Unidade VII:
Tabelas *Hash*

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Introdução

- 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

Introdução

- 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 de transformação 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

Exemplo

- Como será nossa função de transformação?
  - 5/1 ⇒ Posição 4 (0 + 5 - 1)
  - 13/3 ⇒ Posição 72 (31+29+13-1)
  - 26/5 ⇒ Posição 146 (31+29+31+30+26-1)
  - 13/9 ⇒ Posição 256 (31+29+31+30+31+30+31+31+13-1)

Exemplo

- Como será nossa função de transformação?

```
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

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

- Qual é o custo de encontrar uma pessoa?

- Algum problema em nossa técnica? Se sim, qual é a chance desse problema acontecer?

Brincadeira

- Quem deseja apostar R$ 1000,00 que nesta turma (60 alunos) existem pelo menos duas pessoas que fazem aniversário no mesmo dia?

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 de transformação 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 de Transformação

- 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 de Transformação

- 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

- Assim, fazendo m = 13, temos:

h(oscar) = 536 % 13 = 3

h(bosco) = 534 % 13 = 1

h(elisa) = 526 % 13 = 6

h(maya) = 424 % 13 = 8

h(lucio) = 540 % 13 = 7

h(paulo) = 545 % 13 = 12

h(sofia) = 530 % 13 = 10

h(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(maya) = 8)
- A ana está (h(ana) = 5)
- O sergio está (h(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 de transformação 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

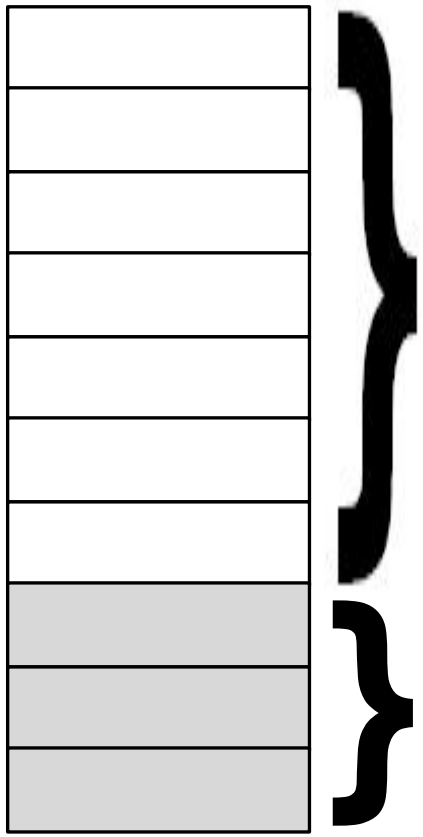
- *Hash* direta com área de reserva (*overflow*)
- *Hash* direta com rehash
- *Hash* indireta com lista flexível simples
- Estrutura híbrida

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 Área de Reserva (*overflow*)

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



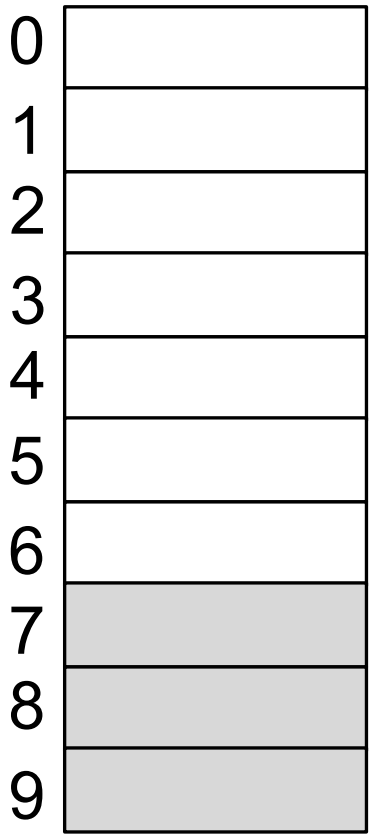
Tamanho da tabela

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



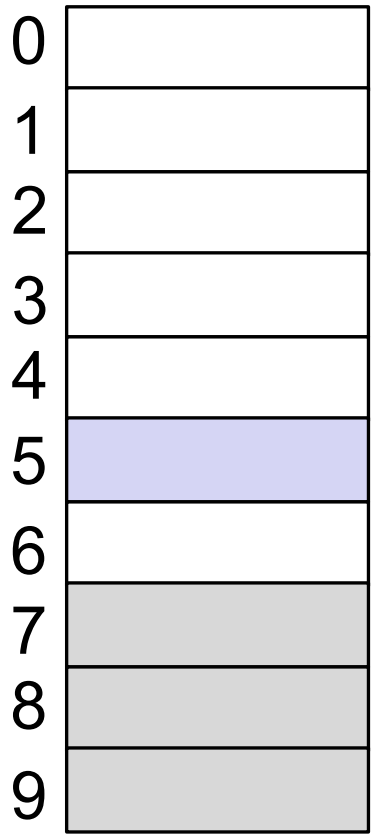

```
int hash (int x){
        return x % 7;
}
```

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



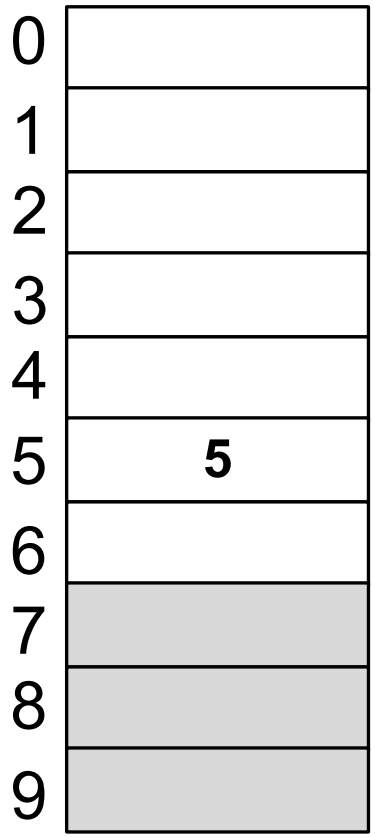

```
int hash (int x){
        return x % 7;
}
```

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




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

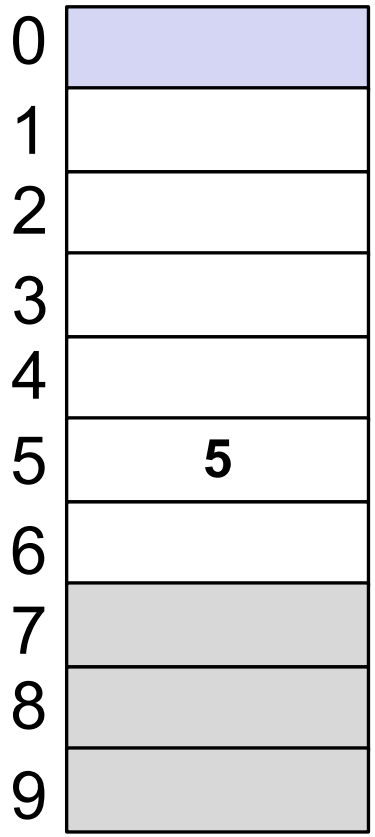
Vamos inserir os números:

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




