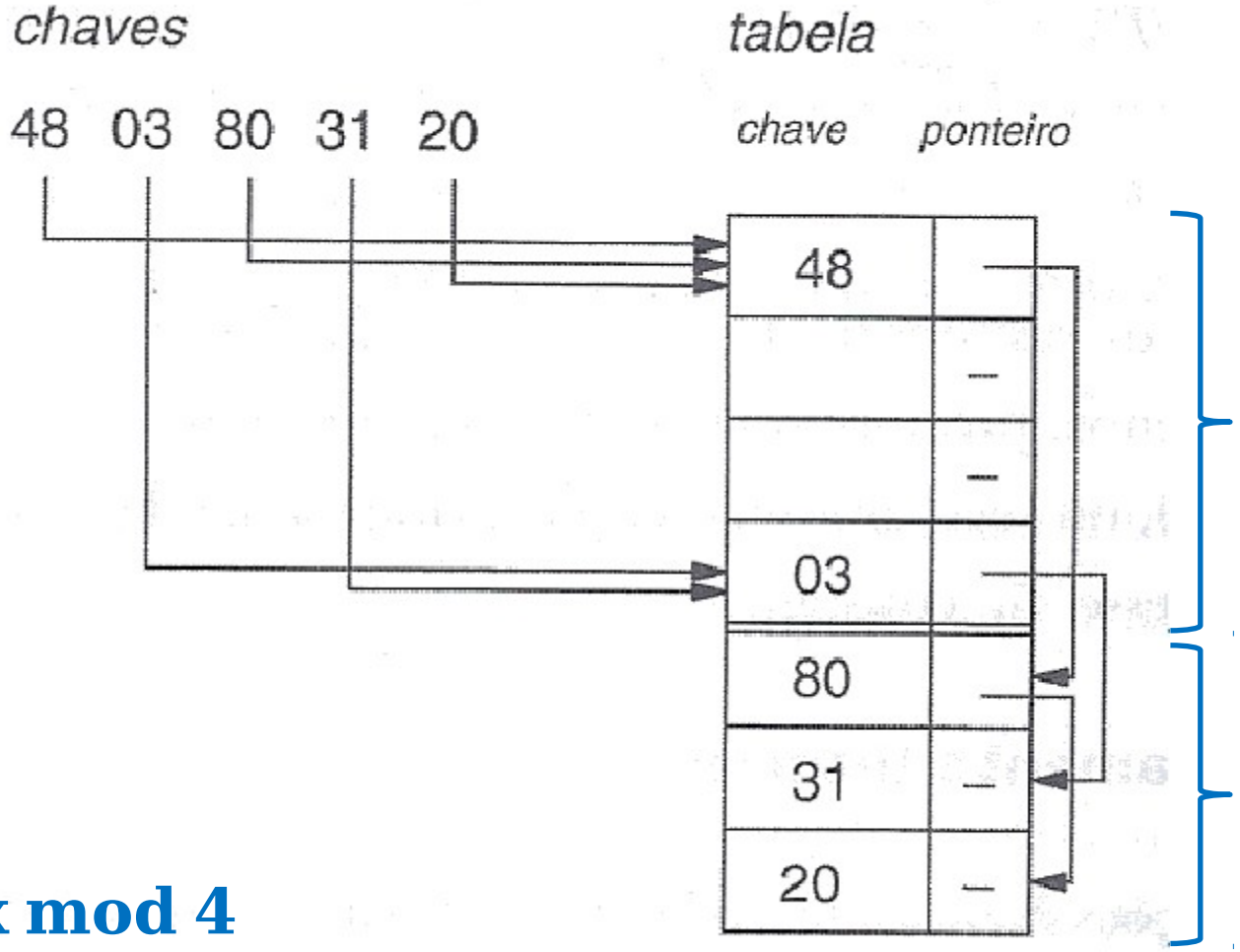
Uma de colisão, de tamanho **s**

s+p=m

Função de hash deve gerar endereços no intervalo **[0, p-1]**

Cada nó tem a mesma estrutura utilizada no Encadeamento Exterior

