

# Aula anterior

- Árvore 2-3-4
  - Introdução e Pesquisa
  - Inserção com fragmentação por ascensão
  - Inserção com fragmentação na descida

# Tabela Hash

---

Prof. Diego Silva Caldeira Rocha

---

# Objetivos

- Introdução a Tabela hash
- Função hash
- Colisão
- Gerenciamento de Colisão
- Mapeamento

- A tabela *hash* é uma estrutura de dados em que um ou mais elementos podem ser acessados com  $\Theta(1)$
- Vantagens:
  - Alta eficiência no custo de pesquisa, que é  $\Theta(1)$  para o caso médio
  - Simplicidade de implementação

- Desvantagens:
  - Custo para recuperar os registros na ordem lexicográfica das chaves é alto, sendo necessário ordenar o arquivo
  - Pior caso é  $\Theta(n)$
- Considera-se uma tabela e uma função hash sobre a chave de pesquisa

## Exemplo

- Seja uma tabela com 366 entradas (uma para cada dia do ano), inserimos cada pessoa conforme sua data de nascimento

Aniversário	Pessoa
5/1	Paulo
13/3	Lica
26/5	Olímpia
13/9	João

- Como será nossa função hash?
- $5/1 \Rightarrow$  Posição 4 ( $0 + 5 - 1$ )
- $13/3 \Rightarrow$  Posição 72 ( $31+29+13-1$ )
- $26/5 \Rightarrow$  Posição 146 ( $31+29+31+30+26-1$ )
- $13/9 \Rightarrow$  Posição 256 ( $31+29+31+30+31+30+31+31+13-1$ )

- Como será nossa função hash?

```
int hash (Data d) throws Exception {  
    int resp;  
    if (d.invalida() == true){        throw new Exception ("Erro mês invalido!"); }  
    else if (d.mes == 1){              resp = d.dia - 1; }  
    else if (d.mes == 2){              resp = 31 + d.dia - 1; }  
    else if (d.mes == 3){              resp = 60 + d.dia - 1; }  
    else if (d.mes == 4){              resp = 91 + d.dia - 1; }  
    else if (d.mes == 5){              resp = 121 + d.dia - 1; }  
    else if (d.mes == 6){              resp = 152 + d.dia - 1; }  
    else if (d.mes == 7){              resp = 182 + d.dia - 1; }  
    else if (d.mes == 8){              resp = 213 + d.dia - 1; }  
    else if (d.mes == 9){              resp = 244 + d.dia - 1; }  
    else if (d.mes == 10){             resp = 274 + d.dia - 1; }  
    else if (d.mes == 11){             resp = 305 + d.dia - 1; }  
    else {                             resp = 335 + d.dia - 1; }  
    return resp;  
}
```



## Exemplo

- Seja uma tabela com 366 entradas, uma para cada dia do ano, e vamos inserir pessoas conforme a data de nascimento das mesmas

Aniversário	Pessoa
5/1	Paulo
13/3	Lica
26/5	Olímpia
13/9	João

- Qual é o custo de encontrar uma pessoa?
- Algum problema em nossa técnica? Se sim, qual é a chance desse problema acontecer?

Tabela	
0	
1	
2	
3	
4	Paulo
...	
72	Lica
...	
146	Olímpia
...	
256	João
...	

# Paradoxo do Aniversário

- Em um grupo de 23 ou mais pessoas quaisquer existe uma chance maior do que 50% de que duas delas comemorem aniversário no mesmo dia
- Em outras palavras, se tivermos uma função hash uniforme para endereçar 23 chaves randômicas em uma tabela de tamanho 366, tem-se que a probabilidade de acontecer colisões é maior do que 50%
- A probabilidade de se inserir  $n$  itens consecutivos sem colisão em uma tabela de tamanho  $m$  é:

$$\frac{m!}{(m-n)!m^n}$$

# Função Hash

- Mapeia chaves em inteiros  $[0, m-1]$ , onde  $m$  é o tamanho da tabela
- Deve ser simples de ser computada
- As chaves de pesquisa devem ser distribuídas de forma uniforme entre as  $m$  entradas possíveis
- Número de comparações nas operações de pesquisa, inserção e remoção depende do tamanho da tabela e da quantidade de elementos inseridos

# Função Hash

- Normalmente, **não** depende do número de itens da coleção
- O valor de  $m$  deve ser escolhido com atenção e uma sugestão é que ele seja um número primo (não qualquer primo)
- As chaves não numéricas devem ser transformadas em números

## Exemplo

- $h(k) = k \% m$ , onde  $k$  é o somatório do código ASCII de todos os caracteres e  $m$ , o tamanho da tabela

**Tabela ASCII (códigos de caracteres 0 - 127)**

000	016 ►	032	048 0	064 @	080 P	096 `	112 p
001 ☺	017 ◀	033 !	049 1	065 A	081 Q	097 a	113 q
002 ☹	018 †	034 "	050 2	066 B	082 R	098 b	114 r
003 ♥	019 !!	035 #	051 3	067 C	083 S	099 c	115 s
004 ♦	020 ¶	036 \$	052 4	068 D	084 T	100 d	116 t
005 ♣	021 §	037 %	053 5	069 E	085 U	101 e	117 u
006 ♠	022 ■	038 &	054 6	070 F	086 V	102 f	118 v
007	023 ‡	039 '	055 7	071 G	087 W	103 g	119 w
008	024 ↑	040 (	056 8	072 H	088 X	104 h	120 x
009	025 ↓	041 )	057 9	073 I	089 Y	105 i	121 y
010	026 →	042 *	058 :	074 J	090 Z	106 j	122 z
011 ♂	027 ←	043 +	059 ;	075 K	091 [	107 k	123 {
012 ♀	028 L	044 ,	060 <	076 L	092 \	108 l	124
013	029 ↔	045 -	061 =	077 M	093 ]	109 m	125 }
014 ♪	030 ▲	046 .	062 >	078 N	094 ^	110 n	126 ~
015 ☼	031 ▼	047 /	063 ?	079 O	095 _	111 o	127 △