```
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	

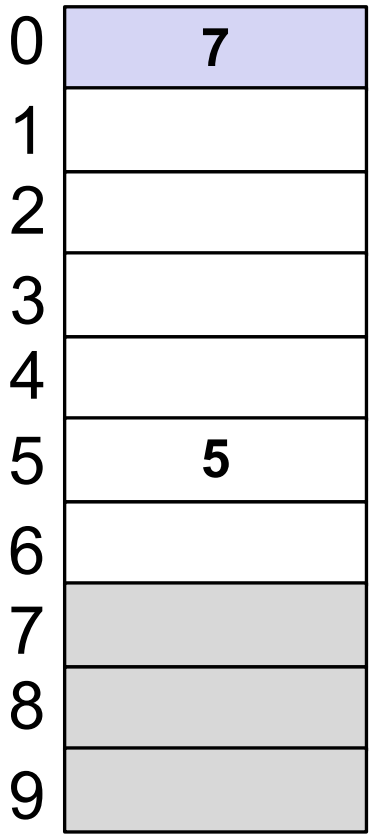
```
int hash (int x){
        return x % 7;
}
```

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



ocupado

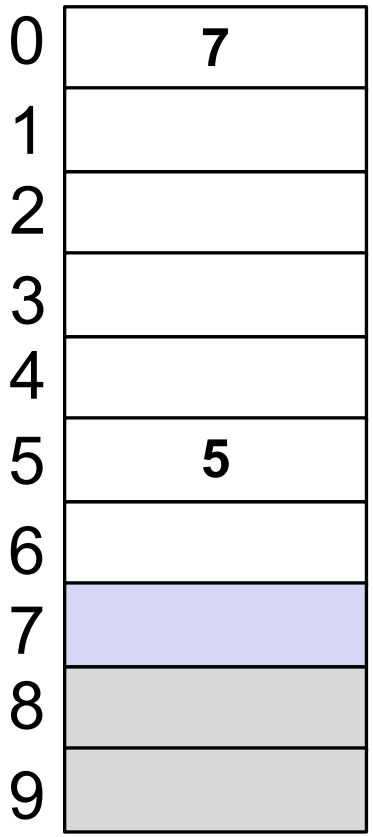
```
int hash (int x){
        return x % 7;
}
```

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




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

ocupado

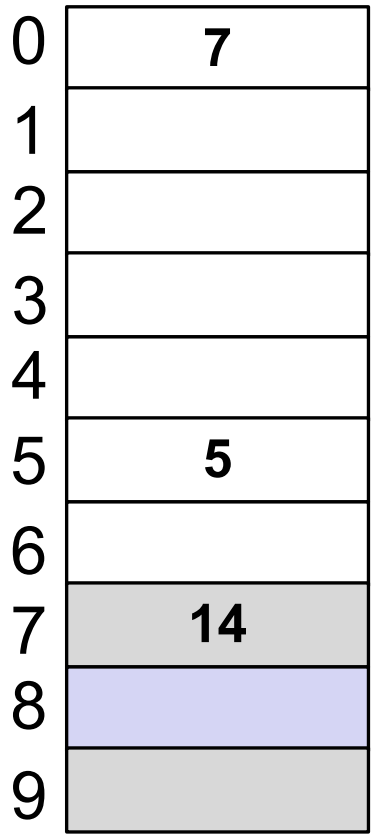
```
int hash (int x){
        return x % 7;
}
```

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




```
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	

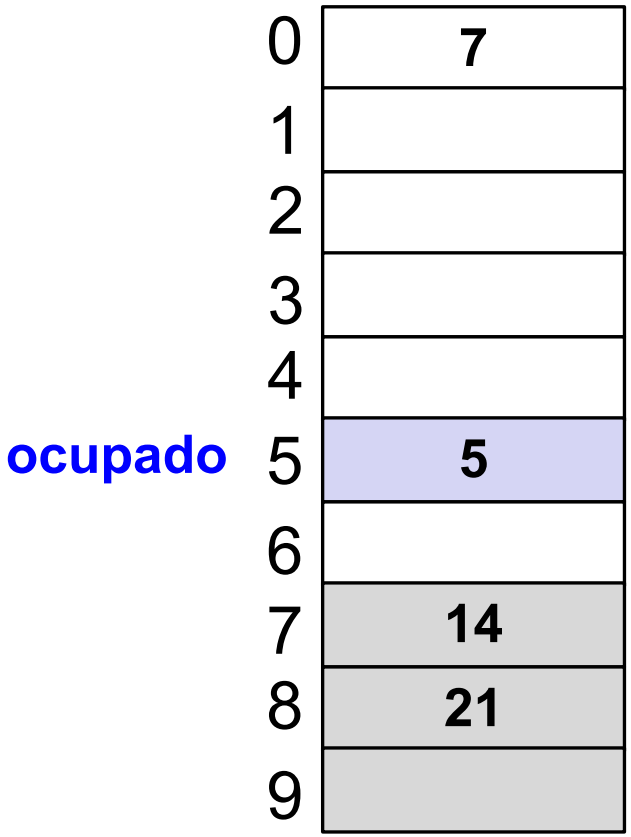
```
int hash (int x){
        return x % 7;
}
```

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



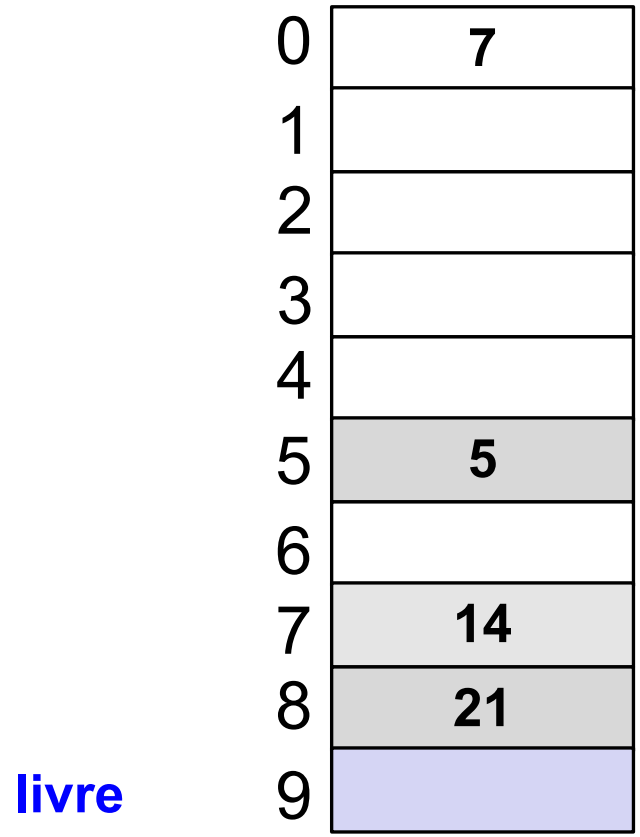

```
int hash (int x){
        return x % 7;
}
```

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




```
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)

- Implemente os métodos inserir, pesquisar e remover da *hash* direta com área de reserva

