

# Algoritmos e Estruturas de Dados III

6 Hashing

Prof. Hayala Curto  
2022



**PUC Minas**

# Hashing

## Tabelas de Dispersão

As tabelas de dispersão (hash) em disco também podem ser usadas como índices, ao invés das árvores.

- Nessas tabelas, o custo de acesso é  **$O(1)$**  → **Principal Vantagem**
- Custo para isso: sobrecarga, maior consumo de disco ou processamento extra
- A posição do registro é determinada por uma função de dispersão (função hash).

# Hashing – Função de Dispersão

## Função de Dispersão

$h(\text{chave}) \rightarrow \text{endereço}$

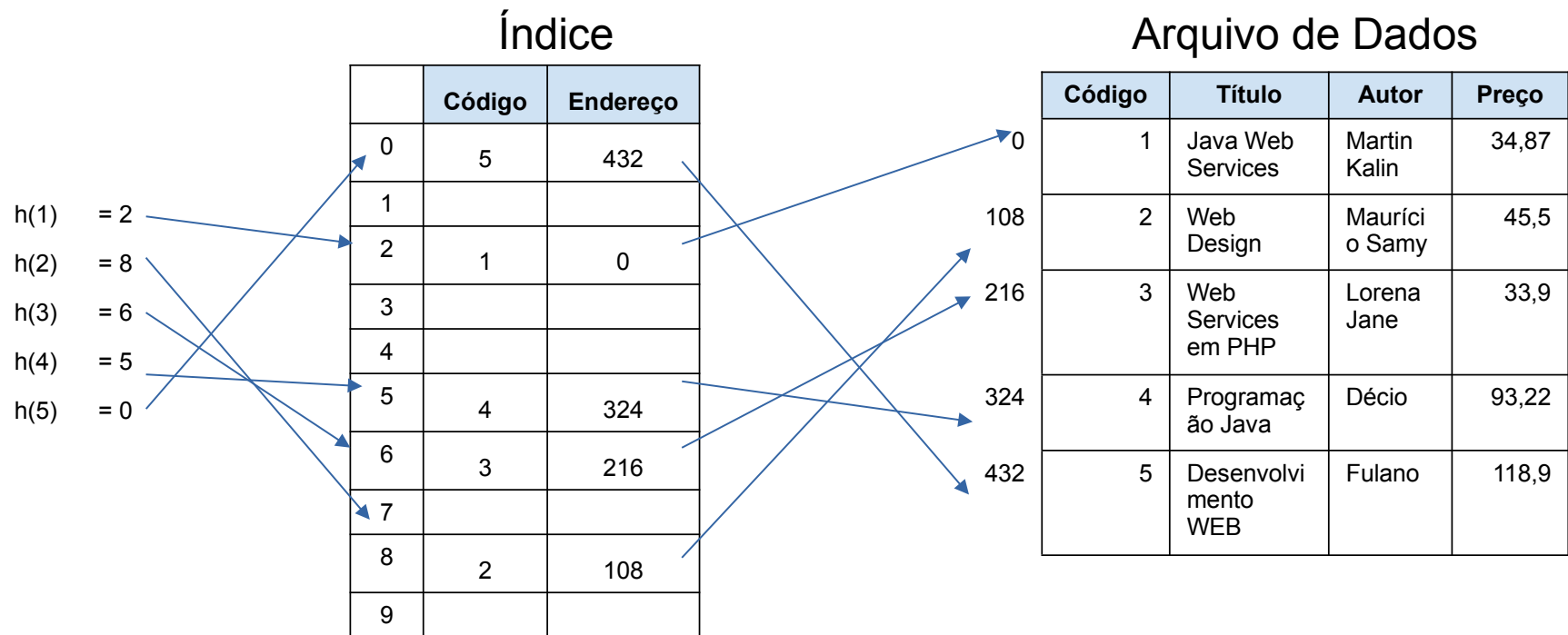
$$H(3204) = 504$$

↑Chave      ↑Endereço

endereço no arquivo de índice

- Depende do número de endereços e da natureza da chave.
- Chave pode ser um campo, uma combinação de campos ou um pedaço de campo
- Registros do índice devem ser de tamanho fixo.
- Quantidade fixa de endereços (depende do tratamento de colisões)
- Quanto mais **disperso**, melhor
- 504 é uma posição relativa. Devemos multiplicar pelo tamanho do registro

# Hashing – Função de Dispersão



Como escolher uma boa função de dispersão?



# Hashing – Função de Dispersão

- **Como escolher uma boa função de dispersão?**

- Operação matemática
- Conversão de base
- Campos não numéricos
- Método da dobra
- Método da divisão
- Outros...

# Hashing – Função de Dispersão

- Como escolher uma boa função de dispersão?
  - **Operação matemática**
  - Conversão de base
  - Campos não numéricos
  - Método da dobra
  - Método da divisão
  - Outros...

# Operação Matemática

## Exemplos de função de dispersão – Operação matemática

Elevar a chave ao **quadrado** e pegar um grupo de dígitos do meio:

- $A = h(453) \rightarrow 453^2 = 205209 \rightarrow A = 52$
- Dois dígitos foram escolhidos pois o arquivo possui apenas 100 endereços
- Resultado fica então dentro da faixa dos endereços disponíveis para o índice



# Hashing – Função de Dispersão

- Como escolher uma boa função de dispersão?
  - Operação matemática
  - **Conversão de base**
  - Campos não numéricos
  - Método da dobra
  - Método da divisão
  - Outros...

# Conversão de Base

## Exemplos de função de dispersão – Conversão de Base

Mudar a chave para outra base:

$$A = h(453) \rightarrow 453_{10} = 382_{11} \rightarrow 382 \bmod 99 = 85 \rightarrow A = 85$$

(99 é a quantidade de endereços no arquivo)

# Hashing – Função de Dispersão

- Como escolher uma boa função de dispersão?
  - Operação matemática
  - Conversão de base
  - **Campos não numéricos**
  - Método da dobra
  - Método da divisão
  - Outros...

# Campos não numéricos

## Exemplos de função de dispersão – Campos não numéricos

Multiplicar o valor ASCII das letras e usar o resto da divisão pelo número de endereços.

Chave	Cálculo	Endereço
<b>J</b> OÃO	$74 \times 79 = 5846$	846
<b>C</b> ARLOS	$67 \times 65 = 4355$	355
<b>G</b> ILBERTO	$71 \times 73 = 5183$	183

- Três dígitos foram escolhidos pois o arquivo possui apenas 1000 endereços
- Resultado fica então dentro da faixa dos endereços disponíveis para o índice

# Hashing – Função de Dispersão

- Como escolher uma boa função de dispersão?
  - Operação matemática
  - Conversão de base
  - Campos não numéricos
  - **Método da dobra**
  - Método da divisão
  - Outros...

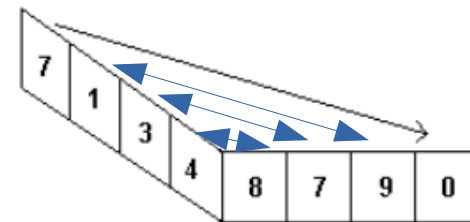
# Método da Dobra

A chave é interpretada como uma sequência de dígitos escritos em um “pedaço papel”.