## Exemplo

- $h(k) = k \% m$ , onde  $k$  é o somatório do código ASCII de todos os caracteres e  $m$ , o tamanho da tabela

- Assim, fazendo  $m = 13$ , temos:

$$h(\text{oscar}) = 536 \% 13 = 3$$

$$h(\text{bosco}) = 534 \% 13 = 1$$

$$h(\text{elisa}) = 526 \% 13 = 6$$

$$h(\text{maya}) = 424 \% 13 = 8$$

$$h(\text{lucio}) = 540 \% 13 = 7$$

$$h(\text{paulo}) = 545 \% 13 = 12$$

$$h(\text{sofia}) = 530 \% 13 = 10$$

$$h(\text{sergio}) = 649 \% 13 = 12$$

## Exemplo

- Inserimos, pesquisamos ou removemos um item a partir de sua chave de pesquisa:

0	
1	bosco
2	
3	oscar

4	
5	
6	elisa

7	lucio
8	maya
9	

10	sofia
11	
12	paulo

- A maya está ( $h(\text{maya}) = 8$ )
- A ana está ( $h(\text{ana}) = 5$ )
- O sergio está ( $h(\text{sergio}) = 12$ ):
- Como podemos tratar das colisões?

# Colisões Primárias

- Acontecem quando desejamos inserir um elemento em uma posição do *array* que está ocupada por outro elemento
- Na verdade, a função hash dá acesso a um subconjunto de elementos



# Gerenciamento de Colisões

- **Métodos de transformação direta** (resolução por cálculo): Quando há uma colisão, calcula-se uma nova posição no *array* a partir da chave do item considerado
- **Métodos de transformação indireta** (resolução por encadeamento): Os elementos que tiverem o mesmo valor da função são encadeados

# Gerenciamento de Colisões

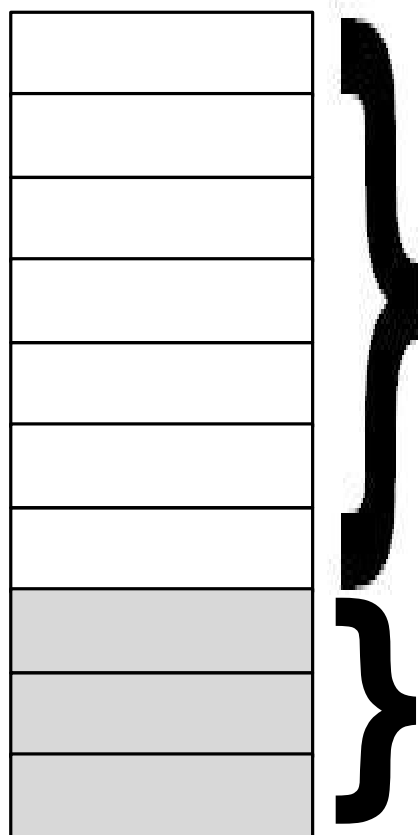
- *Hash* direta com área de reserva (*overflow*)
- *Hash* direta com rehash
- *Hash* indireta com lista flexível simples

# Gerenciamento de Colisões

- *Hash* direta com área de reserva (*overflow*) ←
- *Hash* direta com rehash
- *Hash* indireta com lista flexível simples

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3



Tamanho da tabela

Tamanho da área de reserva

```
int hash (int x){  
    return x % tamTabela;  
}
```

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

livre

0	
1	
2	
3	
4	
5	5
6	
7	
8	
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	
1	
2	
3	
4	
5	5
6	
7	
8	
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

livre

0	
1	
2	
3	
4	
5	5
6	
7	
8	
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19



# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7
1	
2	
3	
4	
5	5
6	
7	
8	
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7	ocupado
1		
2		
3		
4		
5	5	
6		
7		reserva
8		
9		

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

livre

0	7
1	
2	
3	
4	
5	5
6	
7	
8	
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7
1	
2	
3	
4	
5	5
6	
7	14
8	
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7	ocupado
1		
2		
3		
4		
5	5	
6		
7	14	
8		
9		

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7
1	
2	
3	
4	
5	5
6	
7	14
8	
9	

livre

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7
1	
2	
3	
4	
5	5
6	
7	14
8	21
9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

	0	7
	1	
	2	
	3	
	4	
ocupado	5	5
	6	
	7	14
	8	21
	9	

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19



# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7
1	
2	
3	
4	
5	5
6	
7	14
8	21
9	

livre

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

# Hash Direta com Área de Reserva (overflow)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7
1	
2	
3	
4	
5	5
6	
7	14
8	21
9	19

```
int hash (int x){  
    return x % 7;  
}
```

Vamos inserir os números:

5, 7, 14, 21 e 19

## Exercício (1)

- Suponha uma tabela de tamanho 7 com uma área de reserva igual a 3

0	7
1	
2	
3	
4	
5	5
6	
7	14
8	21
9	19

```
int hash (int x){  
    return x % 7;  
}
```

**Exercício:** Quantas comparações são necessárias para pesquisar o:

a) 5

b) 21

c) 19

d) 6

# Exercício Resolvido (1): Método Inserir

```
void inserir(int x) throws Exception {  
  
    int i = hash(x);  
    if (x == NULO){  
        throw new Exception ("Erro!");  
    } else if (tabela[i] == NULO){  
        tabela[i] = x;  
    } else if (numReserva < tamReserva) {  
        tabela[tamTabela + numReserva] = x;  
        numReserva++; //o valor inicial de numReserva é zero  
    } else {  
        throw new Exception ("Erro!");  
    }  
}
```

# Exercício Resolvido (1): Método Pesquisar

```
int pesquisar (int x){  
    int i = hash(x), resp = NULO;  
    if (tabela[i] == x){ resp = i;  
    } else if (tabela[i] != NULO) {  
        for (int i = 0; i < tamReserva; i++){  
            if (tabela[tamTabela + i] == x){  
                resp = tamTabela + i;    i = tamReserva;  
            } } }  
    return resp;  
}
```

h


# Análise de Complexidade

• Como mostrado por Knuth (1973), o custo de uma pesquisa com sucesso é

$C(n) = \frac{1}{2} \left( 1 + \frac{1}{1-\alpha} \right)$ , onde  $\alpha = n/m$  é o fator de carga da tabela