Exercício Resolvido (1): Método Inserir

- Im *sh*

dir

```
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

- I… …sh
d…

```
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;
}
```

Exercício Resolvido (1): Método Remover

- Implemente os métodos inserir, pesquisar e remover da *hash* direta com área de reserva

Bom trabalho!!!

Análise de Complexidade

- Como mostrado por Knuth (1973), o custo de uma pesquisa com sucesso é $C(n) = \frac{1}{2}(1 + \frac{1}{1-\alpha})$, onde $\alpha = n/m$ é o fator de carga da tabela

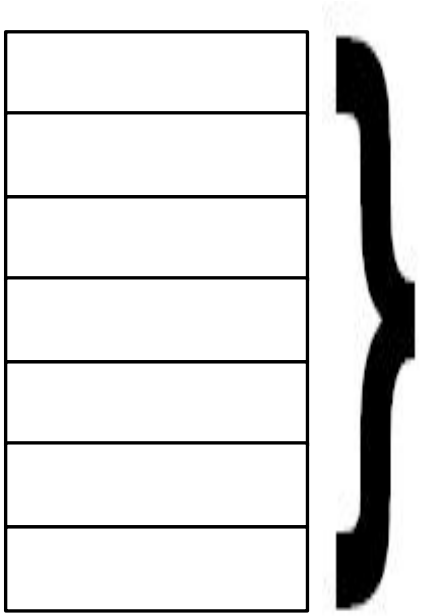
- A tabela hash direta sofre de um mal chamado *clustering* (Knuth, 1973, pp.520-521) no qual o tempo de pesquisa médio aumenta quando a tabela começa a ficar cheia

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

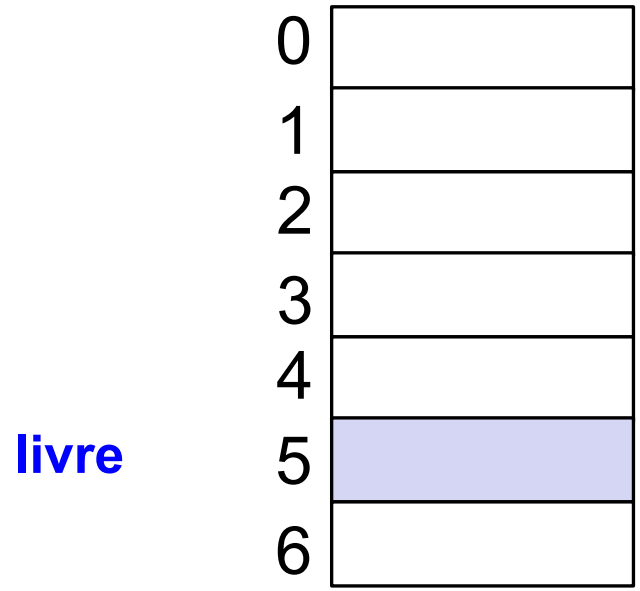
- Suponha uma tabela de tamanho 7

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

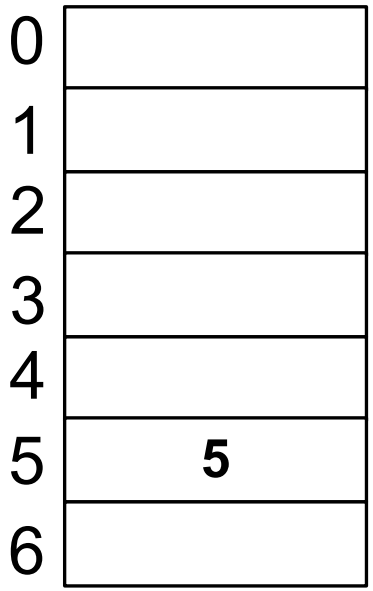


Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

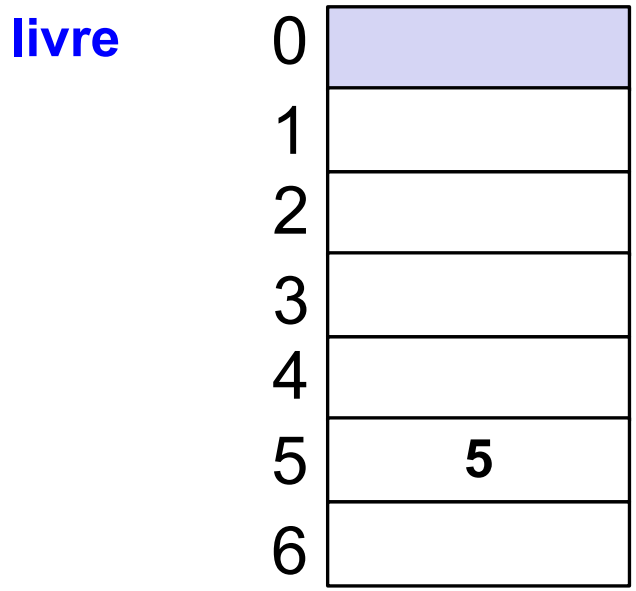


Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7



Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

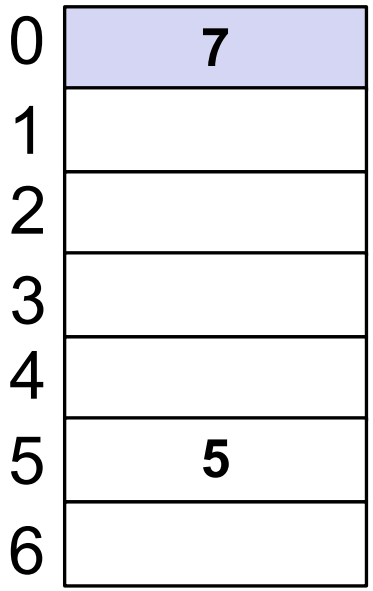
0	**7**
1	
2	
3	
4	
5	**5**
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7



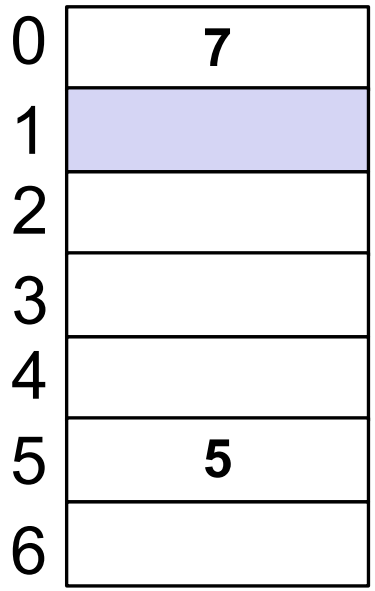
Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

livre




