

Índices: Hash

Tabela de dispersão

- As tabelas de dispersão (*hash*) em disco também podem ser usadas como índices, ao invés das árvores.
- Nessas tabelas, o curso de acesso é $O(1)$.
- A posição do registro é determinada por uma função de dispersão (ou função *hash*).

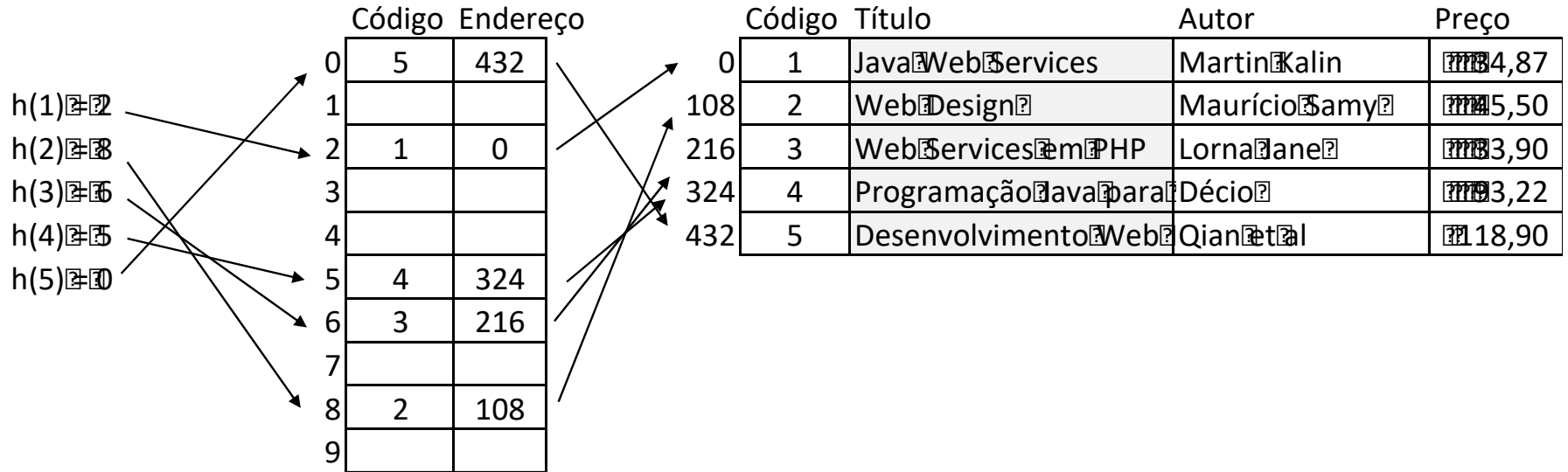
Função de dispersão

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

$$\begin{array}{ccc} h(3204) = 504 \\ \uparrow \quad \quad \uparrow \\ \text{chave} \quad \quad \text{endereço} \end{array}$$

- Depende do número de endereços e da natureza da chave.
- Registros do índice devem ser de tamanho fixo.
- Quantidade fixa de endereços
(depende do tratamento de colisões)

Tabela de dispersão



Exemplos de função de dispersão

- 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)

Exemplos de função de dispersão

- 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)

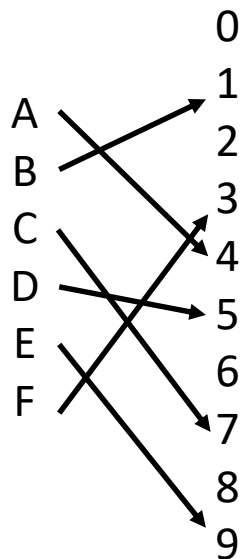
Exemplos de função de dispersão

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

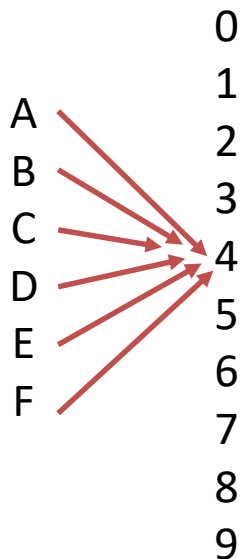
Chave	Cálculo	Endereço
JOÃO	$74 \times 79 = 5846$	846
CARLOS	$67 \times 65 = 4355$	355
GILBERTO	$71 \times 73 = 5183$	183