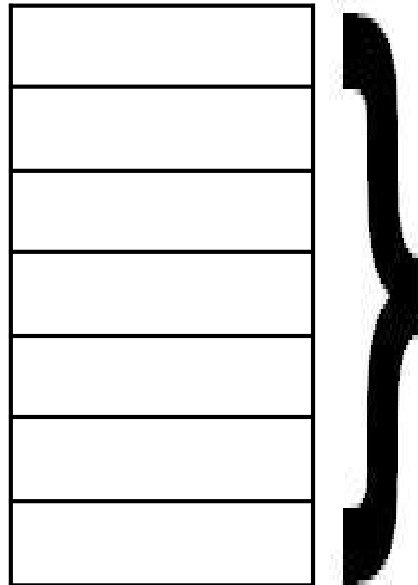
• A tabela hash direta sofre de um mal chamado *clustering* (Knuth, 1973, pp.520-521) no qual o tempo de pesquisa médio aumenta quando a tabela começa a ficar cheia

# Gerenciamento de Colisões

- *Hash* direta com área de reserva (*overflow*)
- ***Hash* direta com rehash** 
- *Hash* indireta com lista flexível simples
- Estrutura híbrida

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7



Tamanho da tabela

```
int hash (int x){  
    return x % tamTabela;  
}
```

```
int rehash (int x){  
    return ++x %  
tamTabela;  
}
```



# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	
1	
2	
3	
4	
5	
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

livre

0	
1	
2	
3	
4	
5	
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	
1	
2	
3	
4	
5	5
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

livre

0	
1	
2	
3	
4	
5	5
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7
1	
2	
3	
4	
5	5
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7	ocupado
1		
2		
3		
4		
5	5	
6		

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

livre

0	7
1	
2	
3	
4	
5	5
6	

```
int rehash (int x){  
    return ++x %  
    tamTabela;  
}
```

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7
1	14
2	
3	
4	
5	5
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1



# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7	ocupado
1	14	
2		
3		
4		
5	5	
6		

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7
1	14
2	
3	
4	
5	5
6	

ocupado

```
int rehash (int x){  
    return ++x %  
    tamTabela;  
}
```

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7
1	14
2	
3	
4	
5	5
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Não é possível inserir o 21.

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

	0	7
	1	14
	2	
	3	
	4	
ocupado	5	5
	6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

livre

0	7
1	14
2	
3	
4	
5	5
6	

```
int rehash (int x){  
    return ++x %  
    tamTabela;  
}
```

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7
1	14
2	
3	
4	
5	5
6	19

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7	ocupado
1	14	
2		
3		
4		
5	5	
6	19	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7
1	14
2	
3	
4	
5	5
6	19

```
int rehash (int x){  
    return ++x %  
    tamTabela;  
}
```

Vamos inserir os números:

5, 7, 14, 21, 19 e 1



# Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

0	7
1	14
2	1
3	
4	
5	5
6	19

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

## Exercício (2)

- Suponha uma tabela de tamanho 7

0	7
1	14
2	1
3	
4	
5	5
6	19

**Exercício:** Quantas comparações são necessárias para pesquisar o:


a) 5

b) 21

c) 19

d) 6

# Gerenciamento de Colisões

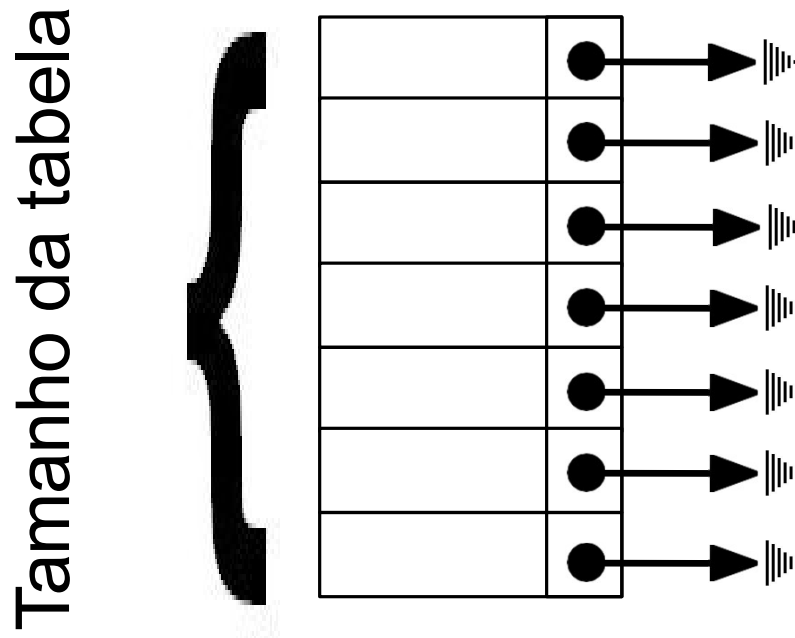
- *Hash* direta com área de reserva (*overflow*)
- *Hash* direta com rehash
- ***Hash* indireta com lista flexível simples** 

# *Hash* Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

# Hash Indireta com Lista Flexível Simples

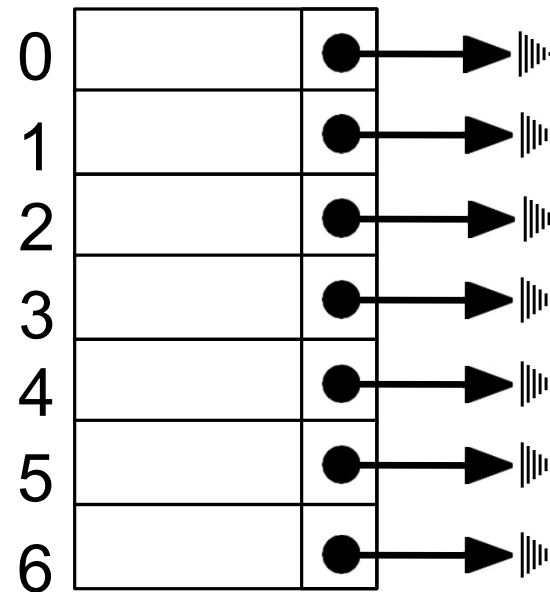
- Suponha uma tabela de tamanho 7



```
int hash (int x){  
    return x % tamTabela;  
}
```

