UFSC-CTC-INE

INE5384 - Estruturas de Dados

Métodos de Pesquisa de Dados (III)

Prof. Ronaldo S. Mello

2002/2

Hashing

- Método de pesquisa com o objetivo de melhorar a performance de busca e inclusão de chaves
- Metas para alcançar o objetivo:
  - uso de vetor: acesso direto a uma posição (*O(1)*)
  - uso de uma função (função *hash – h(ch)*) que traduz um valor de chave para uma posição no vetor
    - *h(ch)* deve ter fácil de computar: complexidade *O(1)*

Hashing

ch → *h(ch)* → posição no vetor de índices

Exemplo:

ch = 56 → *h(56)* = 3 ⟶ (acesso direto)

Tabela Hash

0	37	end(37)
1	12	end(12)
2	1	end(1)
3	56	end(56)
. . .	. . .	. . .
M-1	4	end(4)

Características

- Tamanho do vetor: M
  - valor estimado: número médio de chaves
  - na prática, M é menor que o número máximo de chaves possível (M < n)
    - evitar desperdício de espaço!
  - exemplo: chave é código do empregado → numérico com 4 dígitos
    - **n**: 0000-9999 → 10000 chaves possíveis
    - **M**: 2000 → média de empregados na empresa

Características

- Função *hash*:
  - $h(ch) \rightarrow \{0, 1, ..., M - 1\}$
- Como M < n, podem ocorrer colisões!
  - $h(ch_x) = h(ch_y) = i$
- Como minimizar colisões?
  - *h(ch)* deve gerar uma distribuição mais uniforme possível das chaves

Funções *Hash*

- Funções *hash* típicas:
  - operam sobre o resto da divisão por M (ch *mod* M)
  - M é um número primo (poucas divisões exatas!)
- Exemplo:

n = 2500 M = 1031 *h(ch) = ch mod M*

h(23) = 23
h(24) = 24
. . .
h(1401) = 70
h(1402) = 71
. . .

cada chave ocupa uma posição consecutiva no vetor *(boa distribuição!)*

Funções Hash

- Chaves grandes:
  - Exemplo: *ch* é o RG
  - Média de chaves reais é muito menor que o máximo permitido (M << n)
    - Exemplo: no máximo 50 empregados na empresa
  - Usar técnicas para reduzir o escopo

    {1, ..., n} → {0, ..., M-1}
  - Exemplo: somar os dígitos

    RG: 0000000000-9999999999

    0.10 a 9.10 → [0, 90] ⇒ *h(ch) = somaDígitos(ch) MOD 89*

Funções *Hash*

- Chaves não-numéricas:
  - Aplicar duas funções:
    - *f(ch)*: converte *ch* para uma representação numérica
    - *g(ch)*: converte *f(ch)* para o intervalo [0, ..., M-1]
  - *h(ch) = g(f(ch))*

Chaves Não-Numéricas

- Técnicas para *f(ch)*:
  - Soma dos códigos numéricos dos caracteres
    - Exemplo: A N A S I L V A

      f(ch) = int(A)+int(N)+ ...+int(V)+int(A)

      g(ch) = f(ch) mod M
  - Considera apenas alguns caracteres
    - Exemplo: A N A_S I L V A (multiplica 1º, 5º e 8º)

      f(ch) = int(A).int(S).int(V), se int(ch) = ASCII(ch) Þ

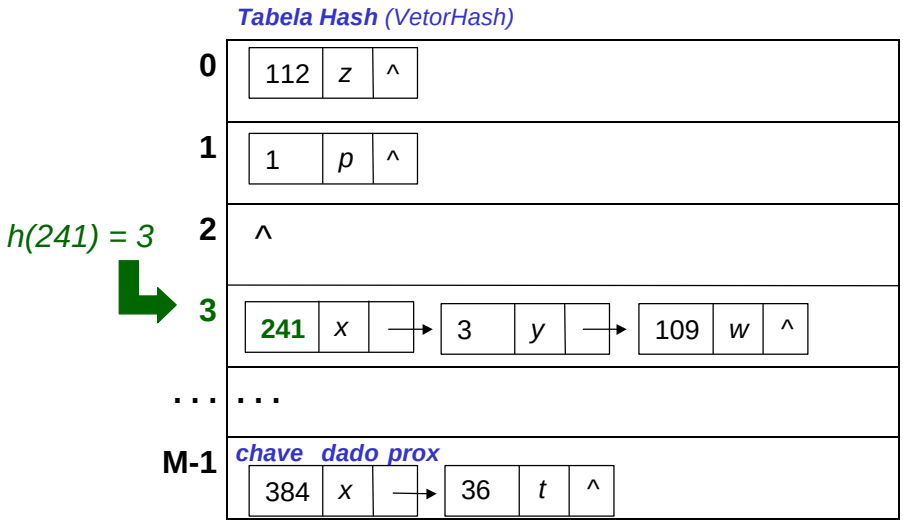
      máximo: 3.ASCII(z) = 3.122 = 366

      g(ch) = f(ch) mod 367

Tabela *Hash* - Implementação

- **Encadeamento por Posição**
- Encadeamento no Vetor

Encadeamento por Posição



Implementação

```
Classe TabelaHash
início
    VetorHash ListaEncadeada[ ]; /* elemento da lista = (chave, dado, prox) */
    nroChaves inteiro;

    construtor TabelaHash (M inteiro);
    início
            i inteiro;
            se M < 1 então Exceção TamanhoInválido();
            VetorHash ← NOVO ListaEncadeada[M];
            nroChaves ← 0;
            para i de 0 até M-1 faça VetorHash[i] ← NOVO ListaEncadeada;
    fim;
    método obtémM() retorna inteiro;
    início
            retorna VetorHash.lenght;
    fim
fim;
```