O método consiste em “dobrar esse papel”, de maneira que os dígitos se sobreponham.

O processo é repetido até que os dígitos formem um número menor que o tamanho da tabela hash.

7	1	3	4	8	7	9	0
---	---	---	---	---	---	---	---



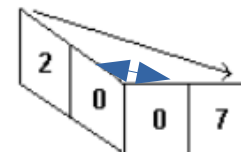
$$7 + 0 = 7$$

$$1 + 9 = \cancel{10}$$

$$3 + 7 = \cancel{10}$$

$$4 + 8 = \cancel{12}$$

2	0	0	7
---	---	---	---



$$2 + 7 = 9$$

$$0 + 0 = 0$$

0	9
---	---

# Método da Dobra - Exemplo

Supondo que cada chave ocupa 10 bits e a dimensão da tabela é 32 ( $2^5$ ).

Deve-se transformar as chaves em endereços que ocupam 5 bits.

Exemplo:

71 = 00010 00111 => ?

46 = 00001 01110 => ?

# Método da Dobra - Exemplo

Supondo que cada chave ocupa 10 bits e a dimensão da tabela é 32 ( $2^5$ ).

Deve-se transformar as chaves em endereços que ocupam 5 bits.

Exemplo:

$71 = 00010\ 00111 \Rightarrow 01111 \Rightarrow 15$

$46 = 00001\ 01110 \Rightarrow 11110 \Rightarrow 30$



# Hashing – Função de Dispersão

- Como escolher uma boa função de dispersão?
  - Operação matemática
  - Conversão de base
  - Campos não numéricos
  - Método da dobra
  - **Método da divisão**
  - Outros...

# Método da Divisão

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

- Fácil e eficiente
- Alguns valores de  $m$  são melhores que outros

$m$  sendo um **número primo** ajuda a “espalhar” os valores

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**

# Método da Divisão

Transformar a chave  $x$  em um endereço-base  $h(x)$ , que é um valor entre 0 e  $m$ .

Exemplo:

$$h(x) = x \bmod 5$$

T	
	0
	1
	2
	3
	4

# Método da Divisão

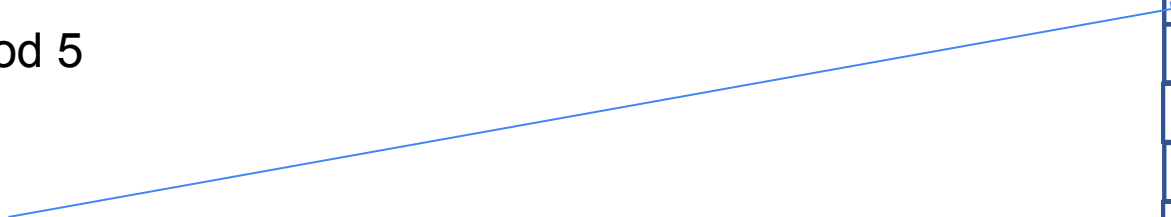
Transformar a chave  $x$  em um endereço-base  $h(x)$ , que é um valor entre 0 e  $m$ .

Exemplo:

$$h(x) = x \bmod 5$$

Chaves: 70

T	
70	0
	1
	2
	3
	4



# Método da Divisão

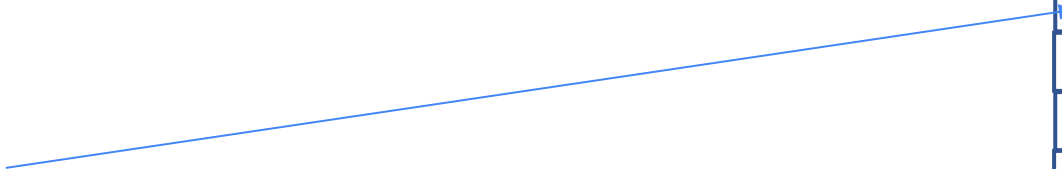
Transformar a chave  $x$  em um endereço-base  $h(x)$ , que é um valor entre 0 e  $m$ .

Exemplo:

$$h(x) = x \bmod 5$$

Chaves: 70, 51

T	
70	0
51	1
	2
	3
	4



# Método da Divisão

Transformar a chave  $x$  em um endereço-base  $h(x)$ , que é um valor entre 0 e  $m$ .

Exemplo:

$$h(x) = x \bmod 5$$

Chaves: 70, 51, 99

T	
70	0
51	1
	2
	3
99	4

# Método da Divisão

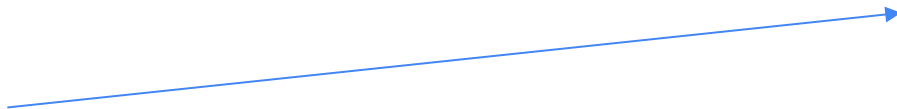
Transformar a chave  $x$  em um endereço-base  $h(x)$ , que é um valor entre 0 e  $m$ .

Exemplo:

$$h(x) = x \bmod 5$$

Chaves: 70, 51, 99, 17

T	
70	0
51	1
17	2
	3
99	4



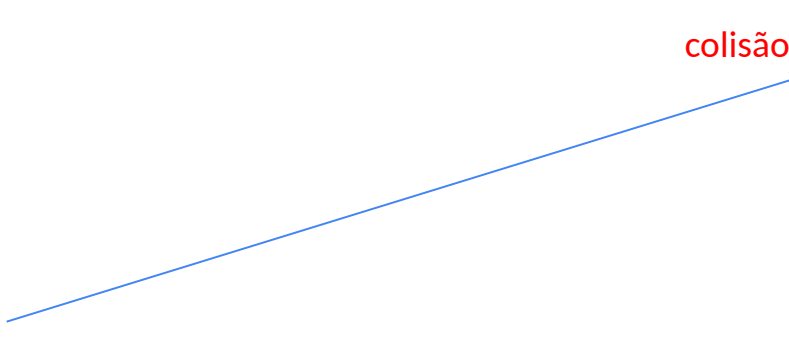
# Método da Divisão

Transformar a chave  $x$  em um endereço-base  $h(x)$ , que é um valor entre 0 e  $m$ .

Exemplo:

$$h(x) = x \bmod 5$$

Chaves: 70, 51, 99, 17, 15



15, 70	0
51	1
17	2
	3
99	4



# Método da Divisão


Transformar a chave  $x$  em um endereço-base  $h(x)$ , que é um valor entre 0 e  $m$ .