```
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

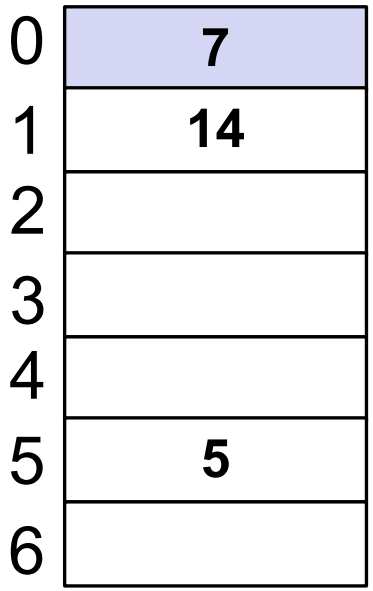
0	**7**
1	**14**
2	
3	
4	
5	**5**
6	

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7



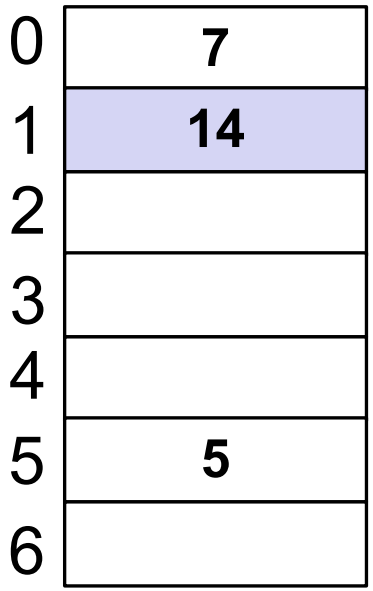
ocupado

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7



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	

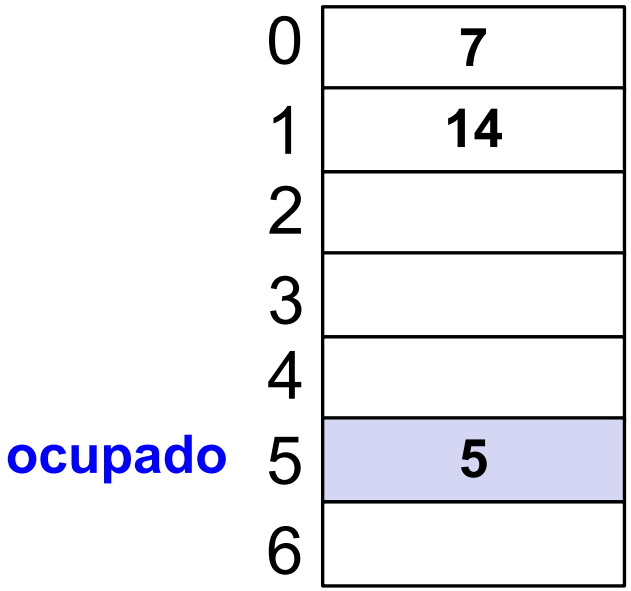
Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Não é possível inserir o 21.