Implementação

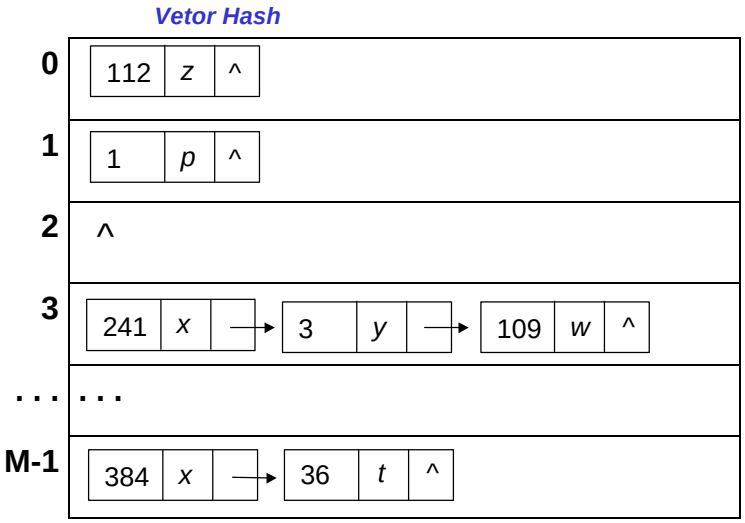
```
Classe TabelaHash
início

    método h(ch Object) retorna inteiro;
    início
          retorna f(ch) mod obtémM();
    fim

    método f(ch Object) retorna inteiro; /* se ch não é numérica */
    início
          ...
    fim
   . . .
fim;
```

Inclusão

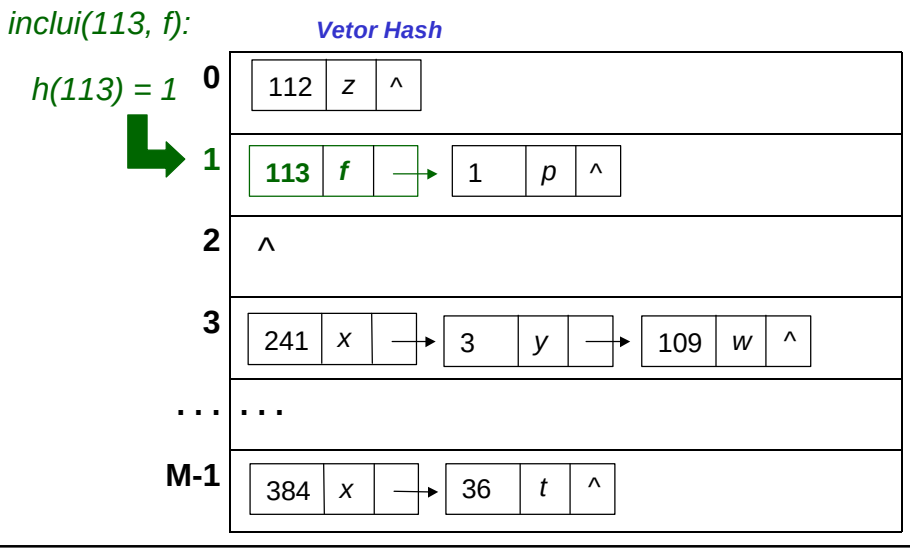
- Como proceder?



Inclusão

- Como proceder?
  - Exemplo: ch = 113
  - Determina *posição* aplicando *h(ch)* (*h(113)* = 1)
  - Insere no início da lista encadeada em *VetorHash[posição]*
  - Incrementa *nroChaves*
  - Complexidade: *O(1)*

Exemplo



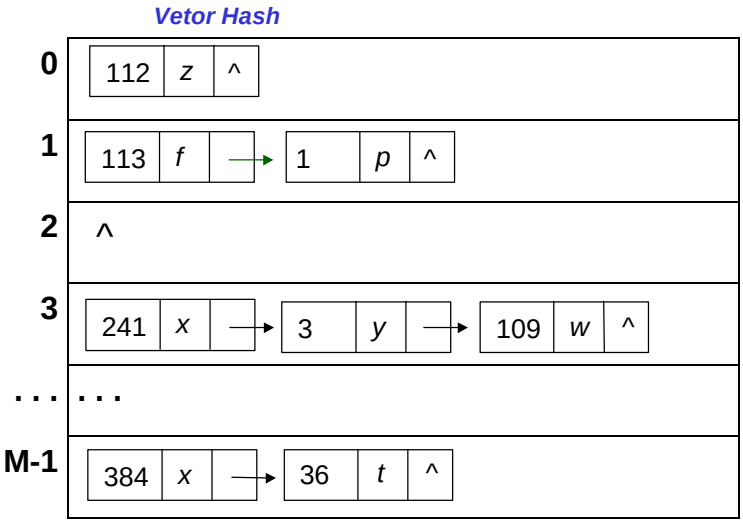
Implementação

```
Método inclui(ch object, d object);
início
    pos inteiro;

    pos ← h(ch);
    VetorHash[pos].insereNoInício(ch, d);
    nroChaves ← nroChaves + 1;
fim;
```

Exclusão

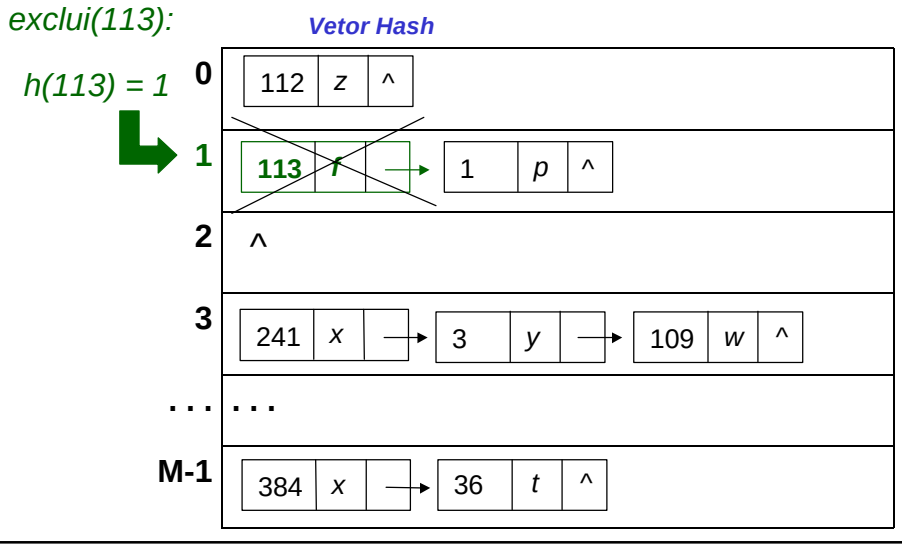
- Como proceder?