Exemplo:

$$h(x) = x \bmod 5$$

Chaves: 70, 51, 99, 17, 15

**COMO  
RESOLVER?**



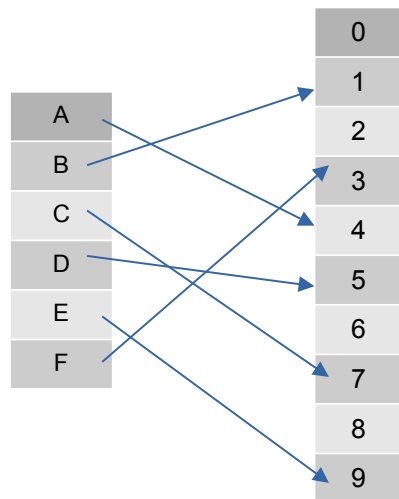
15, 70	0
51	1
17	2
	3
99	4

# Colisões

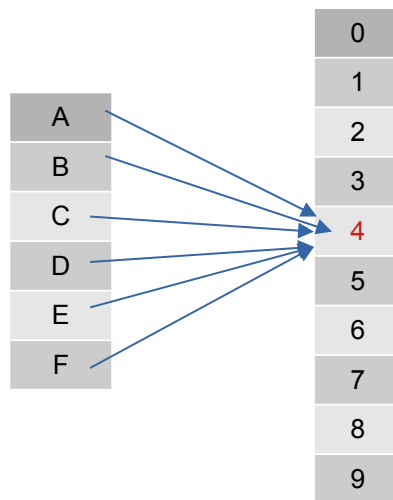


**PUC Minas**

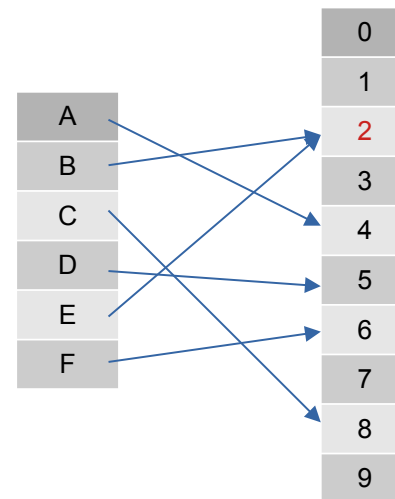
# Hashing – Colisões



**Desejável**

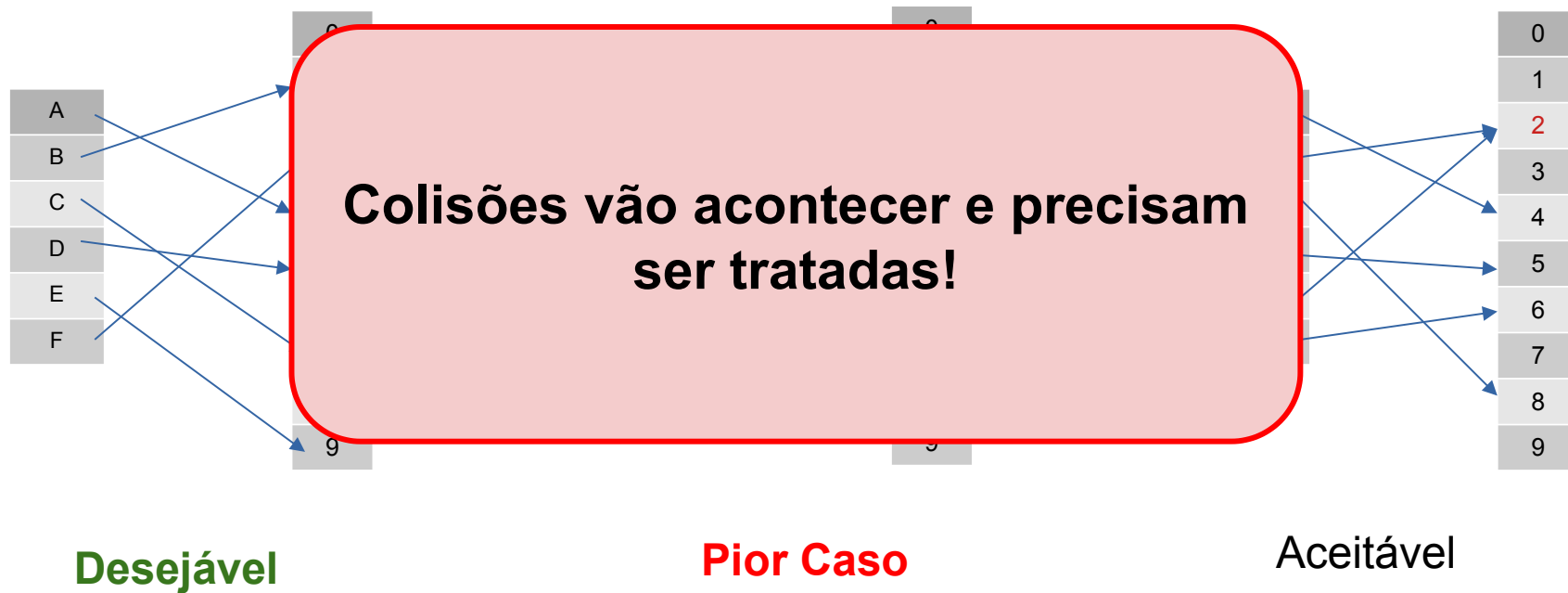


**Pior Caso**



**Aceitável**

# Hashing – Colisões



# Colisões - Tratamento de Colisões

Alternativas para o tratamento de colisões:

- **Endereçamento aberto** – usa outras posições vazias dentro da própria da tabela hash
- **Encadeamento interno** – usa uma área extra dentro da própria da tabela hash
- **Encadeamento externo** – usa uma área externa, fora da tabela (p.ex., um segundo arquivo).

# Colisões - Tratamento de Colisões

Alternativas para o tratamento de colisões:

- **Endereçamento aberto** – usa outras posições vazias dentro da própria da tabela hash
- **Encadeamento interno** – usa uma área extra dentro da própria da tabela hash
- **Encadeamento externo** – usa uma área externa, fora da tabela (p.ex., um segundo arquivo).

# Endereçamento aberto

Uma nova posição **dentro** da área da tabela será procurada

Fórmula matemática para encontrar nova posição

- Sondagem linear
- Sondagem quadrática
- Duplo hash (double hashing)

# Endereçamento aberto

Uma nova posição **dentro** da área da tabela será procurada

Fórmula matemática para encontrar nova posição

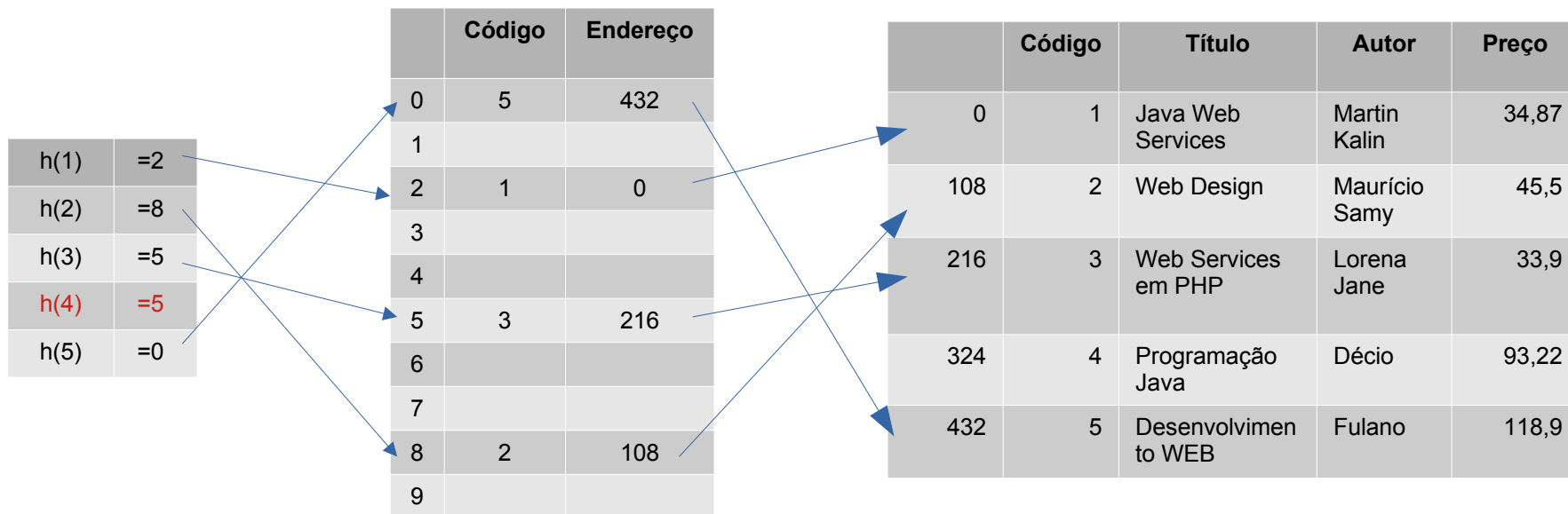
- **Sondagem linear**
- Sondagem quadrática
- Duplo hash (double hashing)