Exemplo: Encadeamento Interior com Zona de Colisões



h(x) = x mod 4

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

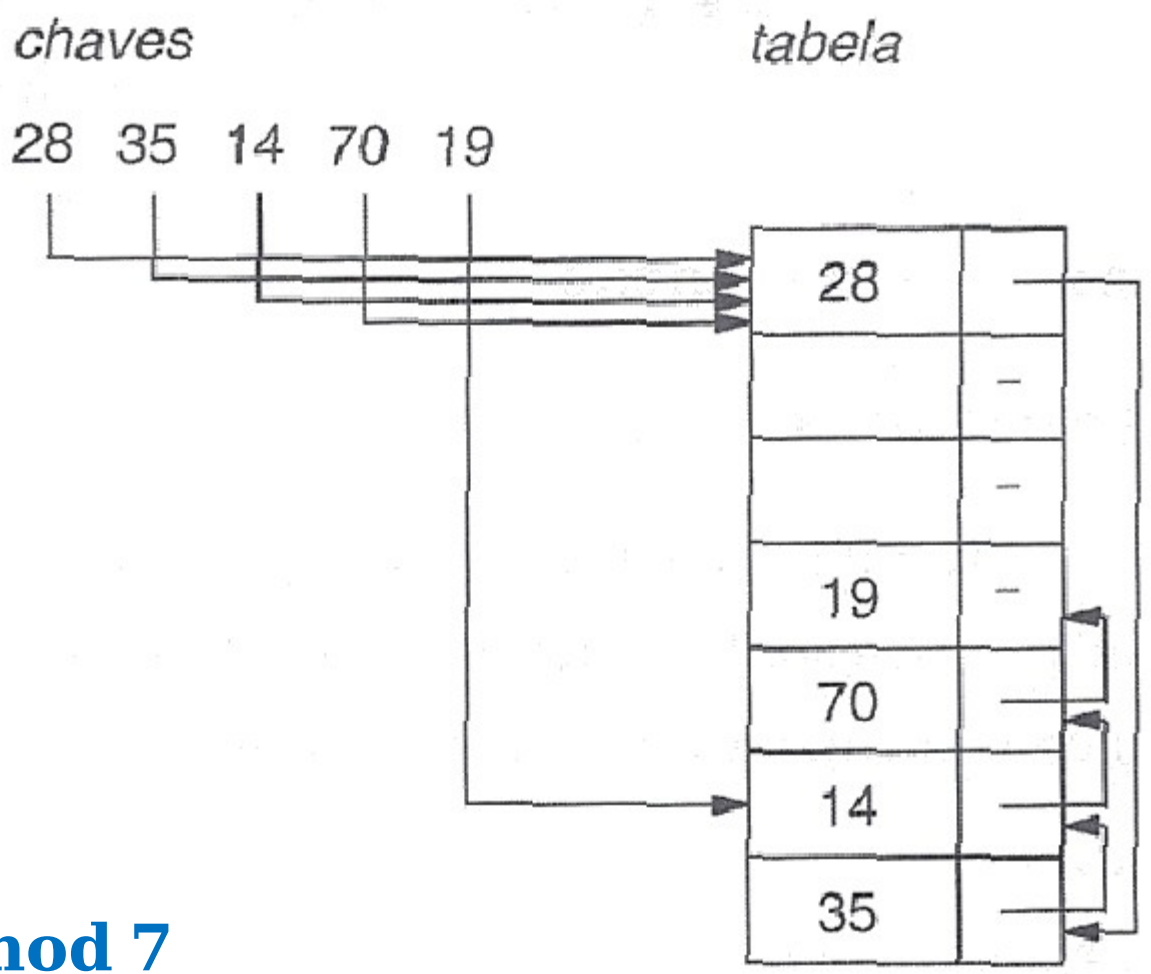
Qualquer endereço da tabela pode ser de base ou de colisão