Colisões

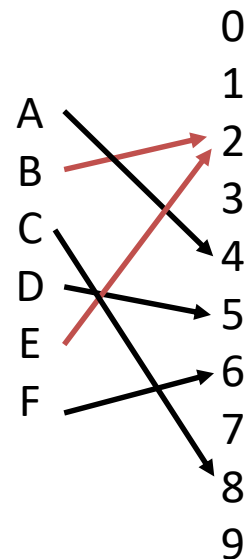
Colisões



DESEJÁVEL



PIOR CASO



ACEITÁVEL

Tratamento de colisões

- Alternativas:
 - **Encadeamento interno** – usa outras posições vazias dentro da própria da tabela *hash*
 - **Encadeamento externo** – usa uma área extra, além da tabela *hash*, como uma área de extensão, ou um segundo arquivo.

Encadeamento interno

- Endereçamento aberto - uma nova posição **dentro da área da tabela** será procurada
 - Sondagem linear
 - Sondagem quadrática
 - Duplo *hash* (*double hashing*)

Encadeamento interno

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

	Código	Endereço
	0	5 432
$h(1) = 2$	1	0
$h(2) = 8$	2	108
$h(3) = 5$	3	216
$h(4) = 15$	4	324
$h(5) = 10$	5	

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	4,87
108	2	Web Design Responsivo	Maurício Samy Silva	5,50
16	3	Web Services em PHP	Lorna Jane Mitchell	3,90
24	4	Programação Java para Web	Décio Heinzemann	3,22
32	5	Desenvolvimento Web Java	Qian et al	18,90

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

Encadeamento interno

- Sondagem quadrática – a distância até a próxima posição a ser sondada é determinada pelo quadrado da tentativa

	Código	Endereço
	0	
	1	0
	4	324
	3	216
	5	432
	2	108

$h(1) = 1$
 $h(2) = 4$
 $h(3) = 9$
 $h(4) = 16$
 $h(5) = 25$

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	4,87
108	2	Web Design Responsivo	Maurício Samy Silva	5,50
16	3	Web Services em PHP	Lorna Jane Mitchell	3,90
24	4	Programação Java para Web	Décio Heinzelmann	3,22
32	5	Desenvolvimento Web Java	Qian et al	18,90

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

Encadeamento interno

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

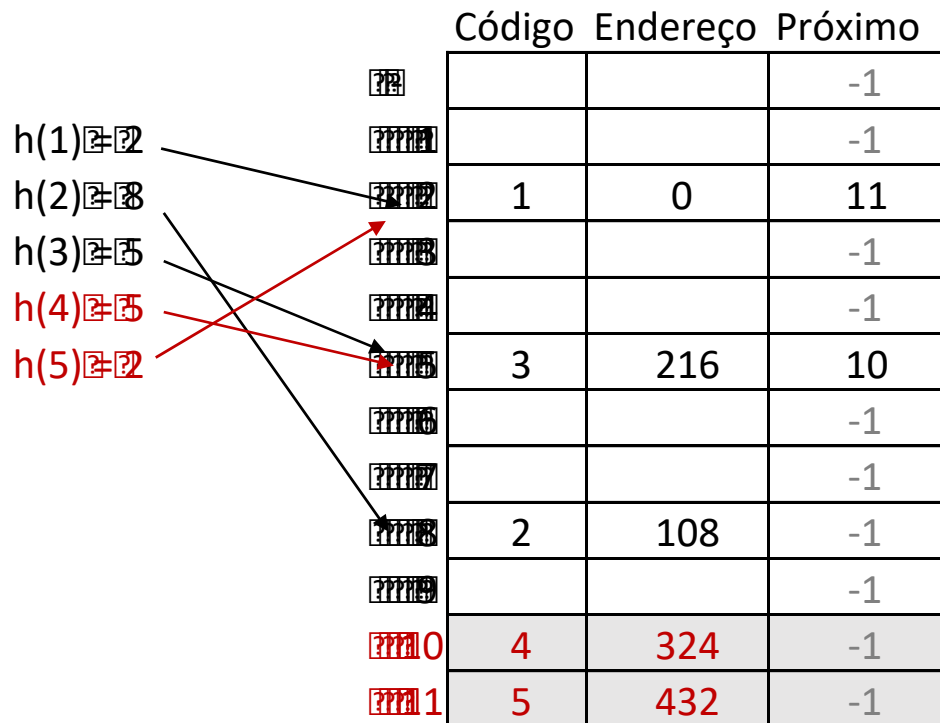
	Código	Endereço
	0	
$h(1) = 2$		
$h(2) = 8$		
$h(3) = 5$	1	0
$h(4) = 2$	5	432
$h'(4) = 4$		
$h(5) = 2$	3	216
$h'(5) = 1$	4	324
	2	108