Exclusão

- Como proceder?
  - Exemplo: ch = 113
  - Se existirem chaves:
    - determina *posição* (aplica *h(ch)*)
    - exclui elemento da lista em *Vetor[posição]* (se existir)
    - decrementa *nroChaves*
  - Complexidade:
    - *pior caso (pouco provável)*: todas as chaves estão na mesma posição ⇒ *O(n)*
    - *caso médio*: as chaves estão distribuídas entre as posições ⇒ *O(n / M)*

Exemplo



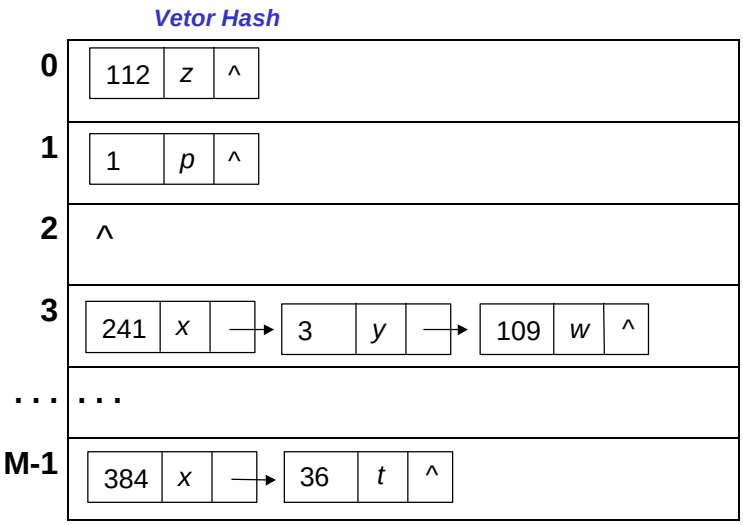
Implementação

```
Método exclui(ch object);
início
    pos inteiro;

    se nroChaves = 0 então Exceção VetorVazio();
    pos ← h(ch);
    VetorHash[pos].exclui(ch);
    nroChaves ← nroChaves - 1;
fim;
```

Pesquisa

- Como proceder?



Pesquisa

- Como proceder?
  - Exemplo: ch = 109
  - Se existirem chaves:
    - determina *posição* (aplica *h(ch)*)
    - busca elemento da lista em *Vetor[posição]* (se existir) e retorna o dado
  - Complexidade:
    - *pior caso (pouco provável)*: todas as chaves estão na mesma posição ⇒ *O(n)*
    - *caso médio*: as chaves estão distribuídas entre as posições ⇒ *O(n / M)*

Implementação

```
Método pesquisa(ch object) retorna Object;
início
    pos inteiro;
    ptr Elemento;

    se nroChaves = 0 então Exceção VetorVazio();
    pos ← h(ch);
    ptr ← VetorHash[pos].obtémPrimeiroElemento();
    enquanto ptr <> NULL faça
    início
          se ptr.chave = ch então retorna ptr.dado;
          ptr ← ptr.prox;
    fim;
    Exceção ChaveInexistente();
fim;
```

Problema desta Alternativa

- Desperdício de espaço se chaves tendem a se agrupar em determinadas posições

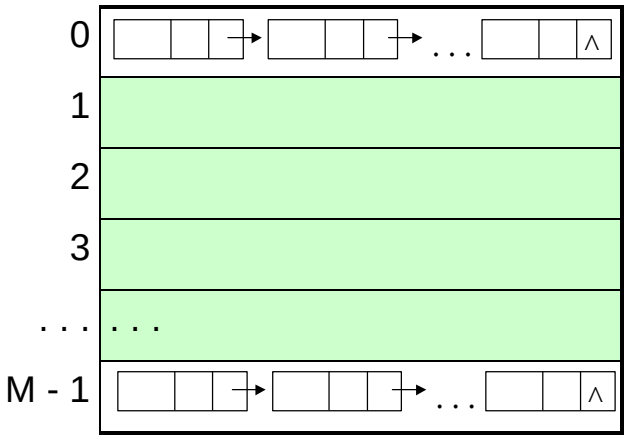
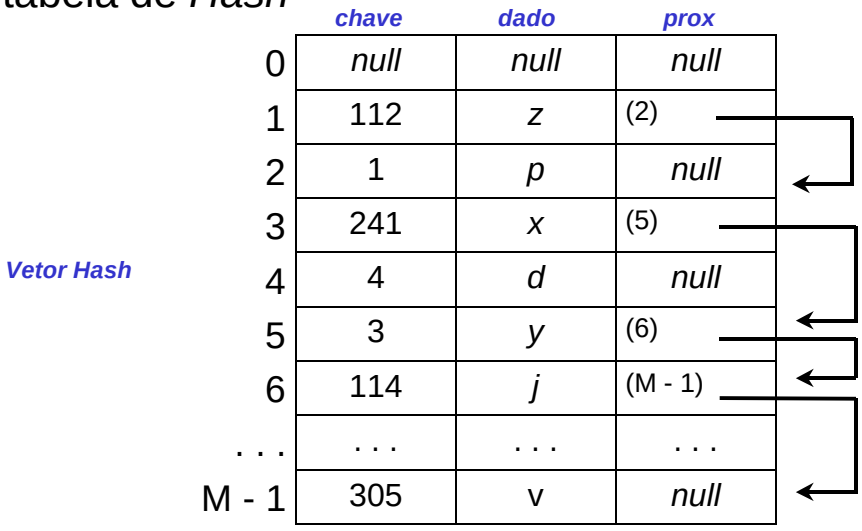