# Endereçamento aberto

**Sondagem linear** – as próximas posições são sondadas (circularmente), até que uma posição livre seja encontrada. Regra:

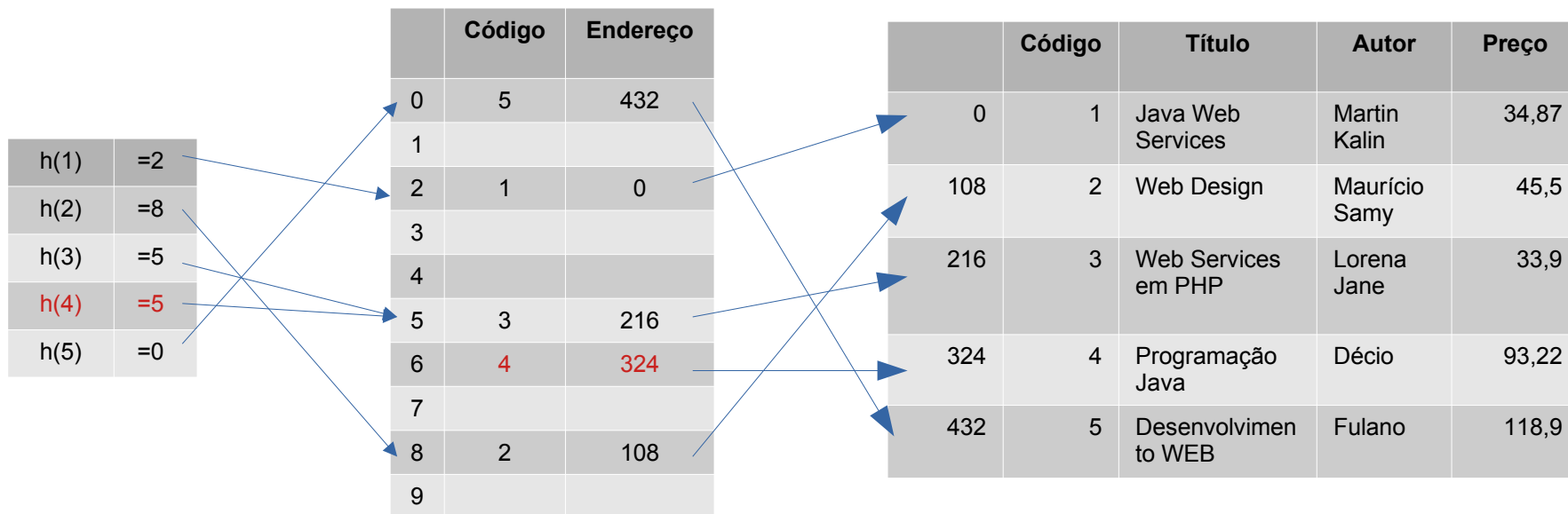
$$h(k, i) = [h(k) + i] \bmod n$$



# Endereçamento aberto

**Sondagem linear** – as próximas posições são sondadas (circularmente), até que uma posição livre seja encontrada. Regra:

$$h(k, i) = [h(k) + i] \bmod n$$



# Endereçamento aberto

**Sondagem linear** – as próximas posições são sondadas (circularmente), até que uma posição livre seja encontrada. Regra:  $h(k, i) = [h(k) + i] \bmod n$

- Avanço unitário  $\rightarrow +i$
- Avanço circular  $\rightarrow \bmod n$ , em que  $n$  é o número de posições da tabela
- Até encontrar uma posição **vazia**
- Retorno ao mesmo índice indica que tabela hash está cheia
- Busca deve ler registros de forma sequencial até encontrar uma posição **vazia**
- Uso de indicador de **lápide** pode ser necessário caso exclusões ocorram, para atender a estratégia de busca acima (sequencial até encontrar posição vazia sem marcação de lápide)
- Se ocorrerem muitas colisões, pode ser criado um agrupamento (clustering) de chaves em uma certa área. `_x0001_` Isso pode fazer com que sejam necessários muitos acessos para recuperar um certo registro.

# Endereçamento aberto

Uma nova posição **dentro** da área da tabela será procurada

Fórmula matemática para encontrar nova posição

- Sondagem linear
- **Sondagem quadrática**
- Duplo hash (double hashing)

# Endereçamento aberto

**Sondagem quadrática** – a distância até a próxima posição a ser sondada é determinada pelo quadrado da tentativa, Regra:

$$h(k, i) = [h(k) + i^2] \bmod n$$

$h(1)$	$=2$
$h(2)$	$=8$
$h(3)$	$=5$
$h(4)$	$=2$
$h(5)$	$=2$

	Código	Endereço
0		
1		
2	1	0
3		
4		
5	3	216
6		
7		
8	2	108
9		

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	34,87
108	2	Web Design	Maurício Samy	45,5
216	3	Web Services em PHP	Lorena Jane	33,9
324	4	Programação Java	Décio	93,22
432	5	Desenvolvimento WEB	Fulano	118,9

# Endereçamento aberto

**Sondagem quadrática** – a distância até a próxima posição a ser sondada é determinada pelo quadrado da tentativa, Regra:

$$h(k, i) = [h(k) + i^2] \bmod n$$

	Código	Endereço
0		
1		
2	1	0
3	4	324
4		
5	3	216
6		
7		
8	2	108
9		

$h(1)$	$=2$
$h(2)$	$=8$
$h(3)$	$=5$
$h(4)$	$=2$
$h(5)$	$=2$

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	34,87
108	2	Web Design	Maurício Samy	45,5
216	3	Web Services em PHP	Lorena Jane	33,9
324	4	Programação Java	Décio	93,22
432	5	Desenvolvimento WEB	Fulano	118,9