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7

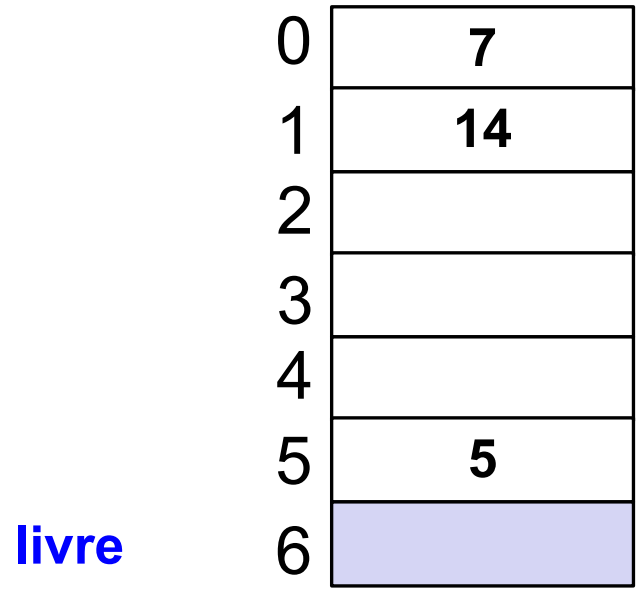


Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7




```
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

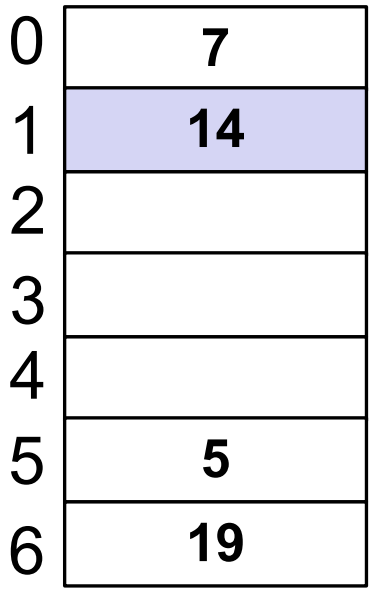
0	**7**
1	**14**
2	
3	
4	
5	**5**
6	**19**

Vamos inserir os números:

5, 7, 14, 21, 19 e 1

Hash Direta com Rehash

- Suponha uma tabela de tamanho 7



ocupado

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

Exercício (3)

- Implemente a classe Hash com Rehash (atributos e métodos construtor, inserir, pesquisar e remover)

Gerenciamento de Colisões

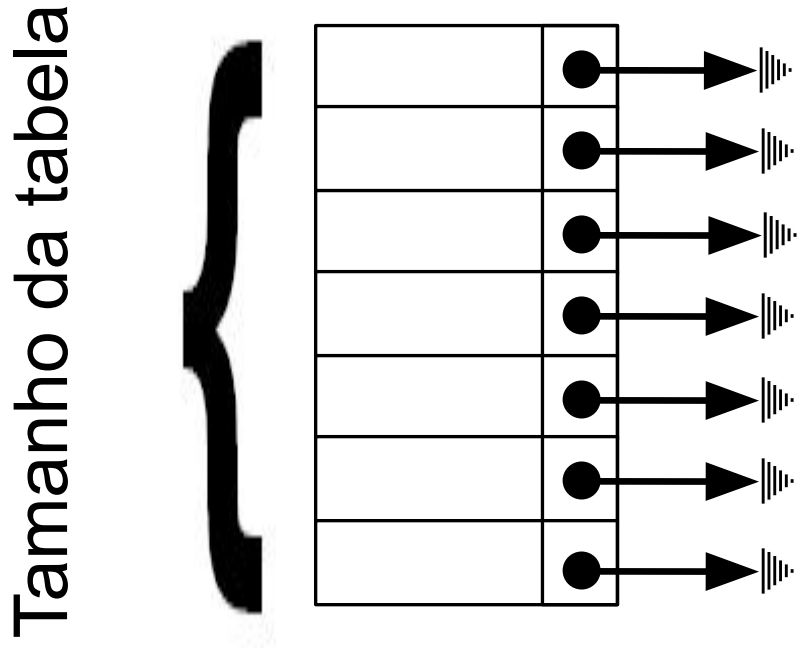
- *Hash* direta com área de reserva (*overflow*)
- *Hash* direta com rehash
- ***Hash* indireta com lista flexível simples** 