Quando ocorre colisão a chave é inserida no **primeiro compartimento vazio pré-definido.**

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 mod 7

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

Tentativa Linear

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

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

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

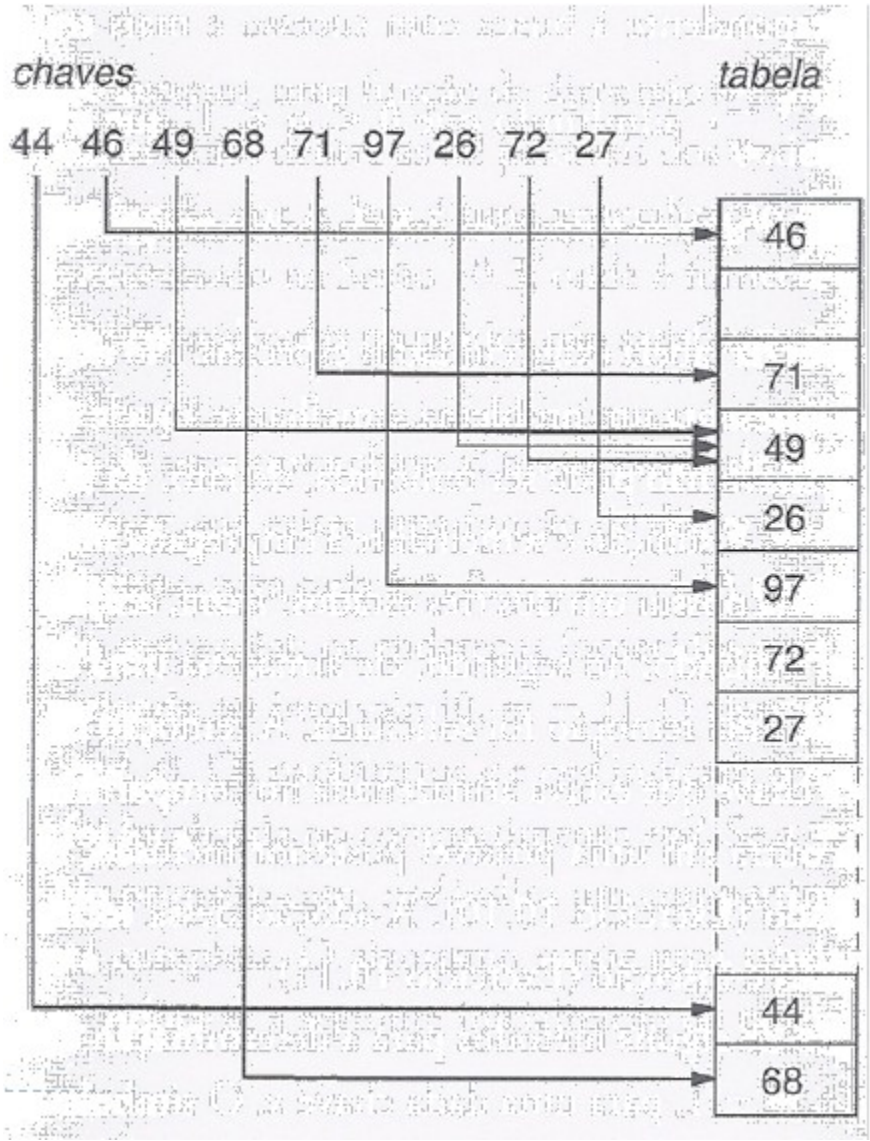
Considera-se uma tabela circular

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: insere no próximo endereço livre



Quais são as desvantagens?