Tabela *Hash* - Implementação

- Encadeamento por Posição
- **Encadeamento no Vetor**

Encadeamento no Vetor

- Aproveita melhor os espaços livres da tabela de *Hash*



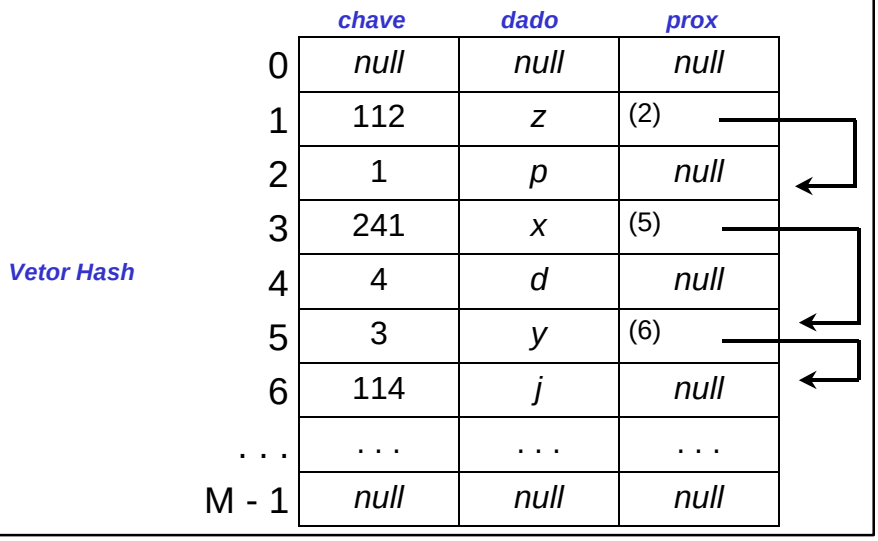
Implementação

```
Classe TabelaHashEncadeada
início
    VetorHash Elemento[ ]; /* elemento da lista = (chave, dado, prox) */
    nroChaves inteiro;

    construtor TabelaHashEncadeada (M inteiro);
    início
            i inteiro;
            se M < 1 então Exceção TamanhoInválido();
            VetorHash ← NOVO Elemento[M];
            nroChaves ← 0;
            para i de 0 até M-1 faça VetorHash[i] ← NOVO Elemento;
    fim;
    método obtémM() retorna inteiro;
    início
            retorna VetorHash.lenght;
    fim
fim;
```

Inclusão

- Como proceder?



Inclusão

- Como proceder?
  - Exemplo: ch = 305
  - Se vetor não está cheio:
    - determina *posição* aplicando *h(ch)* (*h(305)* = 3)
    - se posição "natural" está livre, então inclui
    - senão (colisão)
      - percorre encadeamento de chaves até o último elemento
      - busca primeira posição livre após o último elemento *(busca circular)* e inclui a chave, mantendo o encadeamento correto
    - incrementa *nroChaves*
  - Complexidade (pior caso): *O(M)*

Exemplo

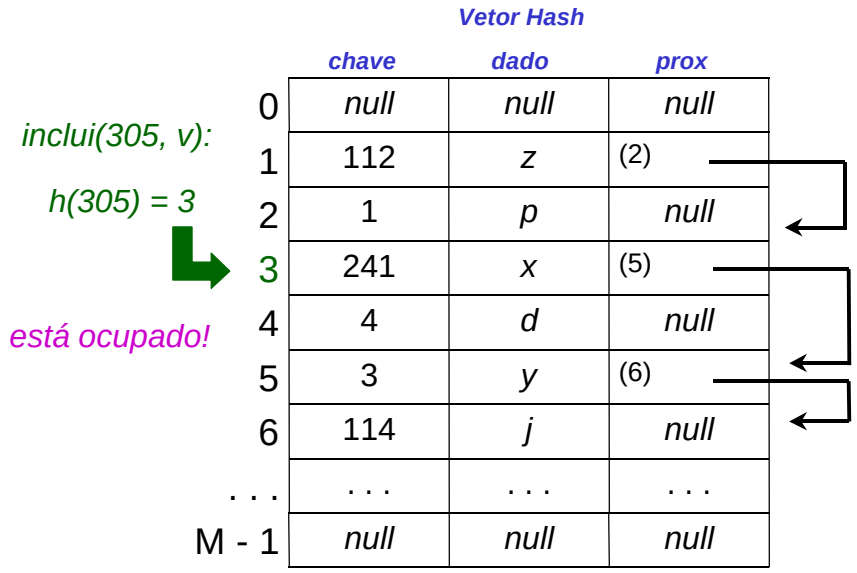
inclui(305, v):

h(305) = 3

está ocupado!

Vetor Hash

	chave	*dado*	*prox*
0	*null*	*null*	*null*
1	112	*z*	(2)
2	1	*p*	*null*
3	241	*x*	(5)
4	4	*d*	*null*
5	3	*y*	(6)
6	114	*j*	*null*
. . .	. . .	. . .	. . .
M - 1	*null*	*null*	*null*



Exemplo

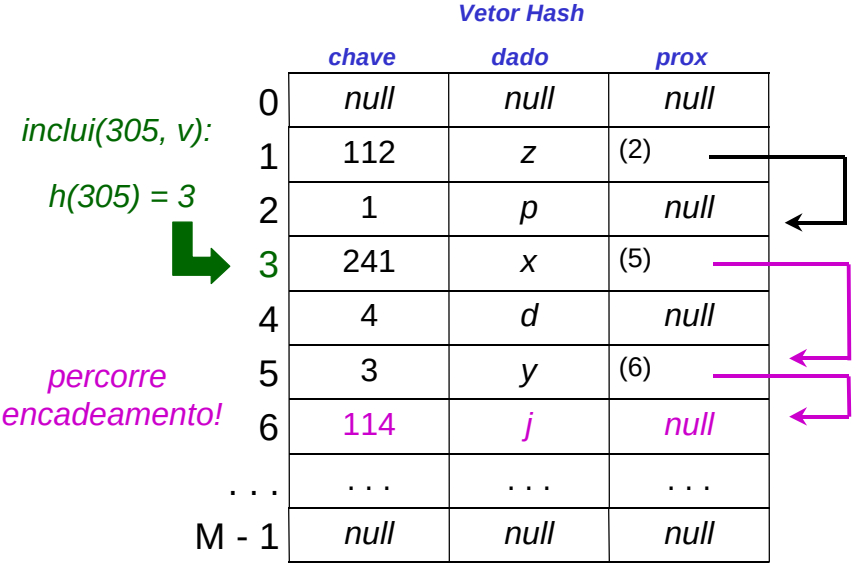
inclui(305, v):

h(305) = 3

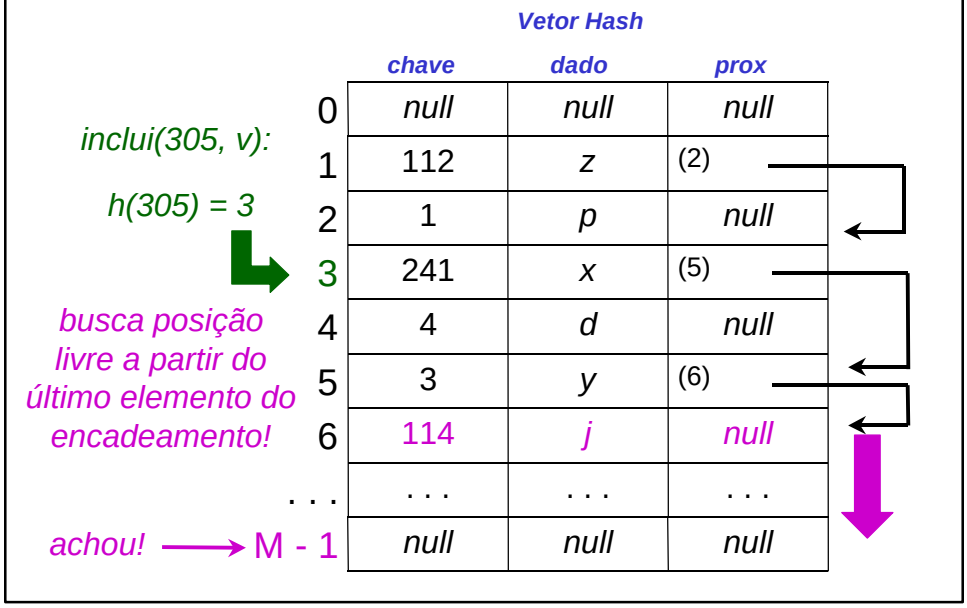
percorre encadeamento!