- Estrutura híbrida

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

Hash Indireta com Lista Flexível Simples

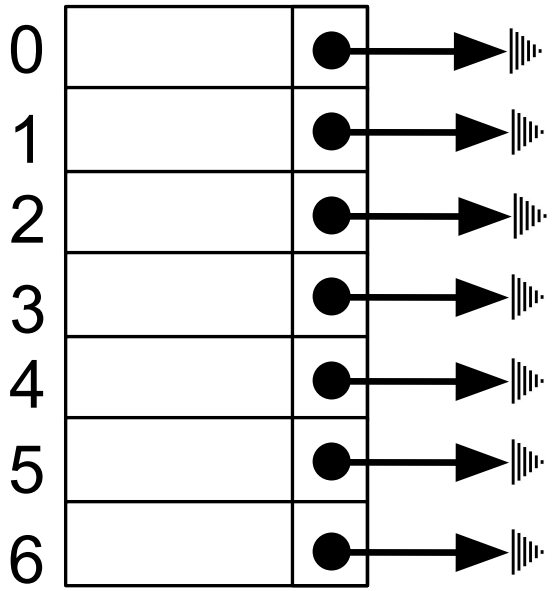
- Suponha uma tabela de tamanho 7




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

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

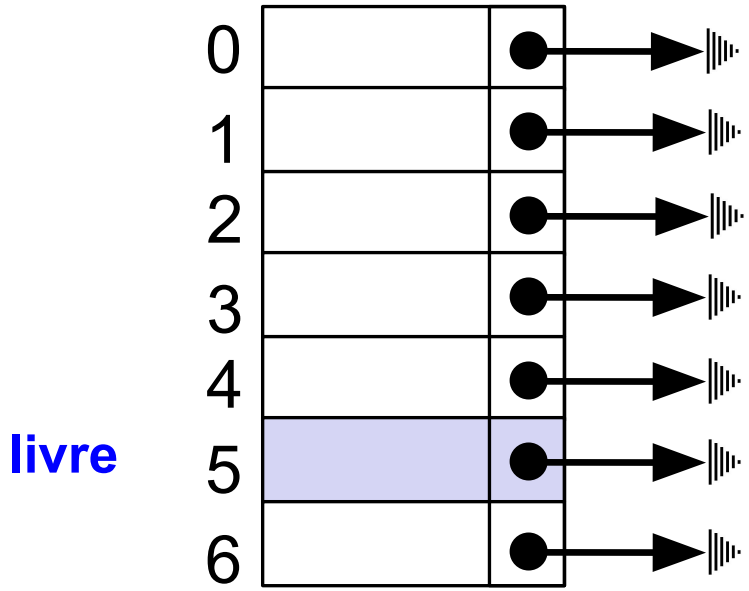


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

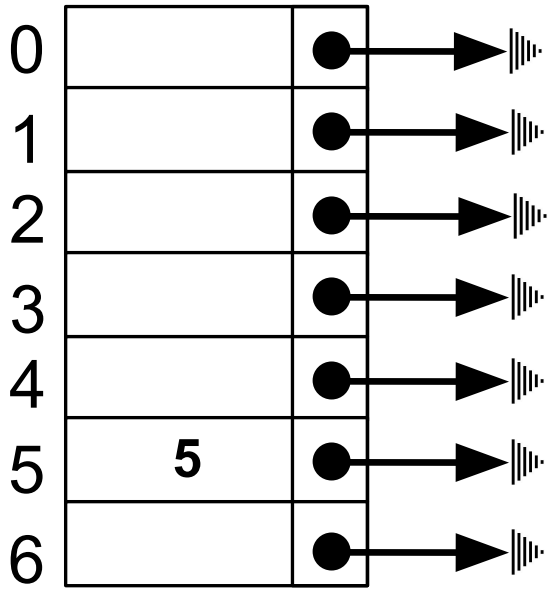


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

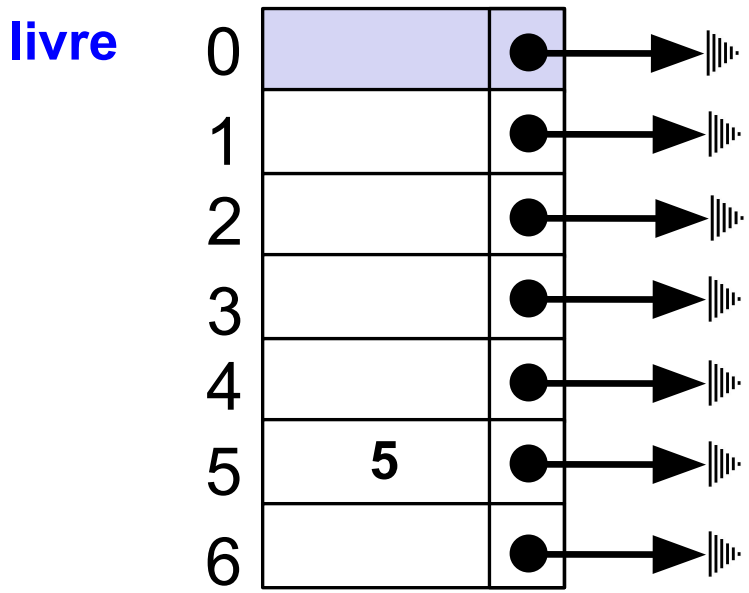


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

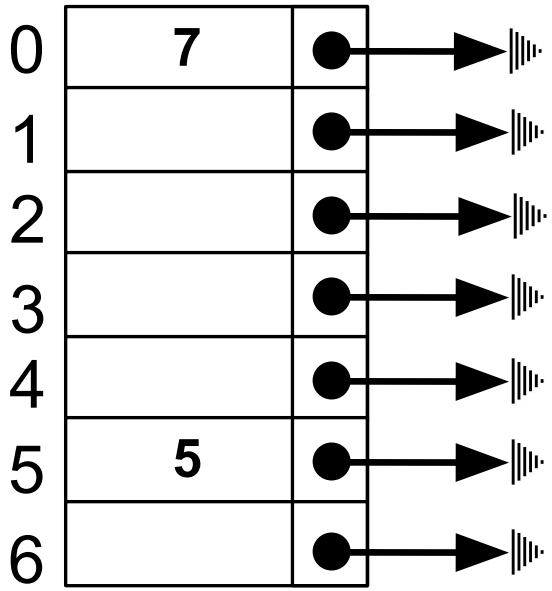


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

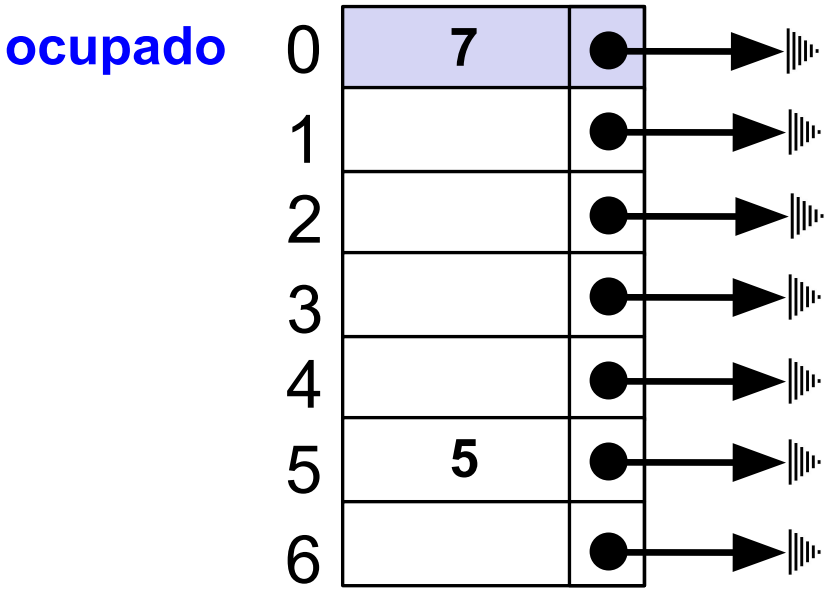


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