	Código	Título	Autor	Preço
0	1	Java Web Services	Martin Kalin	4,87
108	2	Web Design Responsivo	Maurício Samy Silva	5,50
216	3	Web Services em PHP	Lorna Jane Mitchell	3,90
324	4	Programação Java para Web	Décio Heinzelmann	3,22
432	5	Desenvolvimento Web Java	Qian et al	18,90

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

Encadeamento externo

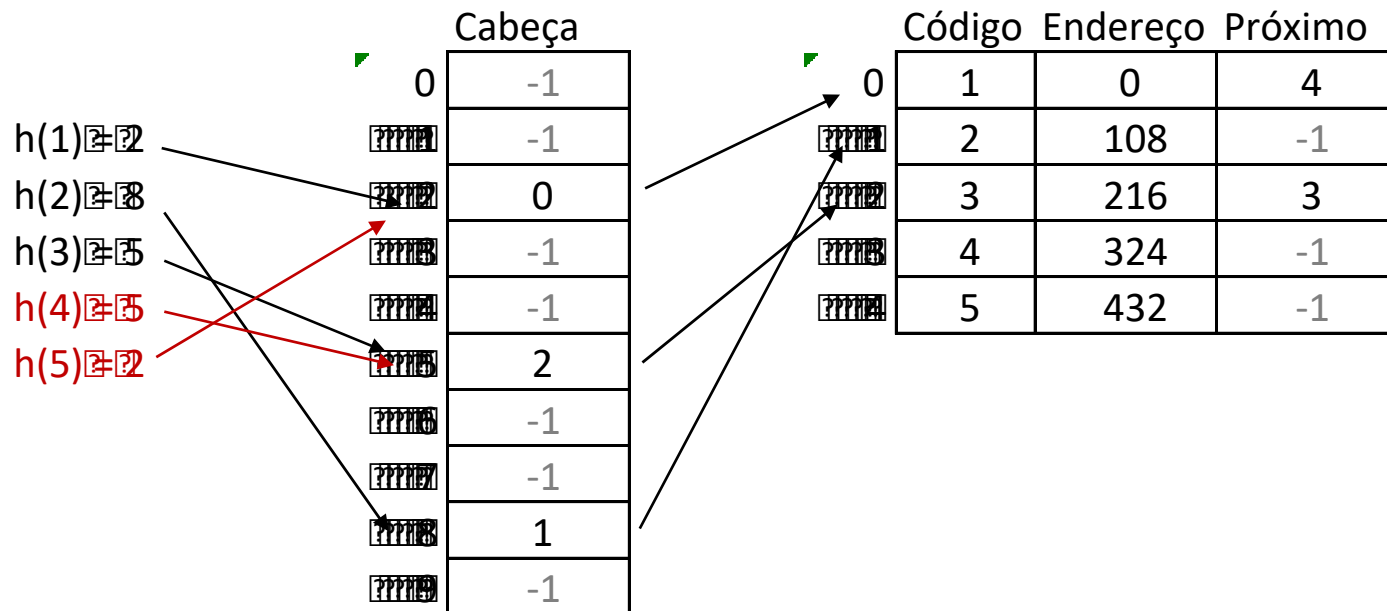
- Área de extensão – os registros colididos são armazenados em uma área de extensão