# Endereçamento aberto

**Sondagem quadrática** – a distância até a próxima posição a ser sondada é determinada pelo quadrado da tentativa, Regra:

$$h(k, i) = [h(k) + i^2] \bmod n$$

	Código	Endereço
0		
1		
2	1	0
3	4	324
4		
5	3	216
6	5	432
7		
8	2	108
9		

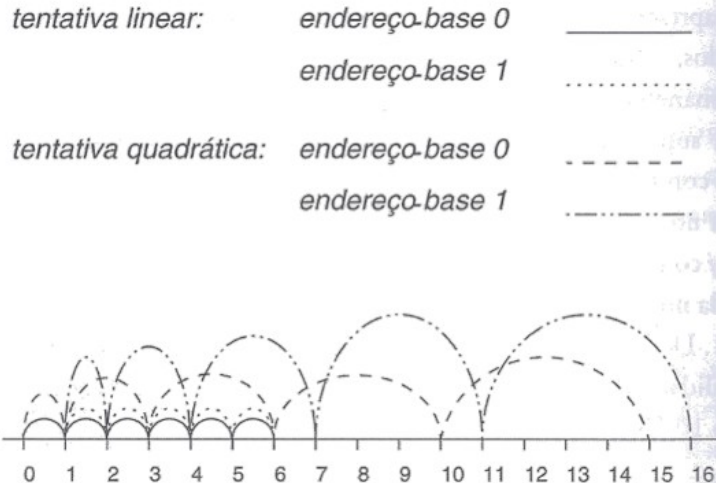
$h(1)$	$=2$
$h(2)$	$=8$
$h(3)$	$=5$
$h(4)$	$=2$
$h(5)$	$=2$

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	34,87
108	2	Web Design	Maurício Samy	45,5
216	3	Web Services em PHP	Lorena Jane	33,9
324	4	Programação Java	Décio	93,22
432	5	Desenvolvimento WEB	Fulano	118,9

# Endereçamento aberto

**Sondagem quadrática** – a distância até a próxima posição a ser sondada é determinada pelo quadrado da tentativa, Regra:  $h(k, i) = [h(k) + i^2] \bmod n$

- Avanço quadrático  $\rightarrow + i^2$
- Avanço circular  $\rightarrow \bmod n$ , em que  $n$  é o número de posições da tabela
- Até encontrar uma posição **vazia**





# Endereçamento aberto

Uma nova posição **dentro** da área da tabela será procurada

Fórmula matemática para encontrar nova posição

- Sondagem linear
- Sondagem quadrática
- **Duplo hash (double hashing)**

# Endereçamento aberto

**Duplo hash** – a distância até a próxima posição a ser sondada é determinada por uma segunda função hash. Regra:

$$h(k, i) = [h(k) + i * h'(k)] \bmod n$$

$h(1)$	=2
$h(2)$	=8
$h(3)$	=5
$h(4)$	=2
$h'(4)$	=4
$h(5)$	=2
$h'(5)$	=1

	Código	Endereço
0		
1		
2	1	0
3		
4		
5	3	216
6		
7		
8	2	108
9		

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	34,87
108	2	Web Design	Maurício Samy	45,5
216	3	Web Services em PHP	Lorena Jane	33,9
324	4	Programação Java	Décio	93,22
432	5	Desenvolvimento WEB	Fulano	118,9

# Endereçamento aberto

**Duplo hash** – a distância até a próxima posição a ser sondada é determinada por uma segunda função hash. Regra:

$$h(k, i) = [h(k) + i * h'(k)] \bmod n$$

$h(1)$	$=2$
$h(2)$	$=8$
$h(3)$	$=5$
$h(4)$	$=2$
$h'(4)$	$=4$
$h(5)$	$=2$
$h'(5)$	$=1$

	Código	Endereço
0		
1		
2	1	0
3		
4		
5	3	216
6	4	324
7		
8	2	108
9		

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	34,87
108	2	Web Design	Maurício Samy	45,5
216	3	Web Services em PHP	Lorena Jane	33,9
324	4	Programação Java	Décio	93,22
432	5	Desenvolvimento WEB	Fulano	118,9

# Endereçamento aberto

**Duplo hash** – a distância até a próxima posição a ser sondada é determinada por uma segunda função hash. Regra:

$$h(k, i) = [h(k) + i * h'(k)] \bmod n$$

$h(1)$	$=2$
$h(2)$	$=8$
$h(3)$	$=5$
$h(4)$	$=2$
$h'(4)$	$=4$
$h(5)$	$=2$
$h'(5)$	$=1$

	Código	Endereço
0		
1		
2	1	0
3	5	432
4		
5	3	216
6	4	324
7		
8	2	108
9		

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	34,87
108	2	Web Design	Maurício Samy	45,5
216	3	Web Services em PHP	Lorena Jane	33,9
324	4	Programação Java	Décio	93,22
432	5	Desenvolvimento WEB	Fulano	118,9

# Endereçamento aberto

**Duplo hash** – a distância até a próxima posição a ser sondada é determinada por uma segunda função hash. Regra:  $h(k, i) = [h(k) + i * h'(k)] \bmod n$

- Avanço com segunda função hash  $\rightarrow +i * h'(k)$
- Avanço circular  $\rightarrow \bmod n$ , em que  $n$  é o número de posições da tabela
- Avanço da segunda função hash a partir da posição atual dada pela primeira função hash.
- Vantagem: tende a espalhar melhor as chaves pelos endereços.
- Desvantagem: os endereços podem estar muito distantes um do outro (o princípio da localidade é violado), provocando seekings adicionais.

# Colisões - Tratamento de Colisões

Alternativas para o tratamento de colisões:

- **Endereçamento aberto** – usa outras posições vazias dentro da própria da tabela hash
- **Encadeamento interno** – usa uma área extra dentro da própria da tabela hash
- **Encadeamento externo** – usa uma área externa, fora da tabela (p.ex., um segundo arquivo).

# Encadeamento



# Hashing – Encadeamento

## Encadeamento Interno

Os registros colididos são armazenados  
Em uma área extra (geralmente  
separada da área principal)

h(1)	=2
h(2)	=8
h(3)	=5
h(4)	=5
h(5)	=2

	Código	Endereço	Próximo
0			-1
1			-1
2	1	0	-1
3			-1
4			-1
5	3	216	-1
6			-1
7			-1
8	2	108	-1
9			-1
10			-1
11			-1



# Hashing – Encadeamento

## Encadeamento Interno

Os registros colididos são armazenados  
Em uma área extra (geralmente  
separada da área principal)

h(1)	=2
h(2)	=8
h(3)	=5
h(4)	=5
h(5)	=2

	Código	Endereço	Próximo
0			-1
1			-1
2	1	0	11
3			-1
4			-1
5	3	216	10
6			-1
7			-1
8	2	108	-1
9			-1
10	4	324	-1
11			-1

# Hashing – Encadeamento

## Encadeamento Interno

Os registros colididos são armazenados  
Em uma área extra (geralmente  
separada da área principal)

h(1)	=2
h(2)	=8
h(3)	=5
h(4)	=5
h(5)	=2

	Código	Endereço	Próximo
0			-1
1			-1
2	1	0	11
3			-1
4			-1
5	3	216	10
6			-1
7			-1
8	2	108	-1
9			-1
10	4	324	-1
11	5	432	-1

# Hashing – Encadeamento

## Encadeamento

Análise: Com uma “boa” função hash, assume-se que qualquer item do conjunto de chaves tem igual **probabilidade** de ser endereçado para qualquer entrada da tabela.