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

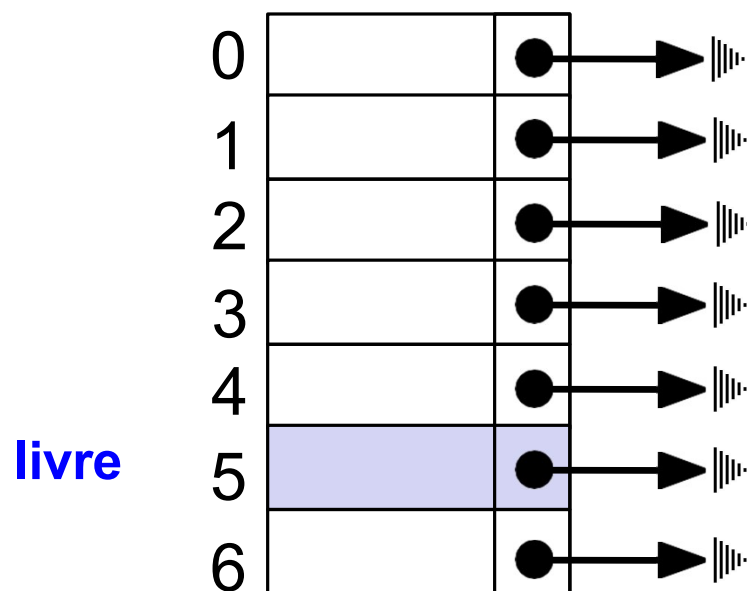


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

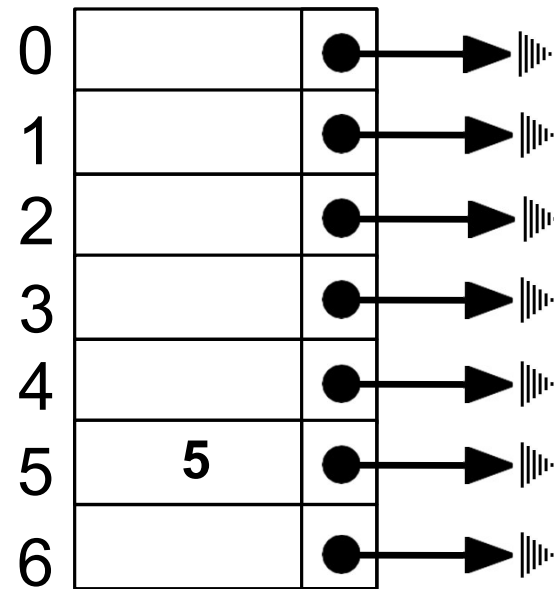


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7



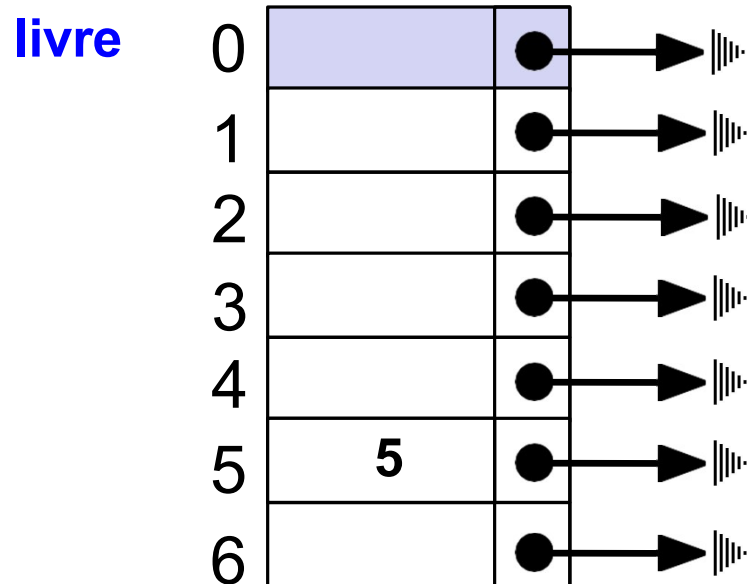
Vamos inserir os números:

5, 7, 14, 21, 19 e 3



# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

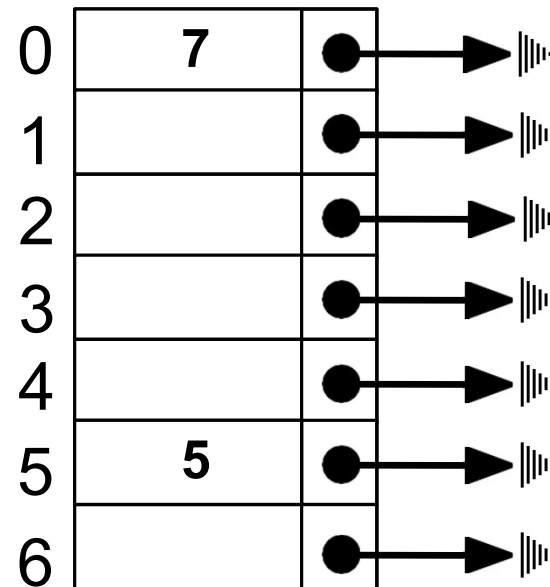


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

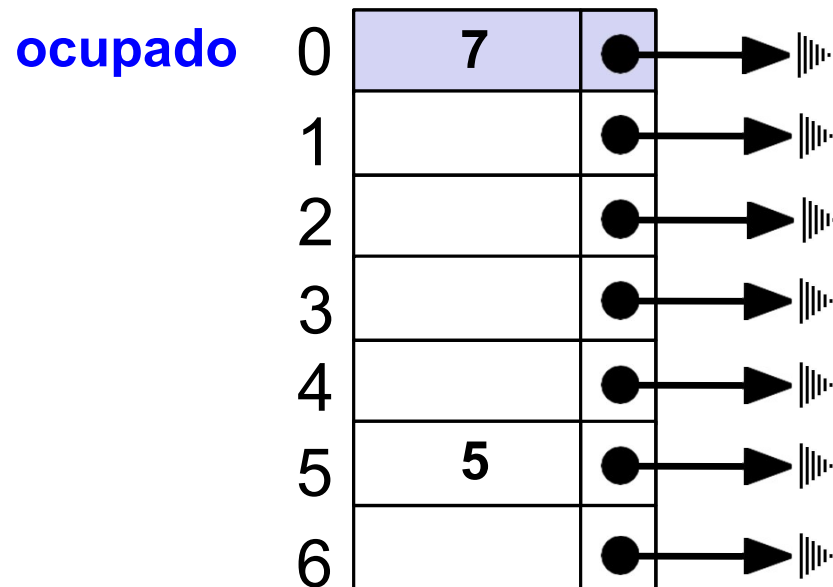


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