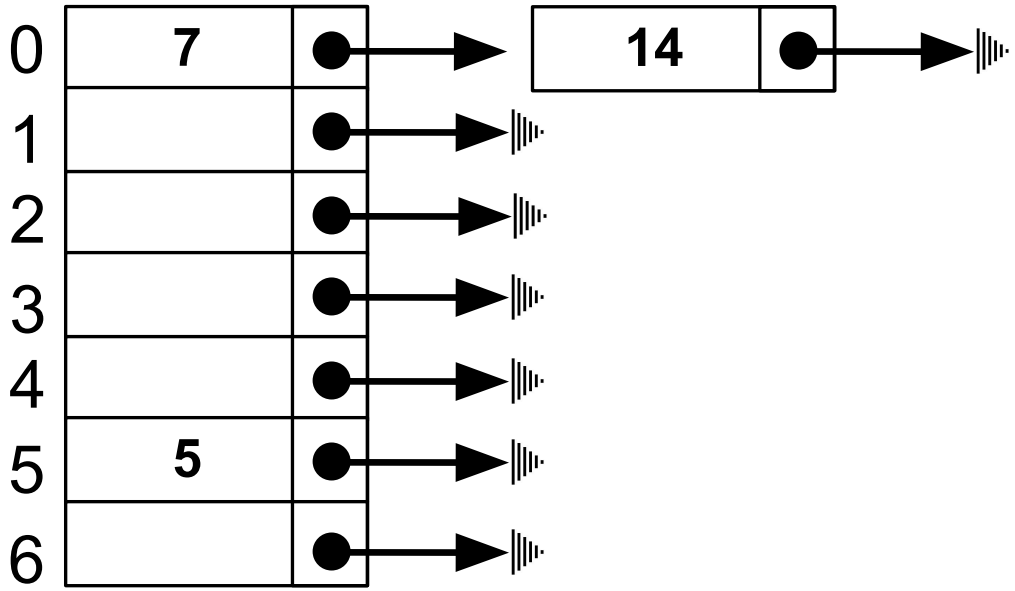


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

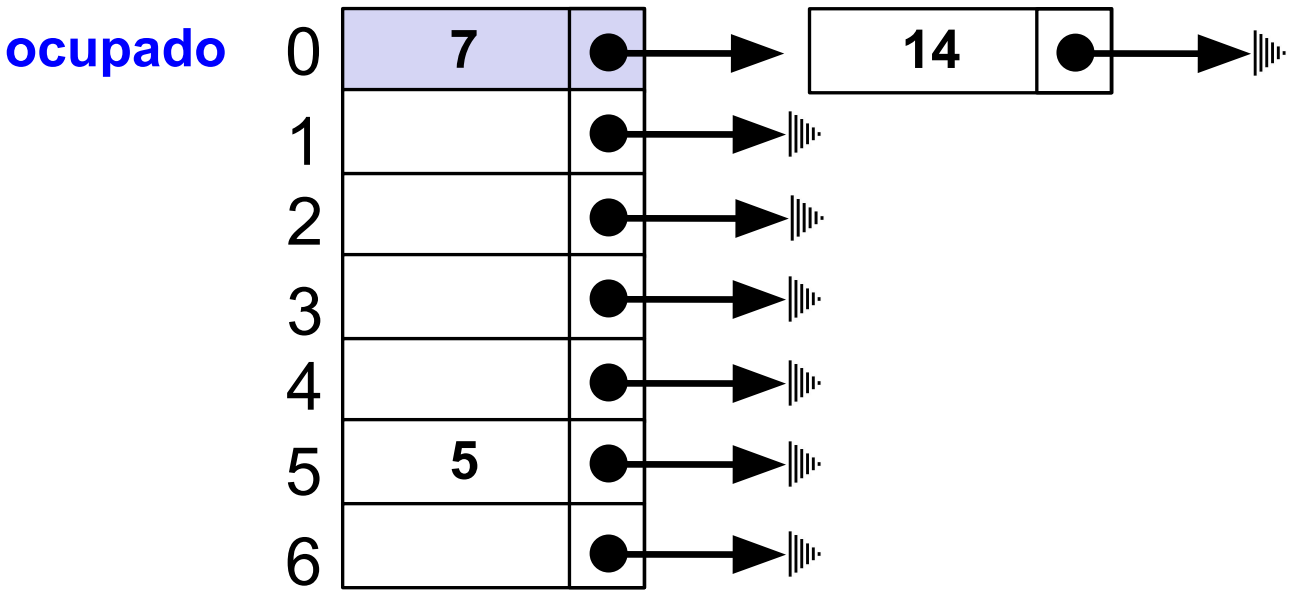


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

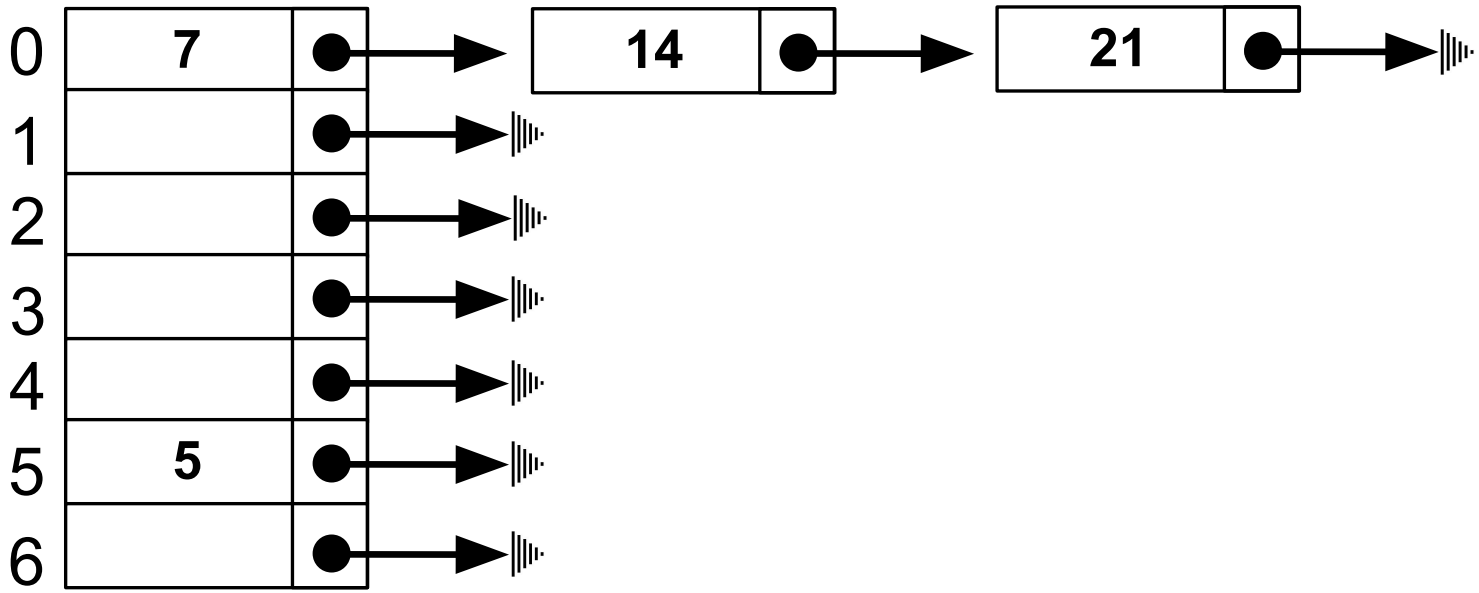


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

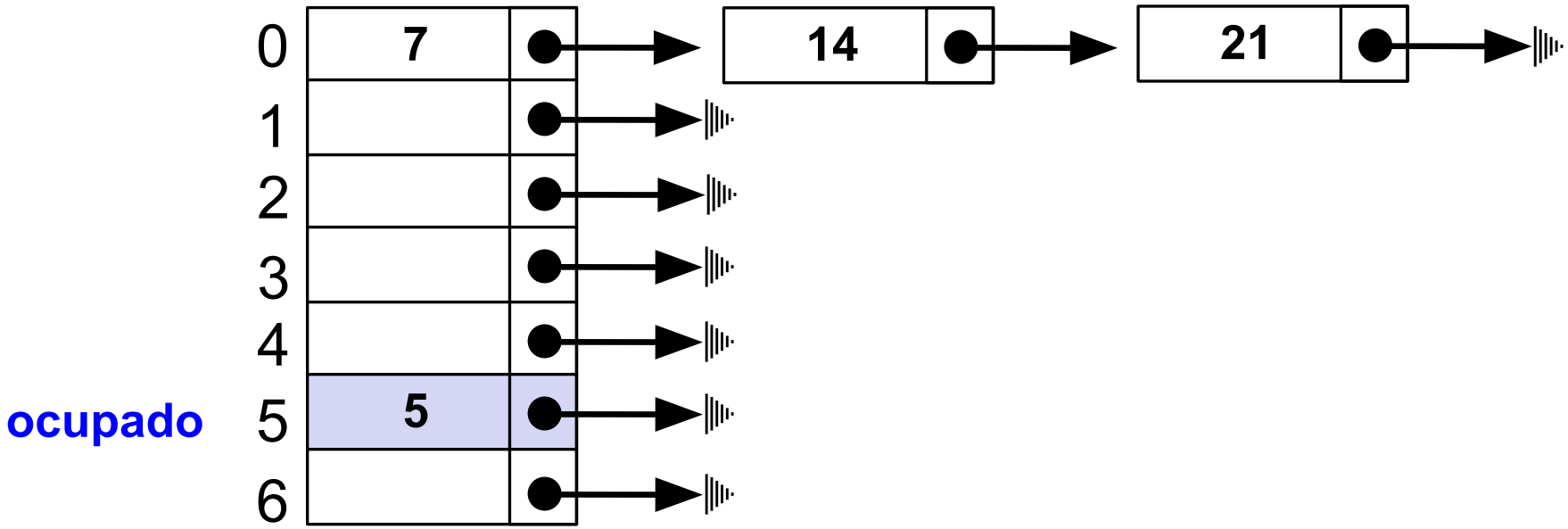


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

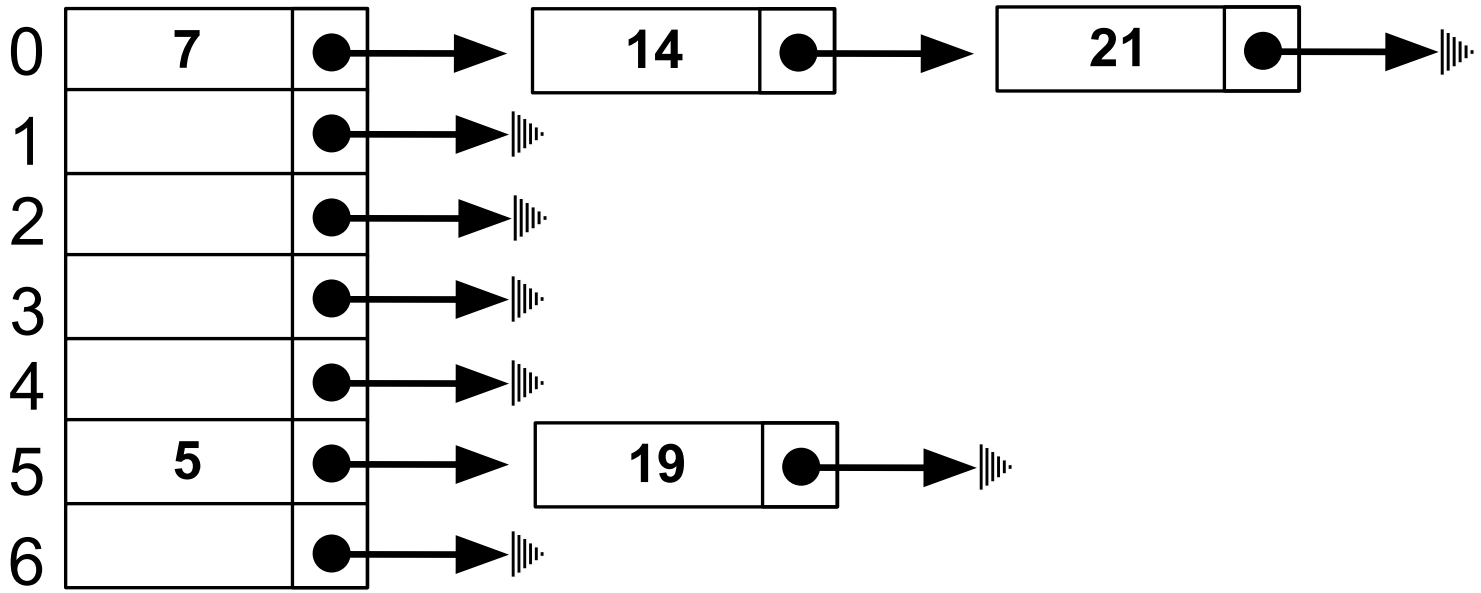


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

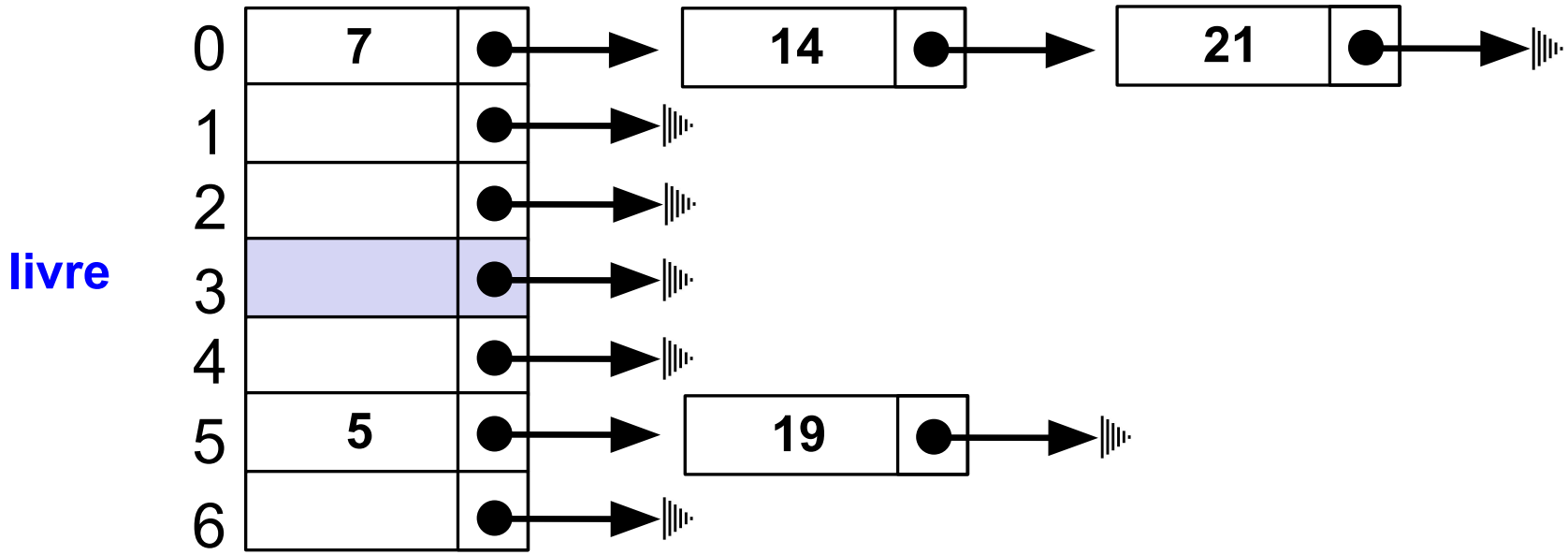


Vamos inserir os números:

5, 7, 14, 21, 19 e 3

Hash Indireta com Lista Flexível Simples

- Suponha uma tabela de tamanho 7

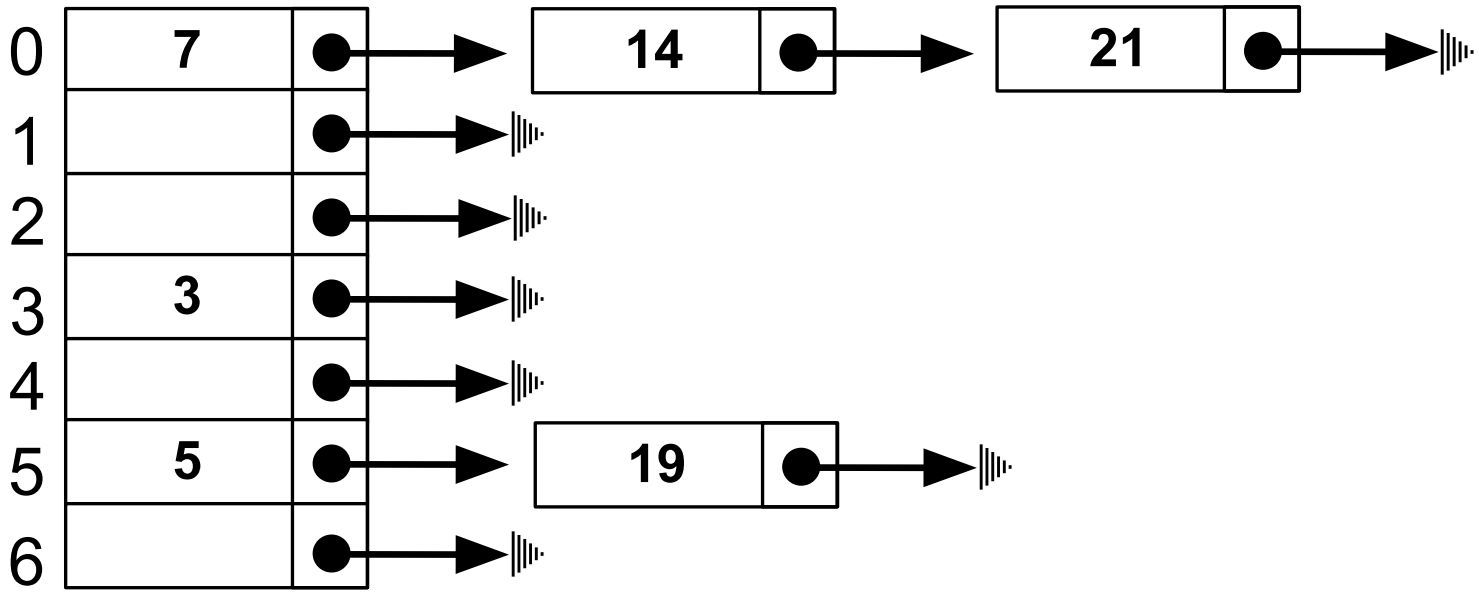


Vamos inserir os números:

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 Remover