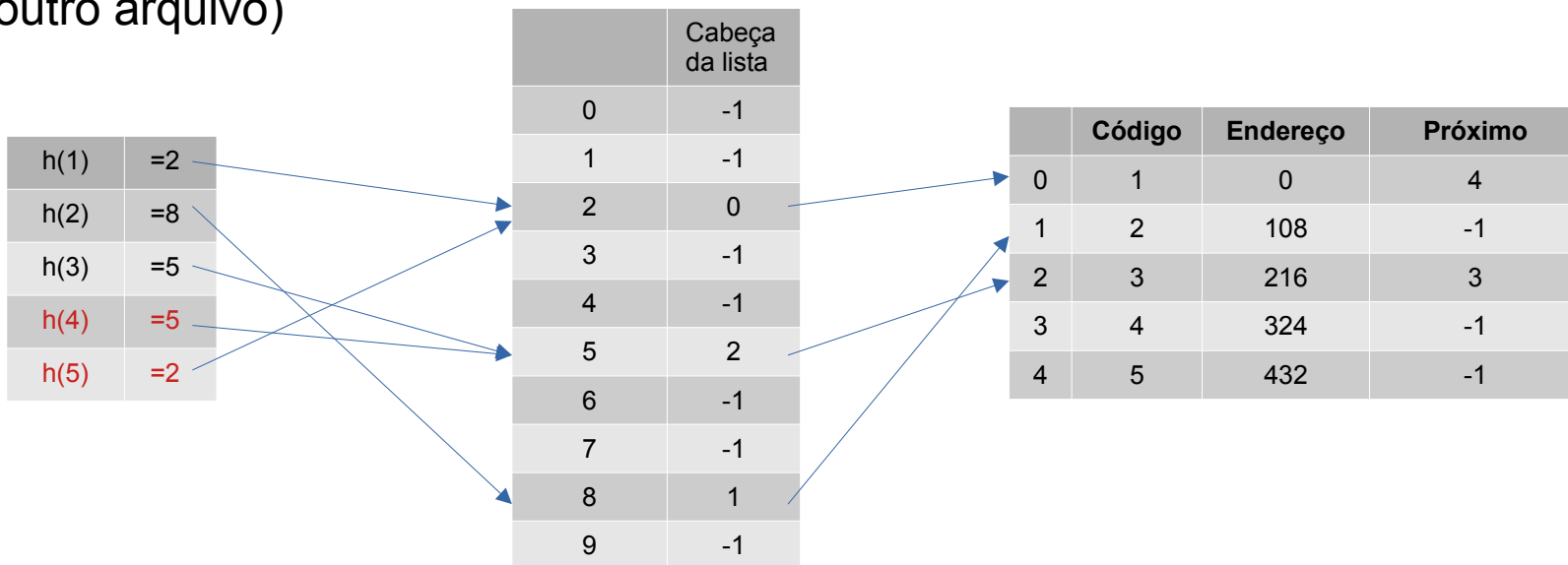
Logo, o comprimento esperado de cada lista encadeada é  $n/m$ , chamado de **fator de carga**, em que  $n$  representa o número de registros na tabela e  $m$  o tamanho da tabela.

As operações pesquisa, inserção e remoção custam  $O(1 + n/m)$  em média, sendo que a constante 1 representa o tempo para encontrar a entrada da tabela (ou seja, calcular a função hash), e  $n/m$  o tempo para percorrer a lista.

# Hashing – Encadeamento

## Encadeamento Externo

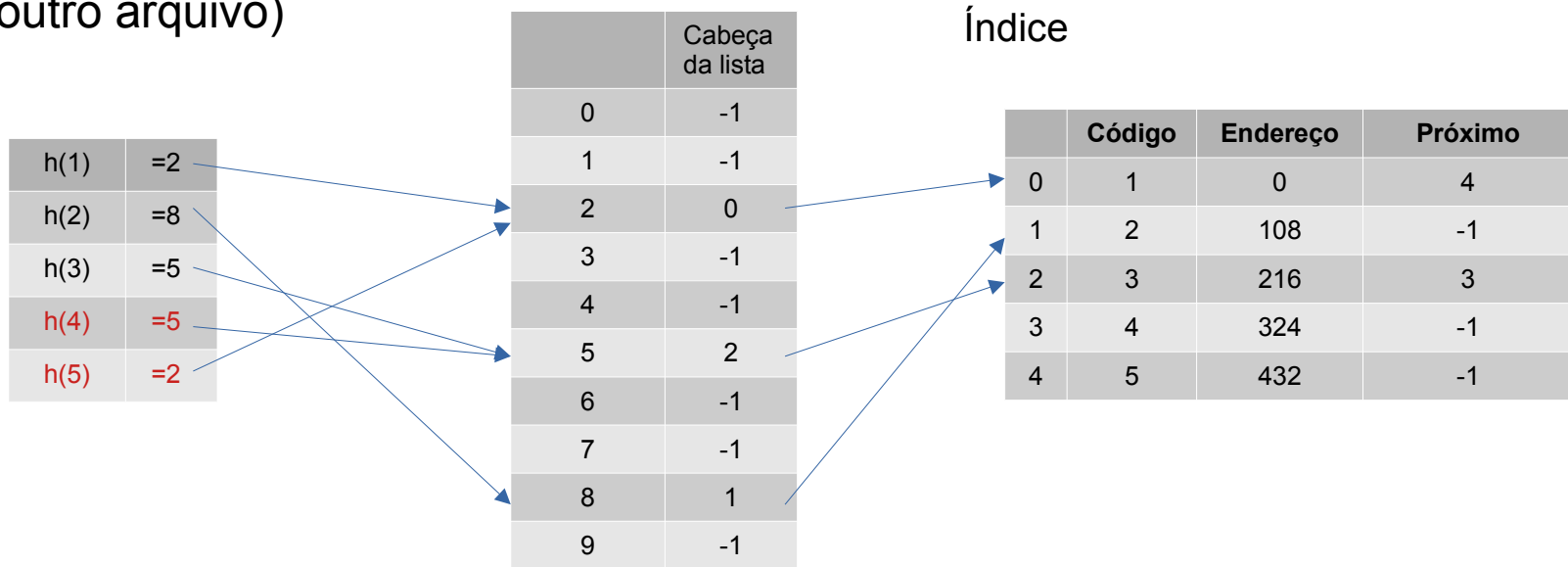
Lista encadeada – todos os registros são armazenados em uma lista encadeada (outro arquivo)