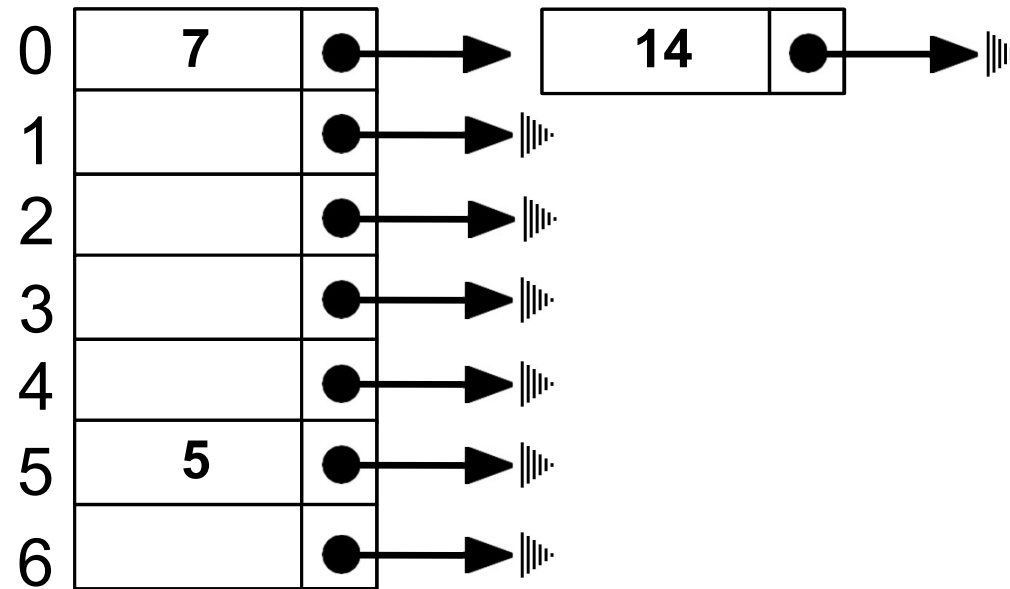


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

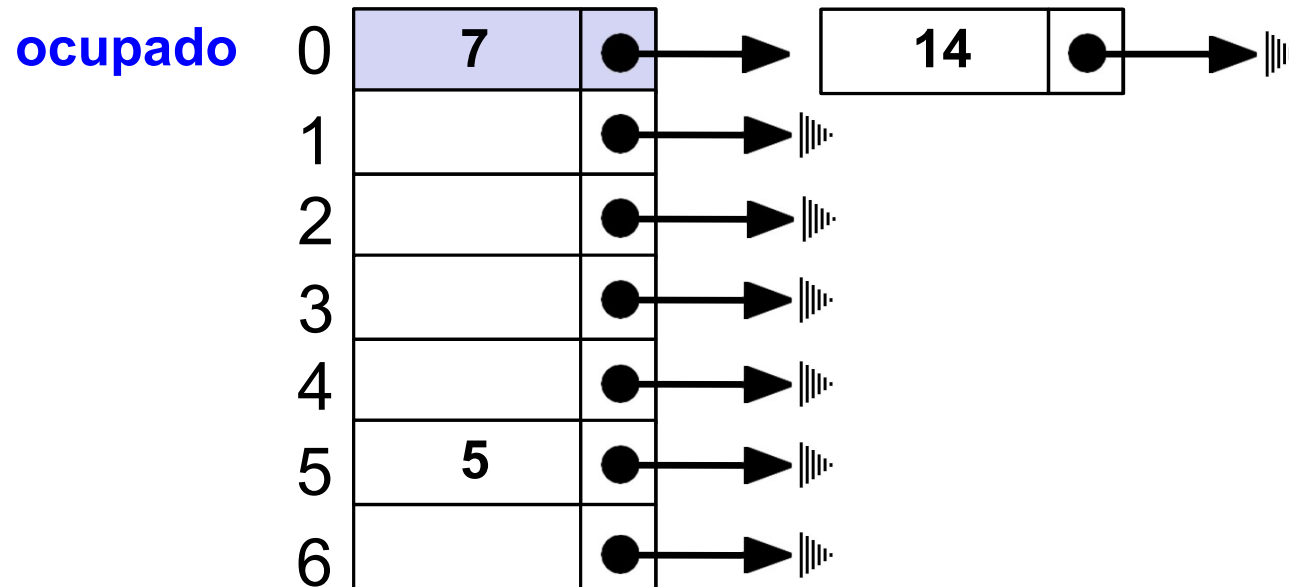


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

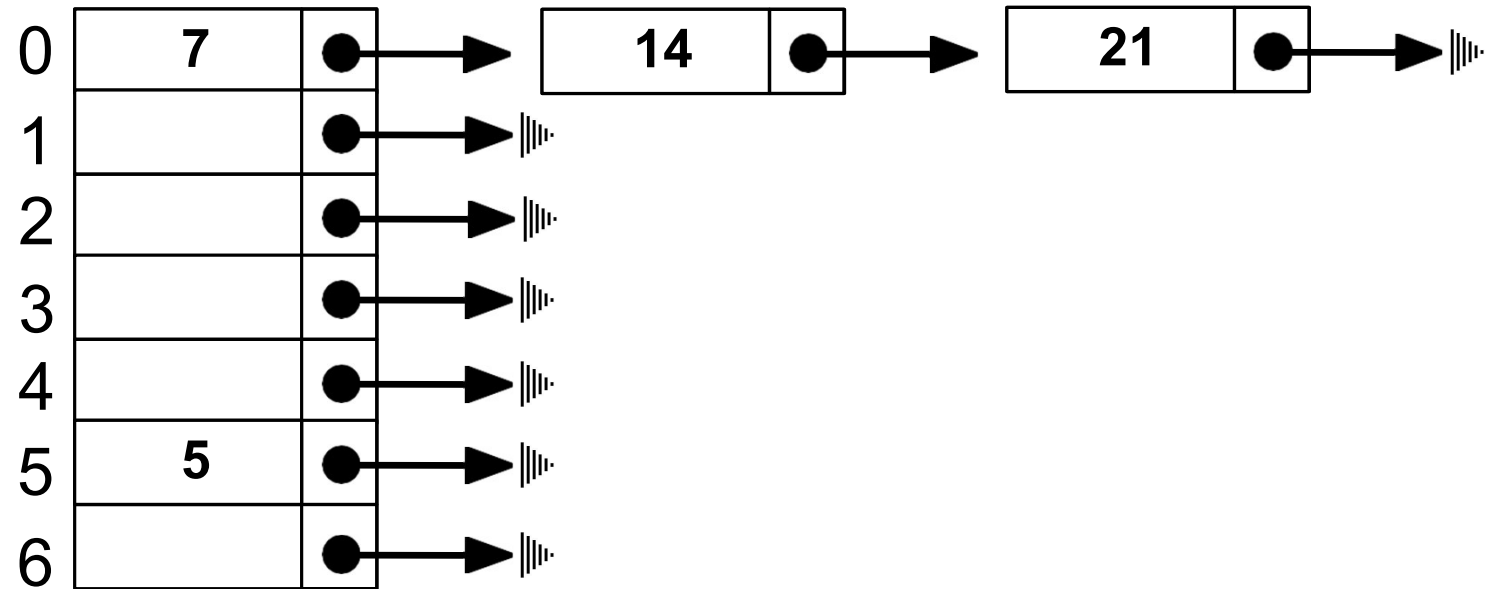


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