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 compartimento 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

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

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

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

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

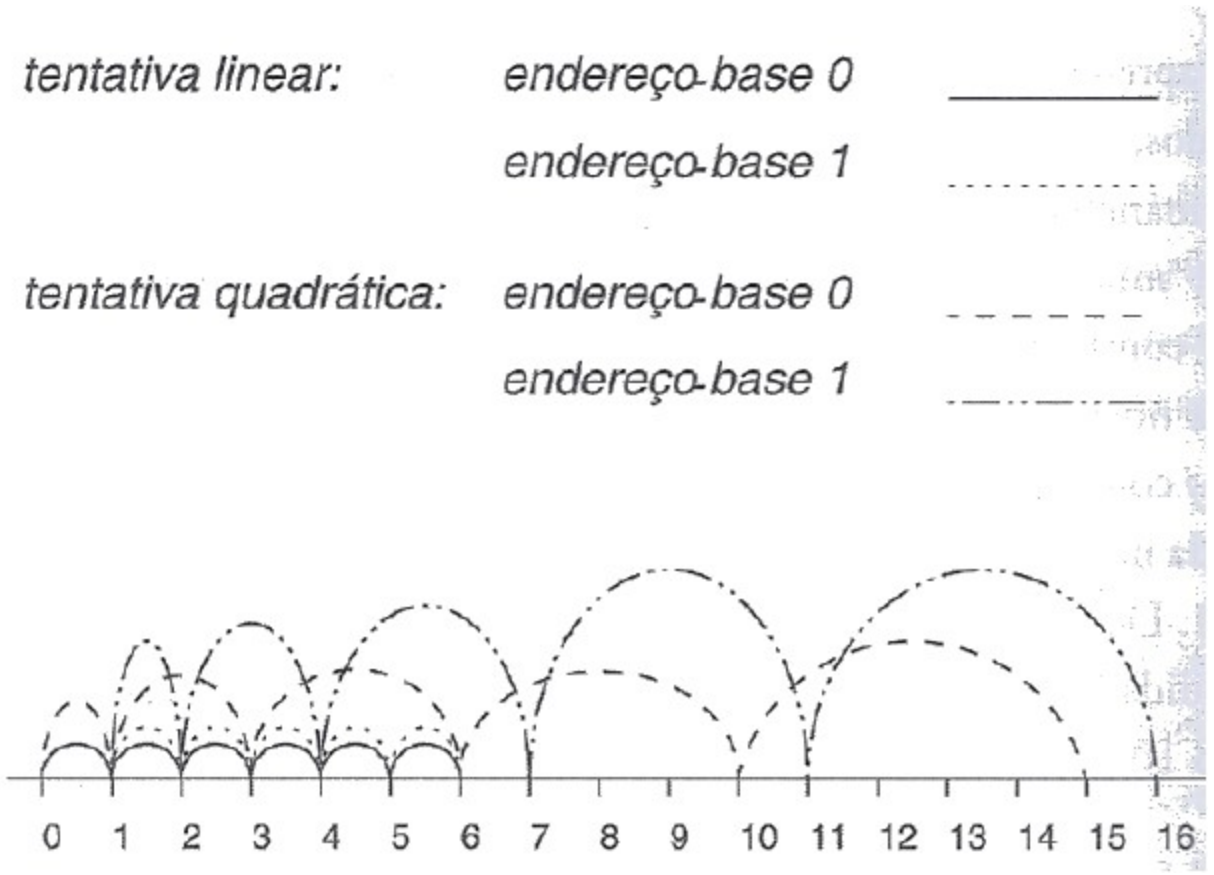
Exemplo:

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

Comparação: Tentativa Linear x Tentativa Quadrática



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)**

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

Trabalho #5: Implementar uma tabela de dispersão dupla

```
class HashItem {
        private:
                string chave;
                string info;
        public:
                HashItem(int string, string info);
                string getChave();
                string getInfo();

};
```

Trabalho #5: Implementar uma tabela de dispersão dupla

```
class TabelaHash {
        private:
                HashItem **tabela;
                 int f1_hash(...);
                 int f2_hash(...);
                 …
        public:
                 TabelaHash(...);
                <tipo>  busca(...);
                string informacao(string chave);
                <tipo> insere(string chave, string info);
                <tipo>  remove(string chave);
                 void imprimeTabela();
                void imprimeItens();
                ~TabelaHash();
};
```