# Hashing – Encadeamento

## Encadeamento Externo

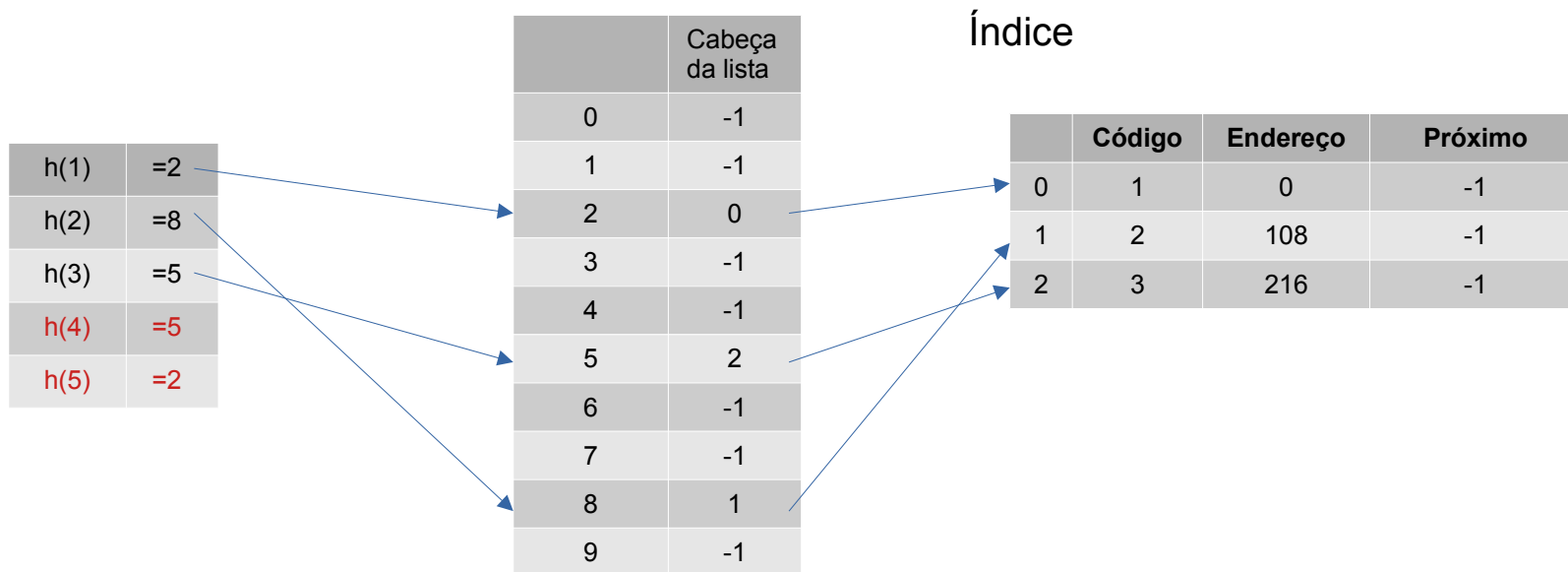
Lista encadeada – todos os registros são armazenados em uma **lista encadeada** (outro arquivo)