Vetor Hash

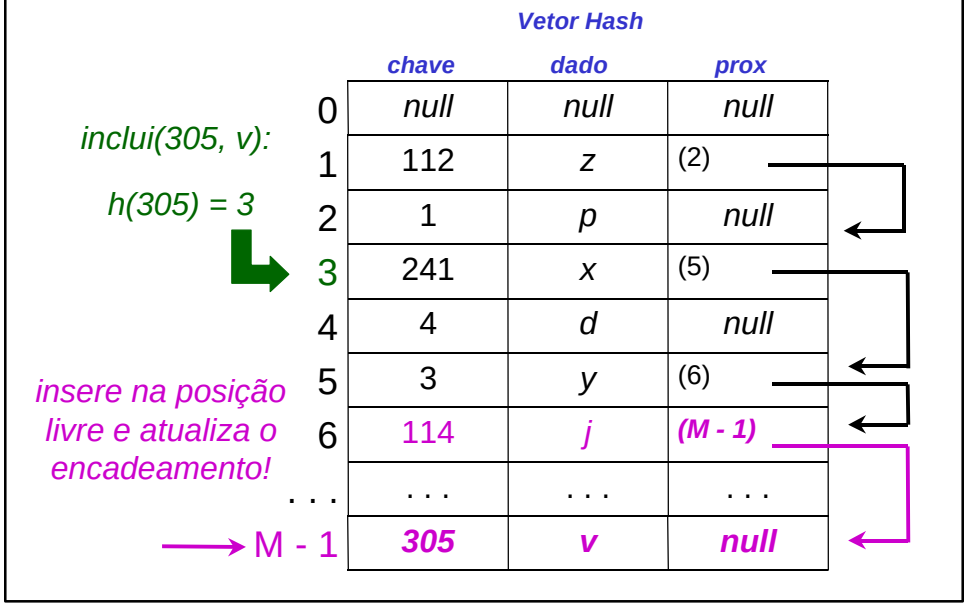
	chave	*dado*	*prox*
0	*null*	*null*	*null*
1	112	*z*	(2)
2	1	*p*	*null*
3	241	*x*	(5)
4	4	*d*	*null*
5	3	*y*	(6)
6	114	*j*	*null*
. . .	. . .	. . .	. . .
M - 1	*null*	*null*	*null*



Exemplo



Exemplo



Pesquisa

- Como proceder?

Vetor Hash

	chave	*dado*	*prox*
0	*null*	*null*	*null*
1	112	*z*	(2)
2	1	*p*	*null*
3	241	*x*	(5)
4	4	*d*	*null*
5	3	*y*	(6)
6	114	*j*	*(M - 1)*
. . .	. . .	. . .	. . .
M - 1	*305*	*v*	*null*

Pesquisa

- Como proceder?
  - Exemplo: ch = 305
  - Se vetor possui chaves:
    - determina *posição* aplicando *h(ch)* (*h(305)* = 3)
    - se posição "natural" é a chave, então retorna dado
    - senão percorre encadeamento de chaves:
      - se achou retorna dado
      - se encadeamento terminou: exceção ChaveInexistente()
  - Complexidade (pior caso): *O(M)*

Implementação

```
Método pesquisa(ch object) retorna Object;
início
    pos inteiro;

    se nroChaves = 0 então Exceção VetorVazio();
    pos ← h(ch);
    enquanto VetorHash[pos].chave <> ch e
    VetorHash[pos].prox <> NULL faça
          pos ← VetorHash[pos].prox;
    se VetorHash[pos].chave = ch então
          retorna VetorHash[pos].dado
    senão Exceção ChaveInexistente();
fim;
```

Exercícios

- Implementar na classe *TabelaHashEncadeada*:
  - inclui(ch Object, d Object);

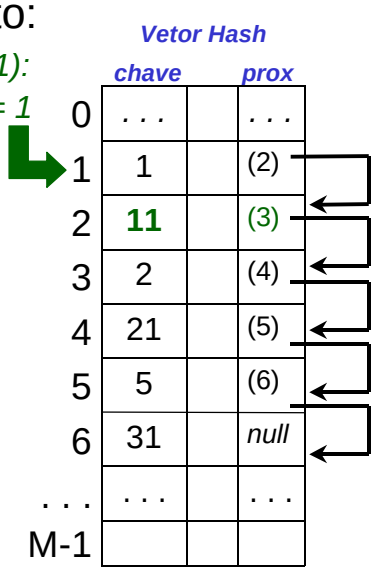
- Implementar na classe *TabelaHash*:
  - percentualOcupação() retorna real;
    /* qual a porcentagem de ocupação da tabela */
  - nroColisões(col inteiro) retorna inteiro;
    /* qtas posições da tabela estão com número de chaves maior ou igual a *col*. *col* deve ser maior que 1 */

- Implementar em uma classe fictícia *Hash*:
  - aumentaTabela(t TabelaHash, novoM inteiro) retorna TabelaHash;
    /* cria uma tabela hash maior que t (novoM > M) e reinsere as chaves nesta nova tabela. A tabela antiga deve ser esvaziada */

Exclusão

- Princípio de funcionamento:

exclui(11):
h(11) = 1



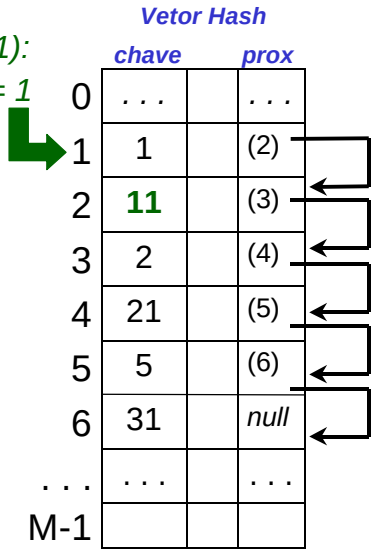
• Quando uma chave ***x*** é removida, deslocam-se as demais chaves no vetor de modo a ocupar a posição vaga da chave ***x***