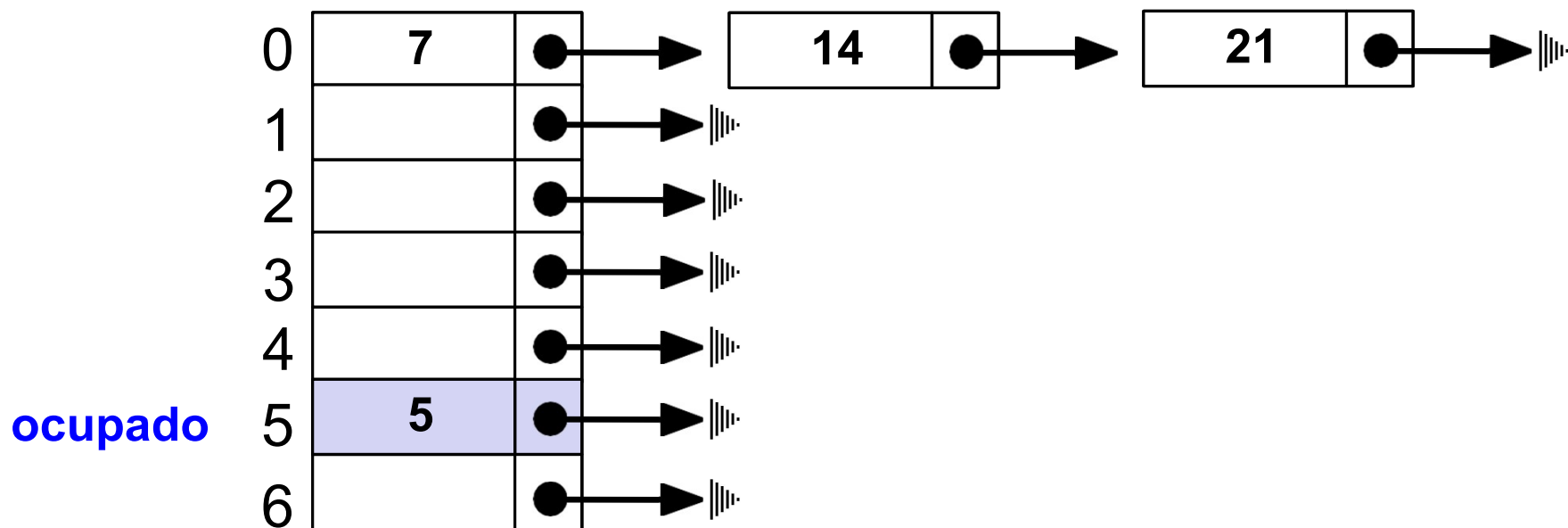


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

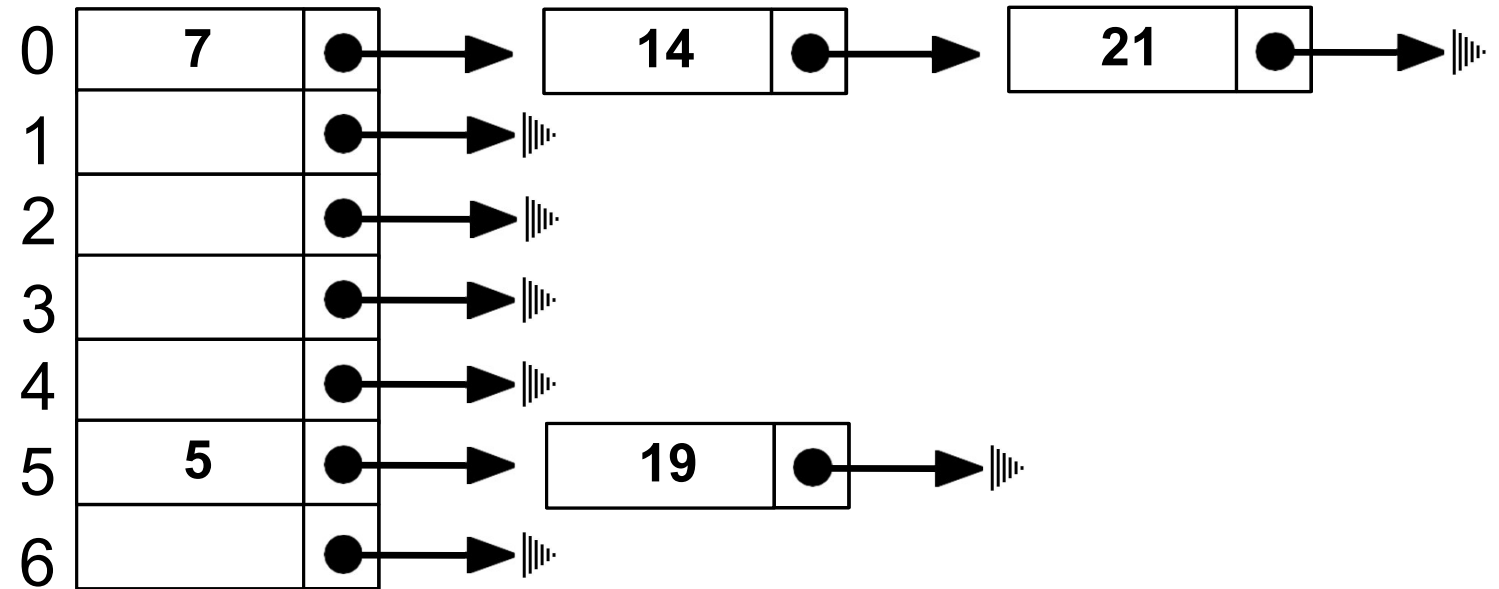


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7



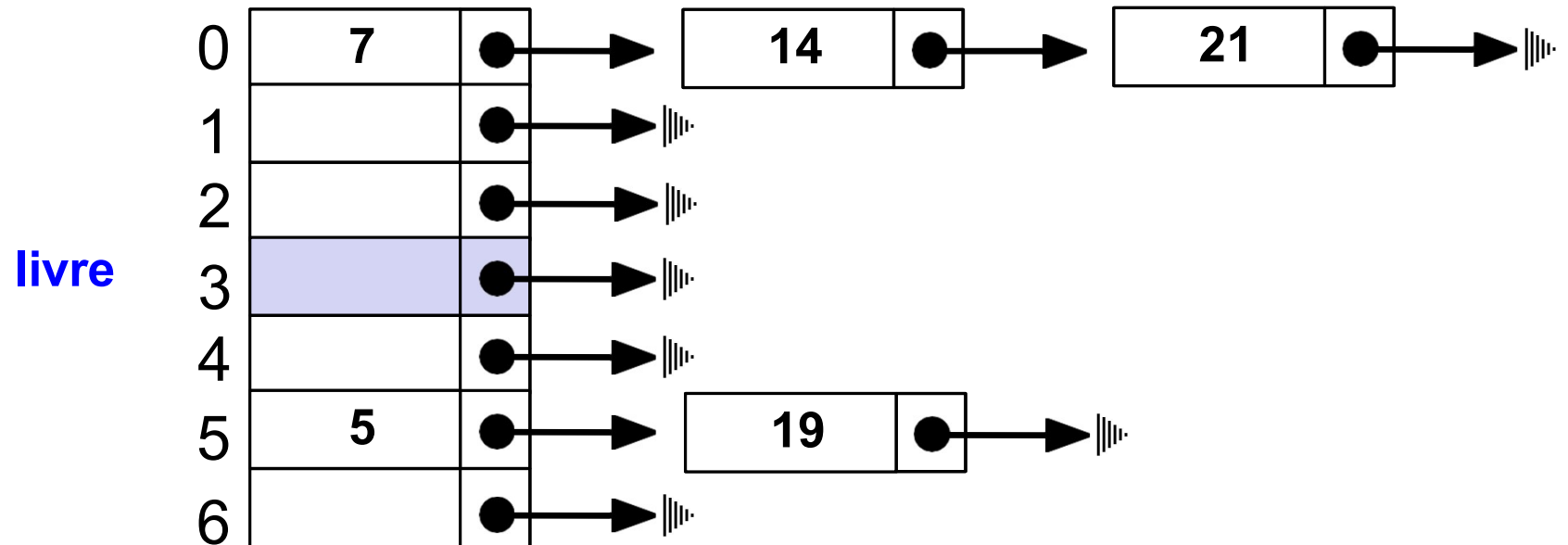
Vamos inserir os números:

5, 7, 14, 21, 19 e 3



# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

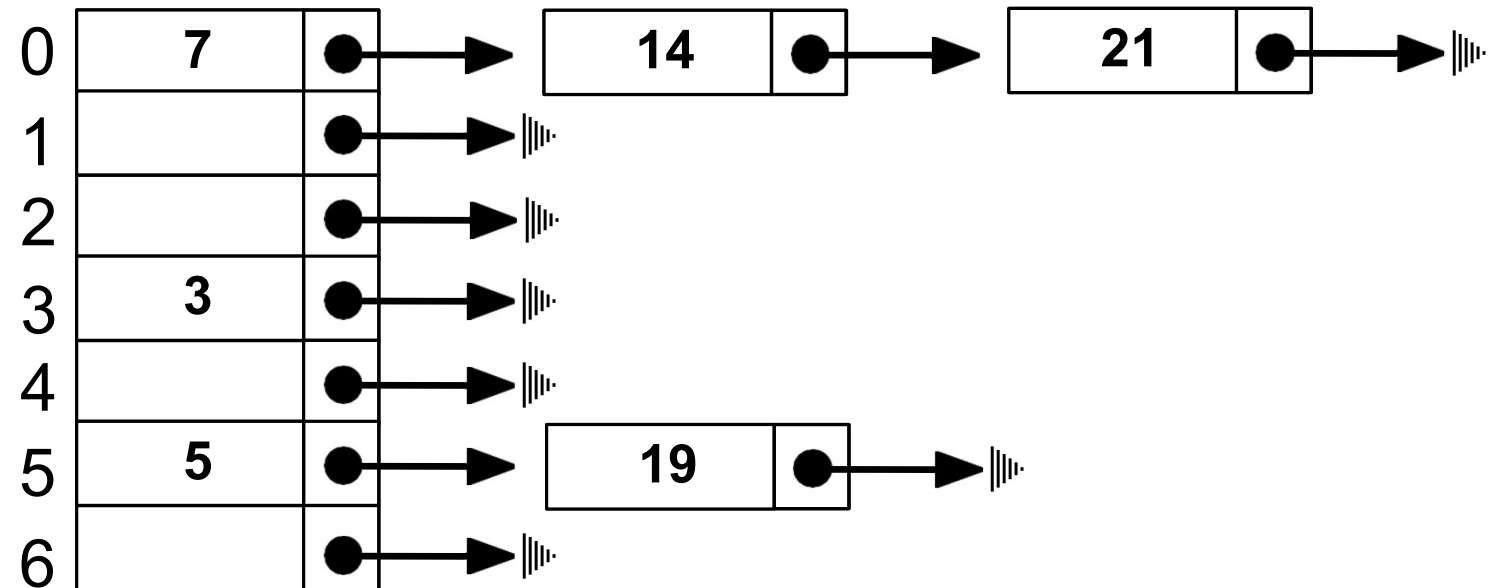


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

# Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7



Vamos inserir os números:

5, 7, 14, 21, 19 e 3

## Exercício Resolvido (2)

- Como podemos implementar este TAD?

## Exercício Resolvido (2): Construtor da Classe

- Como podemos implementar este TAD?

```
class Hash {  
    public Lista[] tabela;  
    public Hash (int m) {  
        tabela = new Lista [m];  
        for (int i = 0; i < m; i++){    tabela[i] = new Lista();  }  
    }  
  
    // Qual é a diferença dessa implementação  
    // para nosso exemplo?  
}
```

# Exercício Resolvido (2): Método Inserir

```
void inserir(int x) throws Exception {  
    if (pesquisar(x) == true){  
        throw new Exception("Erro ao inserir!");  
    } else {  
        tabela[hash(x)].inserir(x);  
    }  
}
```

# Exercício Resolvido (2): Método Pesquisar

```
boolean pesquisar(int x){  
    return tabela[hash(x)].pesquisar(x);  
}
```

# Exercício Resolvido (2): Método Remove

```
void remover(int x){  
    tabela[hash(x)].remover(x);  
}
```

# Análise de Complexidade

- Supondo que todos os elementos têm a mesma probabilidade de endereçamento, o comprimento esperado de cada lista será  $n/m$ , onde  $n$  é número de registros e  $m$  o tamanho da tabela
- As operações três operações custam em média  $\Theta(1 + n/m)$ , onde 1 é para encontrarmos a entrada na tabela e  $n/m$ , para percorrermos a lista
- Para valores de  $m$  próximos de  $n$ , o tempo se torna constante, ou seja independente de  $n$



# Gerenciamento de Colisões

- *Hash* direta com área de reserva (*overflow*)
- *Hash* direta com rehash
- *Hash* indireta com lista flexível simples

# Mapas (maps)

- Muitas vezes queremos buscar um objeto a partir de alguma informação sobre ele.
- Ex:
  - Buscar um carro a partir do número da placa
  - Buscar um usuário a partir do e-mail
  - Buscar uma pessoa a partir do CPF
  - Buscar um livro a partir do ISBN

# Mapas (maps)

- Um mapa é composto por um conjunto de associações entre um objeto chave e um objeto valor.
- É equivalente ao conceito de dicionário.
- É representado pela interface Map e as principais implementações são HashMap, TreeMap e Hashtable.

# Mapas (maps)

- Um mapa é composto por um conjunto de associações entre um objeto chave e um objeto valor.
- É equivalente ao conceito de dicionário.
- É representado pela interface Map e as principais implementações são HashMap, TreeMap e Hashtable.

# Mapas (maps)

```
...
import java.util.HashMap;
import java.util.Map;
...

...
Map<String, String> alunos = new HashMap<String,String>();

//Adicionando elementos no mapa
alunos.put("435", "Zezinho");
alunos.put("436", "Huguinho");
alunos.put("437", "Luizinho");

//Remove um elemento do mapa
alunos.remove("437");

//Verifica se Huguinho está no mapa
boolean huguinhoAdicionado = alunos.containsKey("436");
System.out.println("Huguinho está na lista? " + huguinhoAdicionado);

//Verifica se Zezinho está no mapa
boolean zezinhoAdicionado = alunos.containsValue("Zezinho");
System.out.println("Zezinho está na lista? " + zezinhoAdicionado);

//Apaga todos os itens do mapa
alunos.clear();
```

# Mapas (maps)

- Podemos utilizar o for para extrair as chaves e buscar os valores em um mapa.

```
Map<String, String> alunos = new HashMap<String,  
String>();  
  
//Adicionando elementos no mapa  
alunos.put("435", "Zezinho");  
alunos.put("436", "Huguinho");  
alunos.put("437", "Luizinho");  
  
//Imprime todos os valores no mapa  
for(String matricula : alunos.keySet()){  
    String nome = alunos.get(matricula);  
    System.out.println("Matricula: " + matricula + " - "  
+ nome);  
}
```