# Hashing – Encadeamento

## Encadeamento Externo

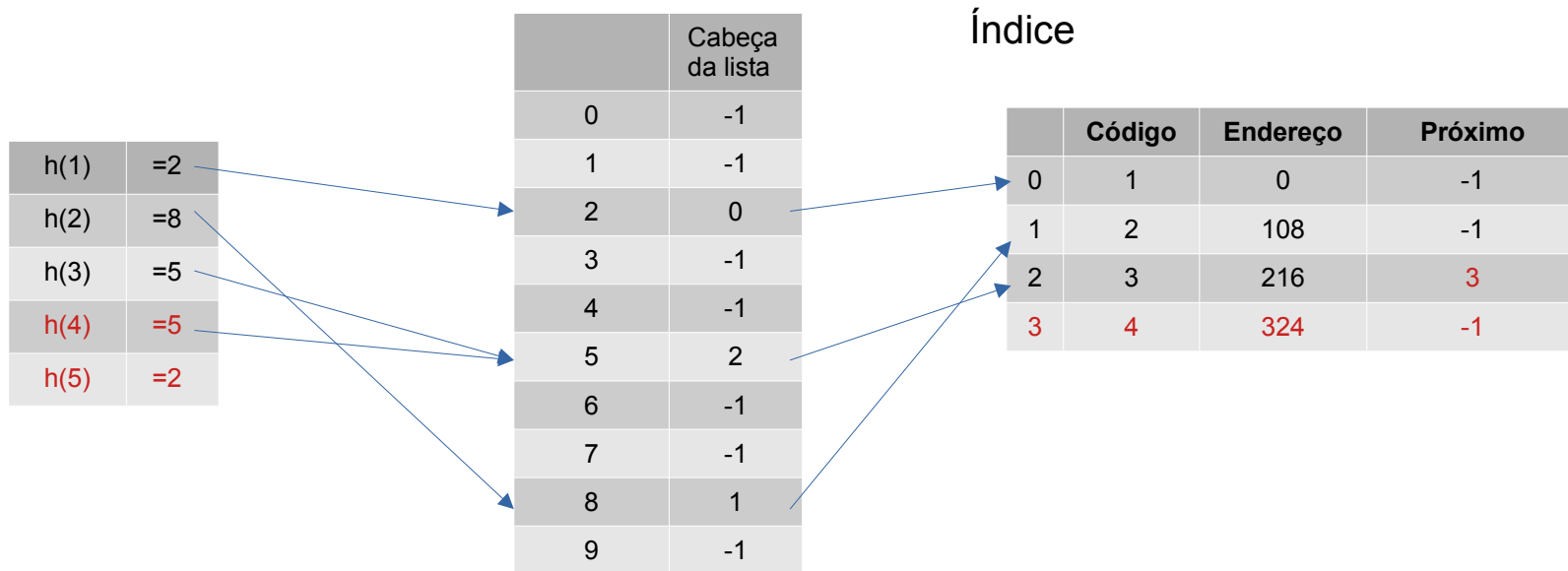
Preenchimento da lista encadeada



# Hashing – Encadeamento

## Encadeamento Externo

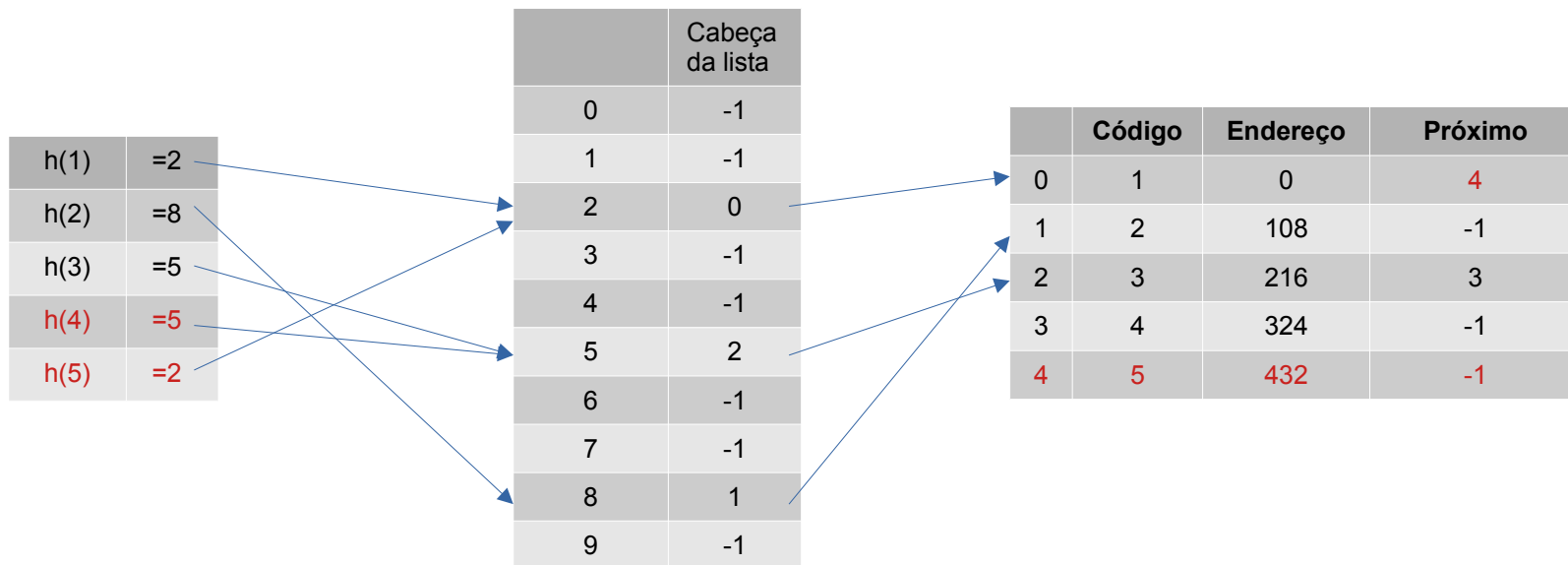
Preenchimento da lista encadeada



# Hashing – Encadeamento

## Encadeamento Externo

Preenchimento da lista encadeada

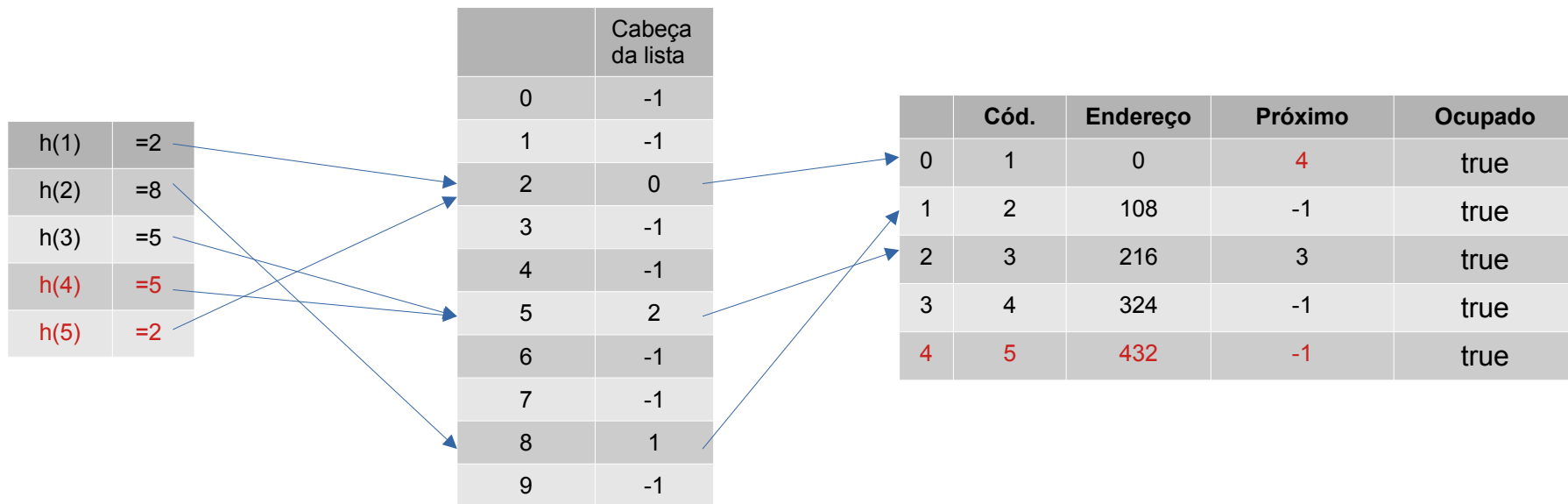




# Hashing – Encadeamento

## Encadeamento Externo

Preenchimento da lista encadeada



# Hashing – Encadeamento

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

# Hashing – Encadeamento

## Encadeamento Externo

Lista encadeada – todos os registros são armazenados em uma **lista encadeada**

- Lista encadeada pode ser outro arquivo
- Lista encadeada pode estar no mesmo arquivo da tabela hash
- Parte fixa, constante, reservada para a tabela
- Parte crescente com a lista encadeada
- Cresce de acordo com a necessidade
- Aparentemente a melhor solução, mas ainda precisamos melhorar e evitar a leitura de apenas 1 registro a cada busca em disco → Buckets

# Buckets



**PUC Minas**

# Hashing – Buckets

Balde → Cesto

- Da mesma forma que no caso da árvore B, é importante otimizar o acesso ao disco. Árvore Binária X Árvore B
- Assim, cada posição no índice, pode conter mais de uma entrada (ou registro). Hash sem Buckets X Hash com Buckets

Exemplo:

- Registro no índice = 12 bytes
- Setor do HD = 4096 bytes = 341,33 registros

# Hashing – Buckets

Buckets com 4 pares de chave/endereço

$h(1)$	=2
$h(2)$	=8
$h(3)$	=5
$h(4)$	=5
$h(5)$	=2

	Código	Endereço	Código	Endereço	Código	Endereço
0						
1						
2	1	0	5	432		
3						
4						
5	3	216	3	324		
6						
7						
8	2	108				
9						

# Hashing – Buckets

Tratamento de colisões (quando o cesto está cheio)

- Alocar o registro no próximo cesto em que houver espaço disponível (usando endereçamento aberto)
- Usar uma das técnicas anteriores (considerando que cada posição equivale a um novo cesto)

# Hashing – Buckets encadeados

Buckets com 4 pares de chave/endereço

	Código	Endereço	Código	Endereço	Código	Endereço	Próx
	0						
	1						
$h(1) = 2$	2	1	0	5	432		
$h(2) = 8$	3						
$h(3) = 5$	4						
$h(4) = 5$	5	3	216	4	324		
$h(5) = 2$	6						
	7						
	8	2	108				
	9						



# Hashing – Buckets

## Funções de Dispersão e Buckets

- A **pior função** de dispersão mapeia todos os valores das chaves de busca para o mesmo bucket; isto faz o tempo de acesso proporcional ao número de valores das chaves de busca no arquivo.
- Uma função de dispersão ideal é **uniforme** isto é, a cada bucket é atribuído o mesmo número de valores das chaves de busca do conjunto de todos os possíveis valores.
- Uma função de dispersão ideal é **aleatória**, assim cada bucket terá quase o mesmo número de valores atribuídos a ele, independente da distribuição real dos valores da chave de busca no arquivo

# Hashing – Estático X Dinâmico

- Tabelas Hash podem ter uma capacidade fixa ou crescer de acordo com os dados armazenados nas mesmas
- Hash Estático: A capacidade da tabela é fixa e determinada em sua criação. Se os dados crescem além da capacidade, novo hash é necessário
- Hash Dinâmico: A capacidade da tabela é pequena para acomodar poucos dados no início, mas aumenta (ou diminui) a medida que novos dados são inseridos (ou removidos)