• Neste exemplo, supor que:
- *h(1) = h(11) = h(21) = h(31) = 1*
- *h(2) = 2*
- *h(5) = 5*

Exclusão

- Como proceder?

exclui(11):
h(11) = 1



•***Temos que lidar com o seguinte problema:*** *chaves com valores de hash diferentes podem estar misturadas em um mesmo encadeamento!*
Neste exemplo: *chaves com hash 1, 2 e 5 estão juntas!*

• a retirada de uma chave da tabela não pode deixar outra chave em um encadeamento inválido!

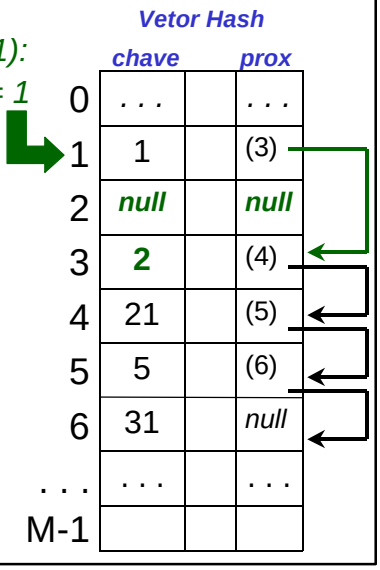
Exclusão

- Como proceder?

exclui(11):
h(11) = 1

• Por quê não liberar simplesmente a posição, atualizando o encadeamento?

Por que chaves podem perder sua posição hash original
Exemplo: chave 2!



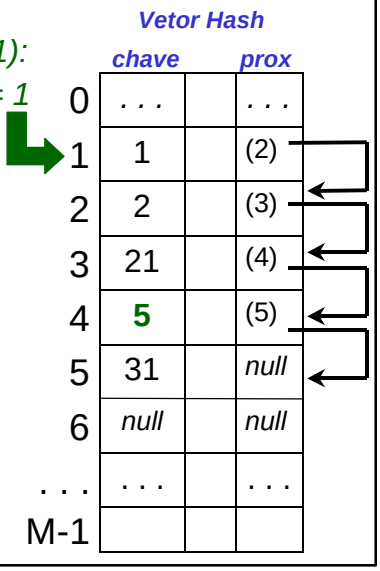
Exclusão

- Como proceder?

exclui(11):
h(11) = 1

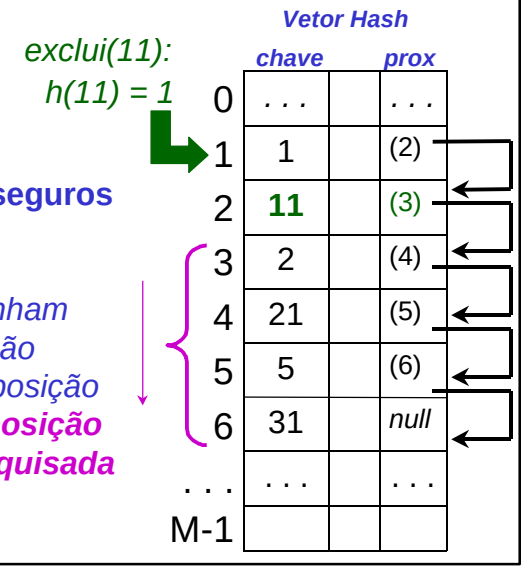
• Qual o problema em deslocar todas as chaves uma posição "para trás"?

Por que chaves também podem perder sua posição hash original
Exemplo: chave 5!



Exclusão

- Solução:

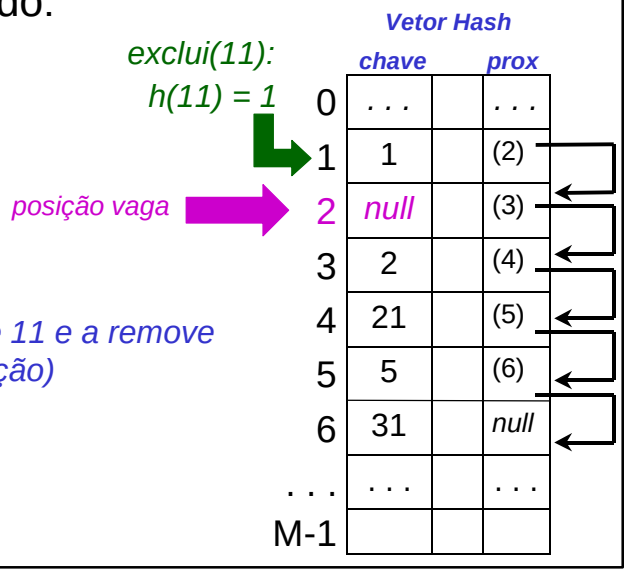


• Apenas **deslocamentos seguros** podem ser feitos, ou seja:

*apenas chaves que **não** tenham valor de hash **entre** a posição imediatamente posterior à posição que deve ser preenchida (**posição vaga + 1**) e a **posição pesquisada** podem ser deslocadas*

Exclusão

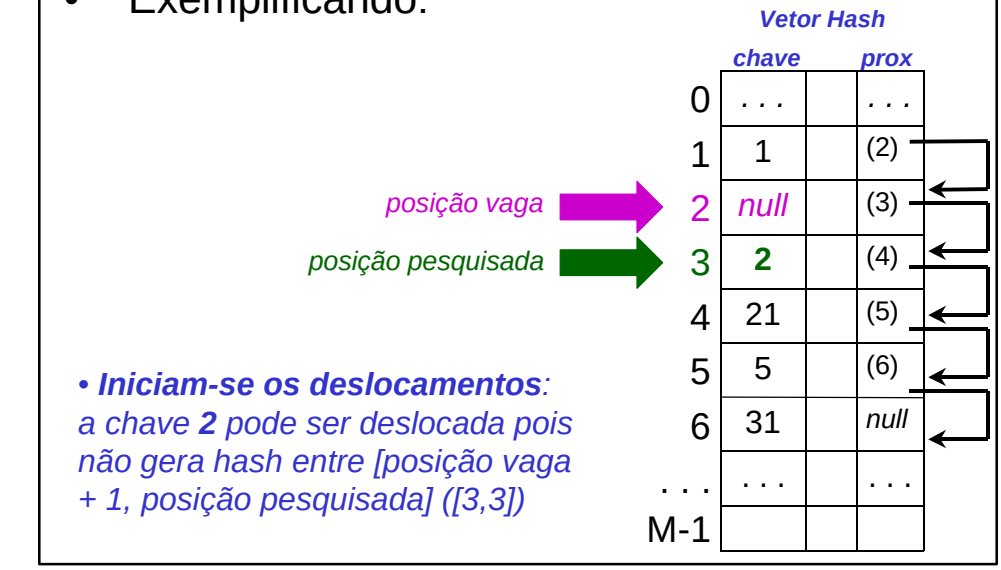
- Exemplificando:



• *Procura-se a chave 11 e a remove (libera-se a sua posição)*

Exclusão

- Exemplificando:

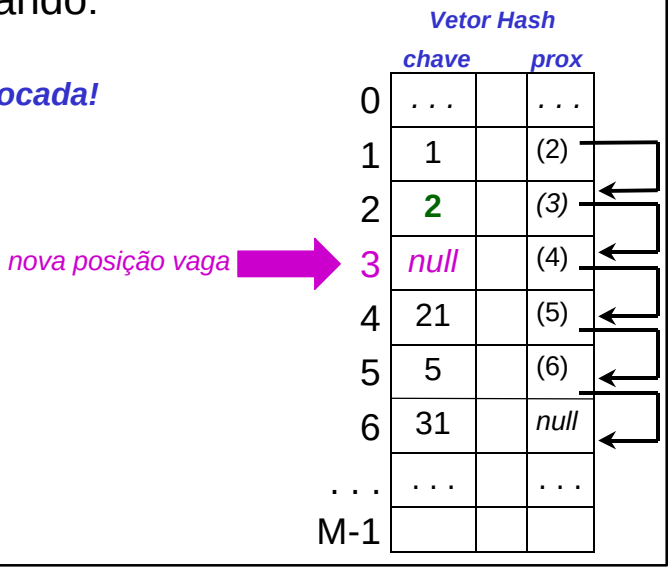


• ***Iniciam-se os deslocamentos***:
*a chave **2** pode ser deslocada pois não gera hash entre [posição vaga + 1, posição pesquisada] ([3,3])*

Exclusão

- Exemplificando:

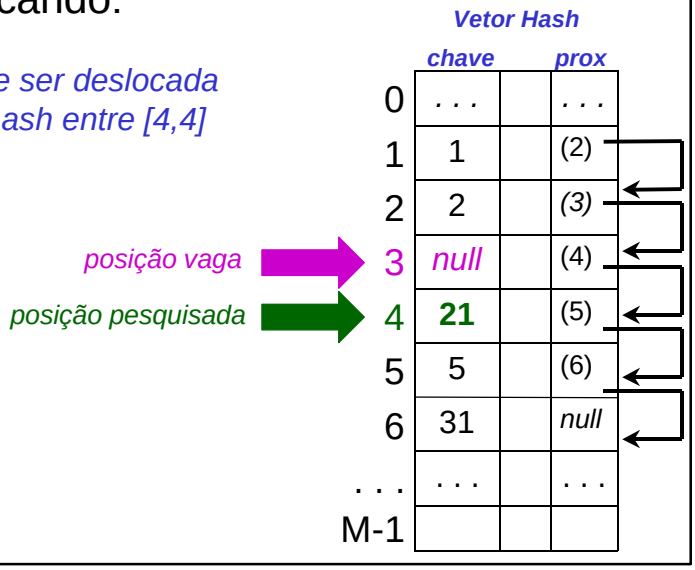
• ***Chave 2 deslocada!***



Exclusão

- Exemplificando:

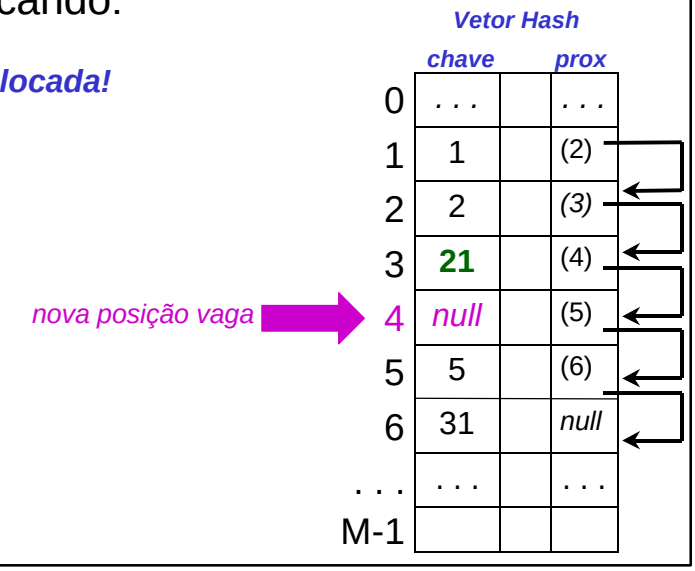
• *chave **21** pode ser deslocada pois não gera hash entre [4,4]*



Exclusão

- Exemplificando:

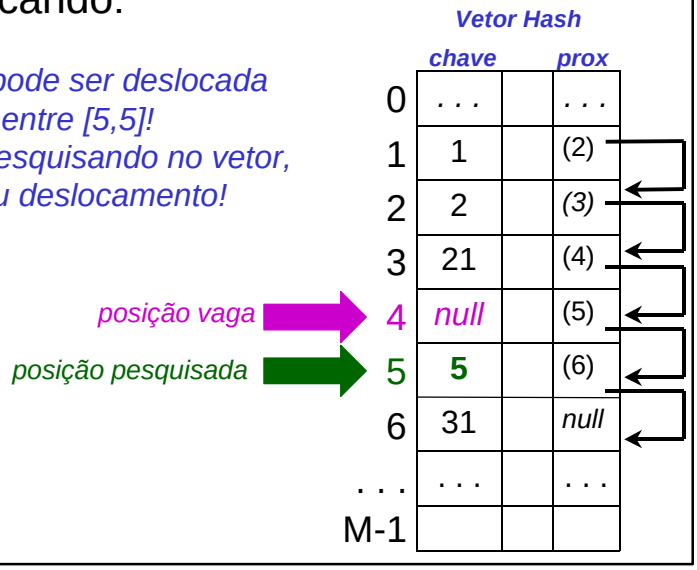
• ***chave 21 deslocada!***



Exclusão

- Exemplificando:

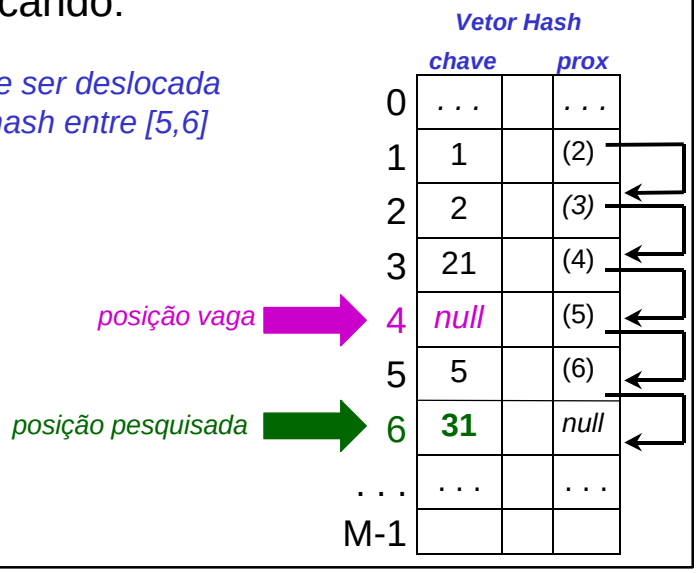
*• chave **5 não** pode ser deslocada pois gera hash entre [5,5]!*
• continua-se pesquisando no vetor, sem fazer o seu deslocamento!



Exclusão

- Exemplificando:

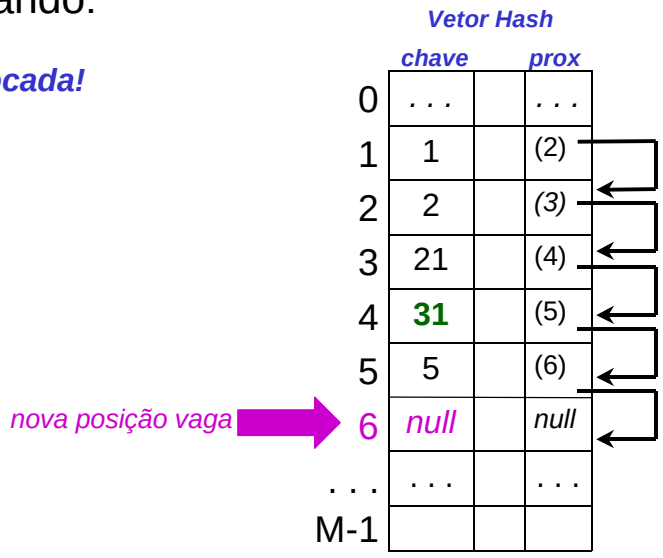
*• chave **31** pode ser deslocada pois não gera hash entre [5,6]*



Exclusão

- Exemplificando:

  - ***chave 31 deslocada!***

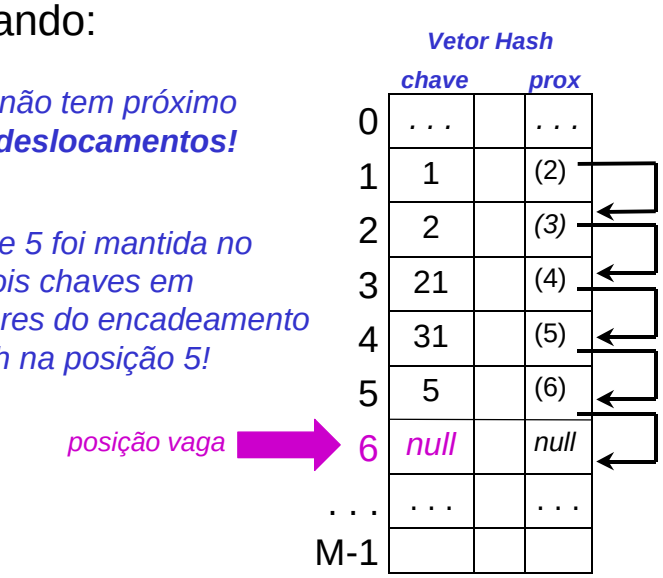


Exclusão

- Exemplificando:

  - *a posição vaga não tem próximo*
    ® ***terminam os deslocamentos!***

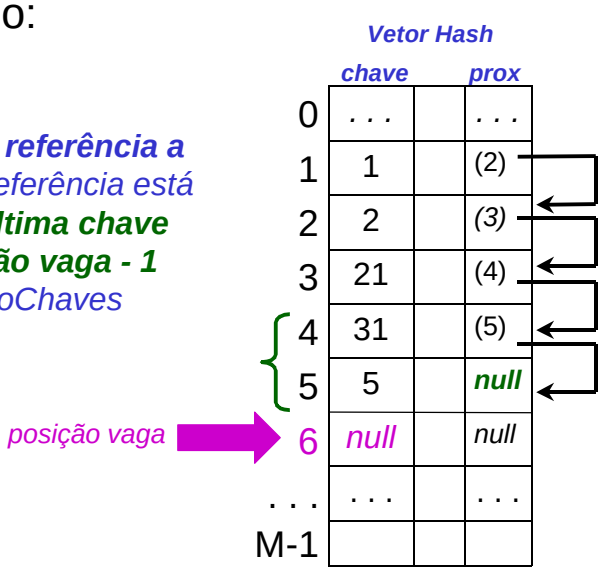
  - *note que a chave 5 foi mantida no encadeamento pois chaves em posições posteriores do encadeamento podem gerar hash na posição 5!*



Exclusão

- Exemplificando:

• *para concluir:*
*- **deve-se anular a referência a posição vaga:** esta referência está entre a **posição da última chave deslocada** e a **posição vaga - 1***
- decrementa-se nroChaves



Exclusão

- Complexidade (pior caso):
  - pesquisa pela chave: ***O(M)***
  - deslocamentos:
    - único encadeamento
    - nenhuma chave pode ser deslocada de forma segura

    → teste para deslocamento seguro em cada posição:

    $M \cdot i$

    → *i* em média = (M-1)/2

    → M . (M-1)/2 = $O(M^2)$