	Código	Endereço	Próximo
h(1) 2			-1
h(2) 8			-1
h(3) 5	1	0	11
h(4) 5			-1
h(5) 2	3	216	10
			-1
			-1
	2	108	-1
			-1
	4	324	-1
	5	432	-1

Encadeamento externo

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



Buckets

Buckets (cesto)

- Da mesma forma que no caso da árvore B, é importante otimizar o acesso ao disco.
- Assim, cada *posição* no índice, pode conter mais de uma entrada (ou registro)
- Exemplo:
 - Registro no índice = 12 bytes
 - Setor do HD = 4096 bytes = 341,33 registros

Buckets (cesto)

- 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)

Tabelas *hash* dinâmicas

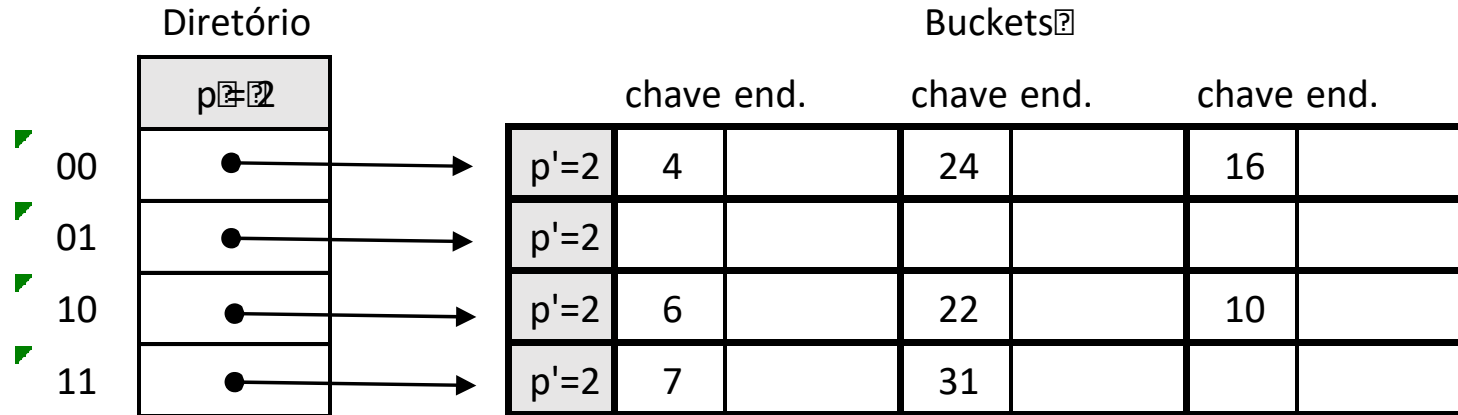
Tabela *hash* dinâmica

- Quando o arquivo de dados cresce ou diminui com frequência (muitas inclusões e exclusões), o índice também precisará ser ajustado.
- Uma tabela *hash* estática, para crescer, precisa reposicionar todos os registros.

Tabela *hash* dinâmica

- Uma tabela *hash* dinâmica é uma tabela *hash* em que apenas alguns registros afetados (aqueles do *bucket*) precisam ser reposicionados.