```
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 é 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

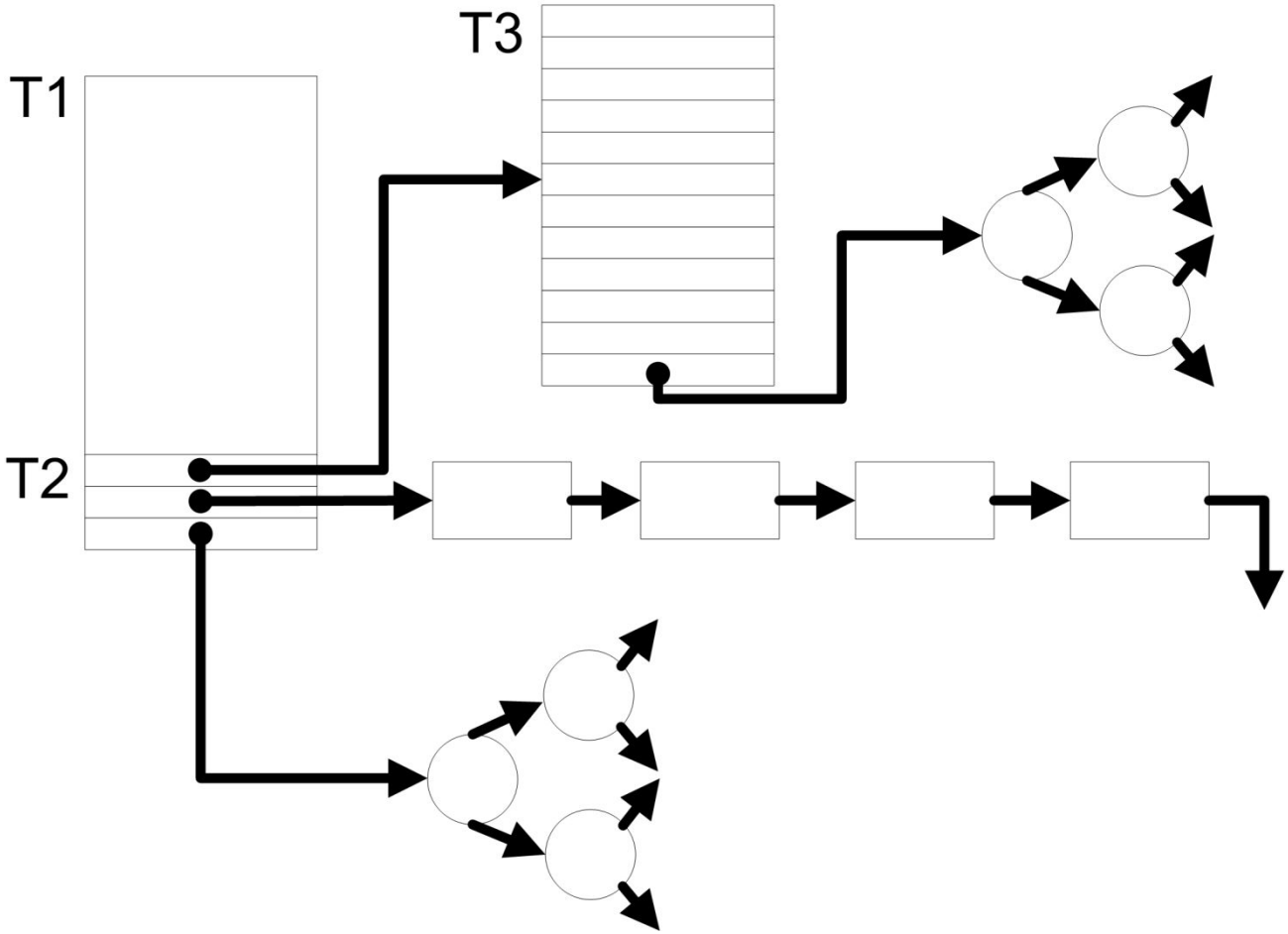
- *Hash* direta com área de reserva (*overflow*)
- *Hash* direta com rehash
- *Hash* indireta com lista flexível simples
- **Estrutura híbrida** 

Exercício (4)

- Faça os métodos de inserir, pesquisar, mostrar e remover na estrutura abaixo. Ela é composta pela tabela *hash* T1 cuja área de reserva é T2 sendo que T2 é virtual.

T3 é uma *hash* com rehash e com uma árvore binária em sua área de reserva.

Considere que os métodos de *hash* e rehash estão implementados. Use as classes de Lista e Árvore apresentadas.



Exercício (5)

- Repita o exercício anterior, contudo, sem usar as classes Lista e Árvore.

Nesse caso, você deve usar as classes Nó e Célula.