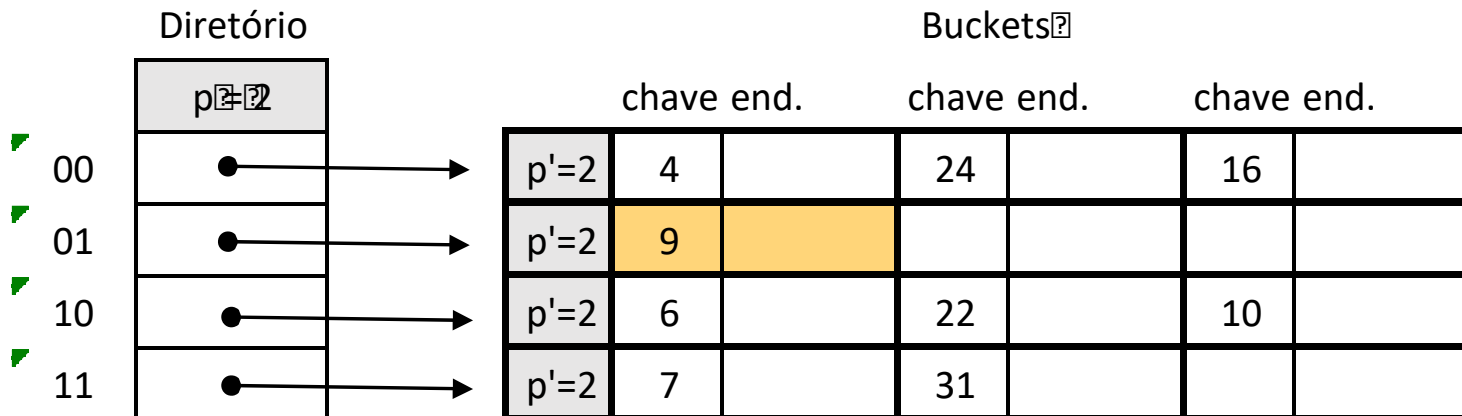
Hash extensível



$$h(k) = k \bmod 2^p$$

Hash extensível

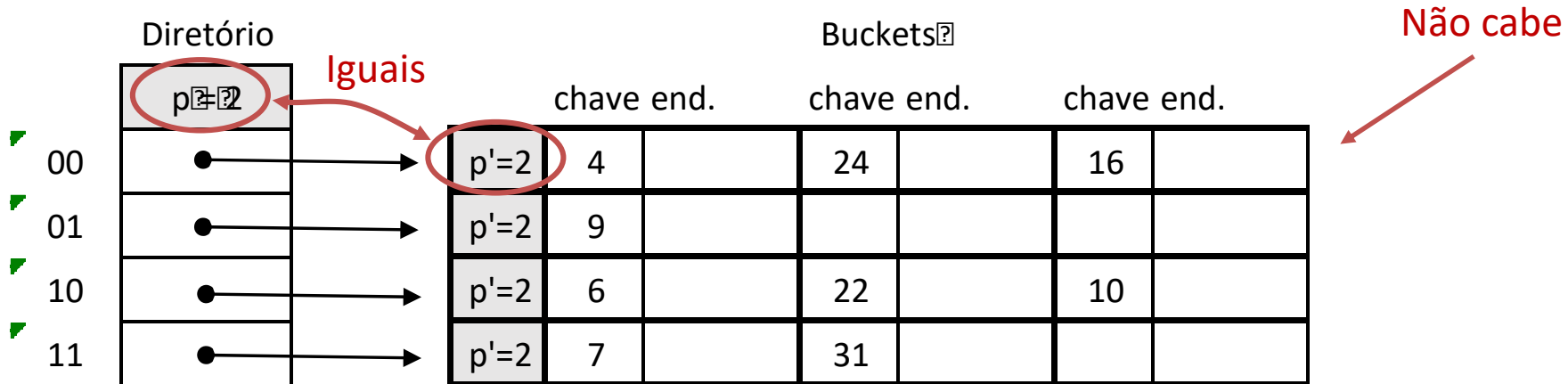
Adicionar chave 9:



$$h(k) = k \bmod 2^p$$

Hash extensível

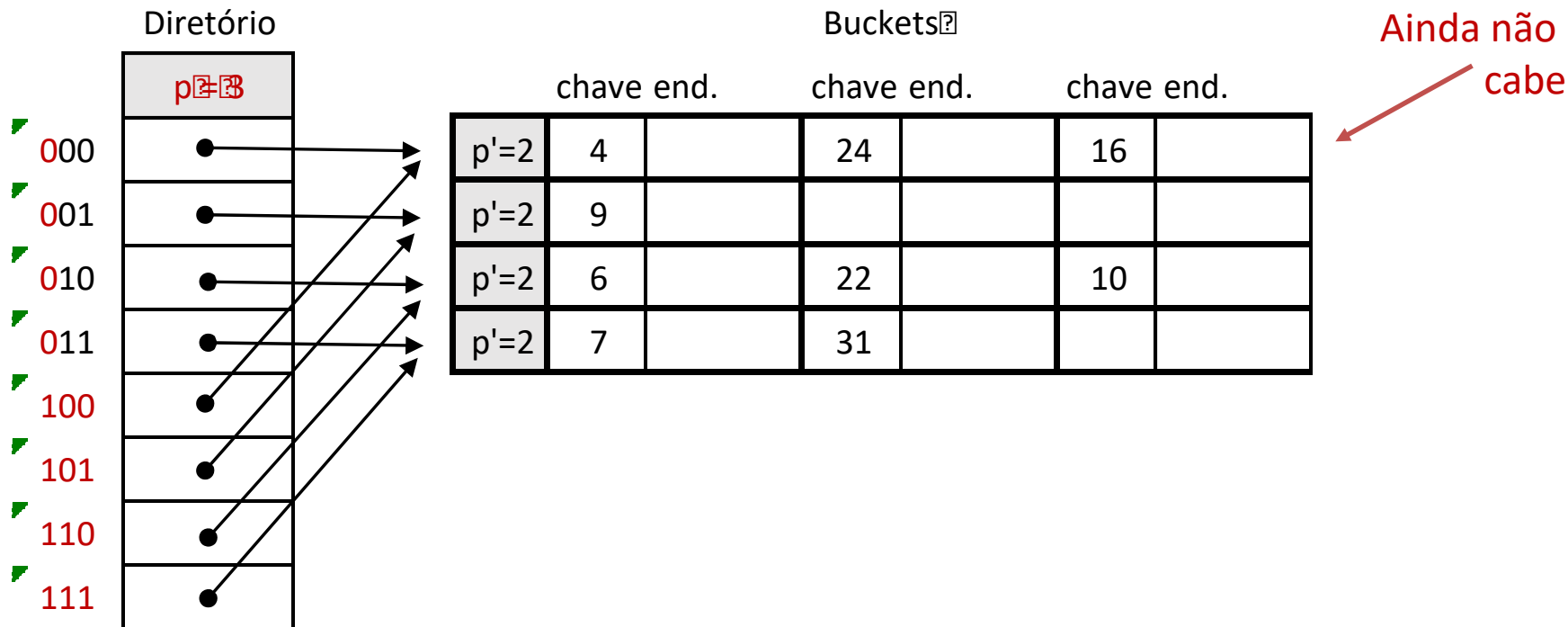
Adicionar chave 20:



$$h(k) = k \bmod 2^p$$

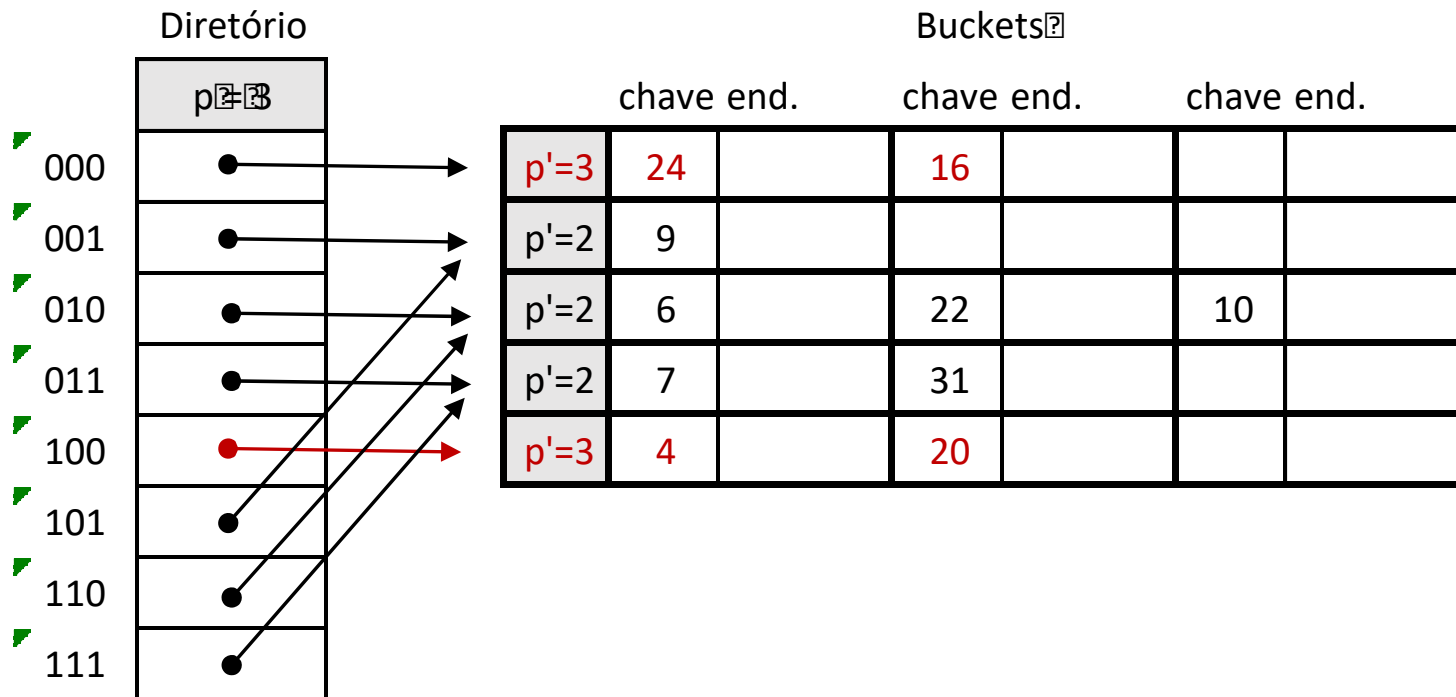
Hash extensível

Adicionar chave 20:



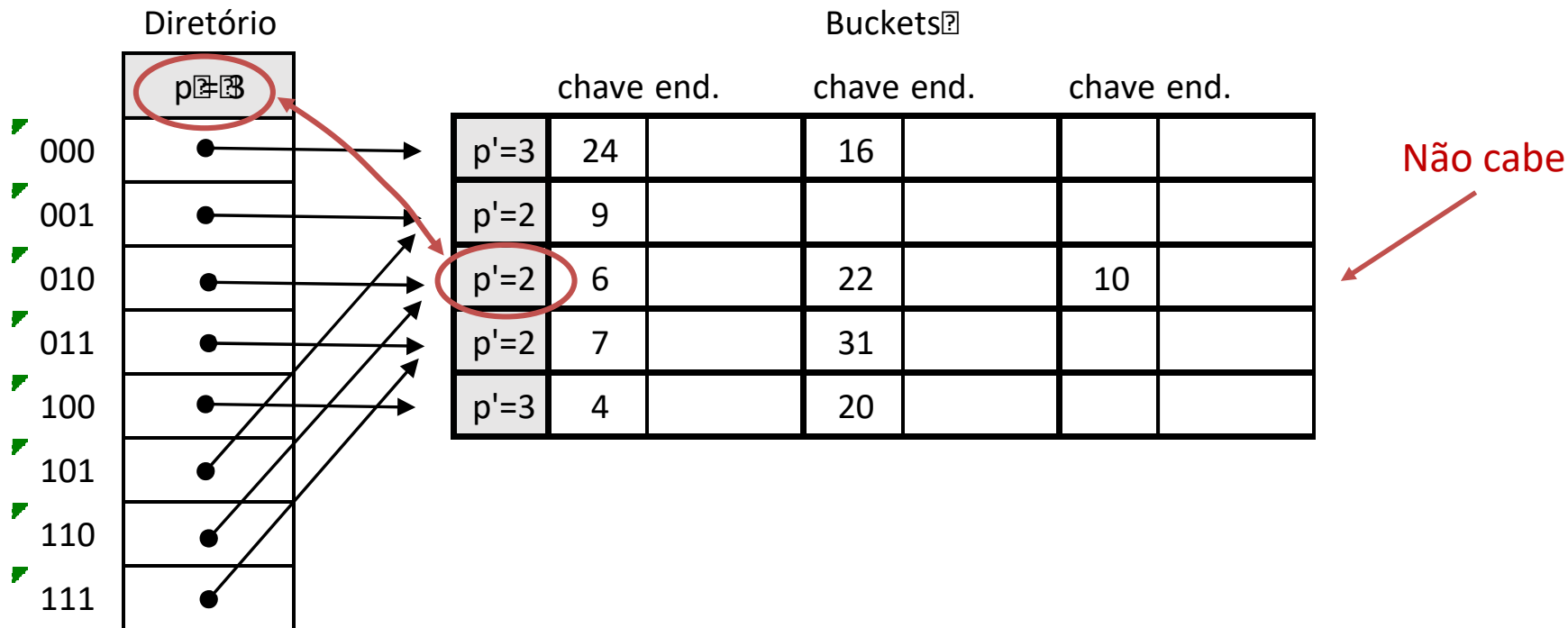
Hash extensível

Adicionar chave 20:



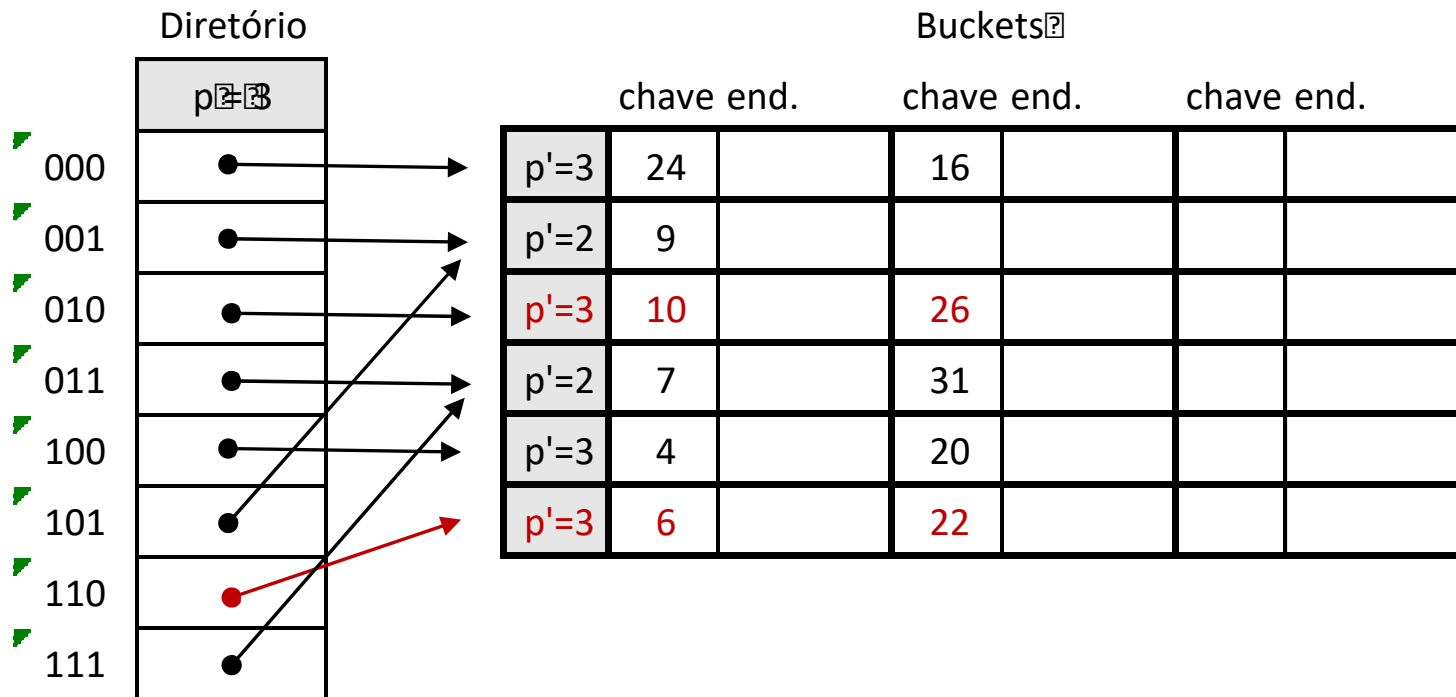
Hash extensível

Adicionar chave 26:



Hash extensível

Adicionar chave 26:



Vantagens do *hash* extensível

- O diretório cresce, sem precisarmos reposicionar todos os registros (do índice)
- O índice (lista de *buckets*) cresce de acordo com a necessidade
- Como não há encadeamento dos *buckets*, não há perda de eficiência