



Algoritmo e Estrutura de Dados II

COM-112

Aula 18

Vanessa Souza



Introdução

- ▶ Até agora vimos basicamente dois tipos de estruturas de dados para o armazenamento flexível de dados: **Listas e Árvores**.
 - ▶ Cada um desses grupos possui muitas variantes.
- ▶ As Listas são simples de se implementar, mas, com um tempo médio de acesso $O(n/2)$ são impraticáveis para grandes conjuntos de dados.
- ▶ As Árvores são estruturas mais complexas, mas que possuem um tempo médio de acesso $T = \log_G n$, onde G é o grau do nó da árvore.





Introdução

- ▶ As árvores de pesquisa são utilizadas principalmente para consultar itens.
- ▶ A consulta é sempre pela busca de chaves ordenadas.
- ▶ Existe uma forma alternativa de se fazer uma busca, chamada **busca direta**, implementada pela Tabela Hash.

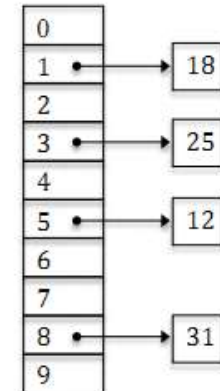




Endereçamento Direto

- Exemplo:** conjunto dinâmico $D = \{18, 25, 12, 31\}$, com chaves do conjunto $K = \{1, 3, 5, 8\}$, respectivamente, onde o conjunto possível de chaves é $C = \{0, 1, \dots, 9\}$.

Neste caso, a implementação de cada operação de dicionário é trivial e pode ser realizada em tempo $O(1)$:



```
int Recuperar(int T[], int k)
{
    return T[k];
}
```

```
void Excluir(int T[], int x)
{
    T[chave(x)] = NULL;
}
```

```
void Incluir(int T[], int x)
{
    T[chave(x)] = x;
}
```



Tabela Hash

- ▶ E se número possível de chaves (m) for muito grande? Neste caso, armazenar uma tabela de tamanho m pode ser impraticável.
- ▶ Quando o número de chaves realmente utilizadas é muito menor do que o número de chaves possíveis, uma tabela hash exige muito menos espaço ($O(|K|)$) do que uma tabela de endereçamento direto.
- ▶ Na tabela hash o elemento com chave k é armazenado na posição $h(k)$ (e não na posição k , como na tabela de endereçamento direto), em que h é denominada função hash.





Tabela Hash

- ▶ **Tabela hash** (de *hashing*, no inglês) é uma estrutura de dados especial, que associa chaves de pesquisa (*hash*) a valores.
- ▶ Seu objetivo é, a partir de uma chave simples, fazer uma busca rápida e obter o valor desejado.
 - ▶ A estruturação da informação em tabelas de hash visa principalmente permitir armazenar e procurar rapidamente grande quantidade de dados.
- ▶ Outros nomes:
 - ▶ Tabela de Dispersão
 - ▶ Tabela de Espalhamento
 - ▶ Tabela de escrutínio





Tabela Hash

- ▶ As tabelas de hash são constituídas por 2 conceitos fundamentais:
 - ▶ Tabela de Hash: estrutura que permite o acesso aos subconjuntos.
 - ▶ Função de Hash: função que realiza um mapeamento entre os valores de chaves e as entradas na tabela.

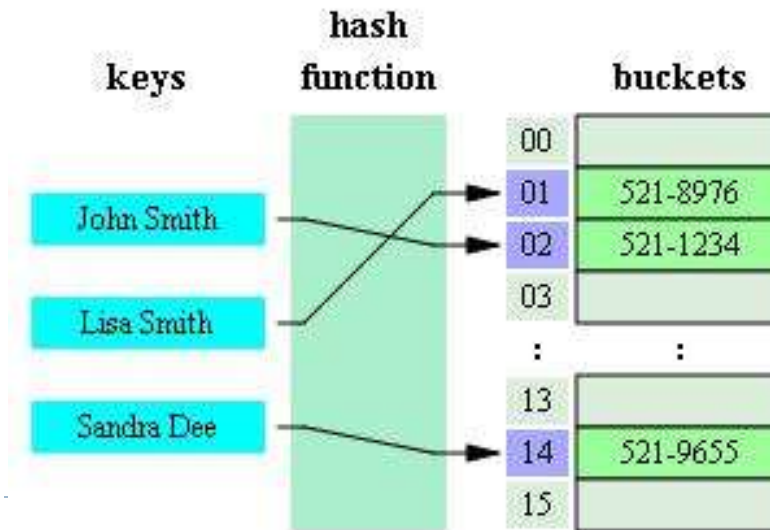




Tabela Hash

- ▶ Uma tabela hash é uma estrutura de dados eficiente para implementar dicionários, ou seja, conjuntos dinâmicos que admitem apenas as operações **Incluir**, **Excluir** e **Recuperar**.
- ▶ Possui uma série de limitações em relação às árvores:
 - ▶ Não permite recuperar/imprimir todos os elementos em ordem de chave, nem outras operações que exijam sequência dos dados.
 - ▶ Não permite operações do tipo recuperar o elemento com a maior ou a menor chave.





Tabela Hash

- ▶ A organização de arquivos *hashing* oferece:
 - ▶ Acesso direto ao endereço do bloco de disco que contém o registro desejado;
 - ▶ Aplica-se uma função sobre o valor da chave de procura do registro para encontrar o bloco de disco correto;





Funções de Hashing

- ▶ Uma função Hash ideal distribui as chaves armazenadas **uniformemente** na tabela Hash.
- ▶ Pior função:
 - ▶ Mapeia todos os valores de chave de busca para o mesmo local da tabela.





Funções de Hashing

```
int hash (int key, int tam)
```

```
{
```

```
    return (key % tam);
```

```
}
```

Neste caso, a chave *key* seria armazenada na posição relativa ao *int* retornado por essa função

► Objetivos da função de Hash:

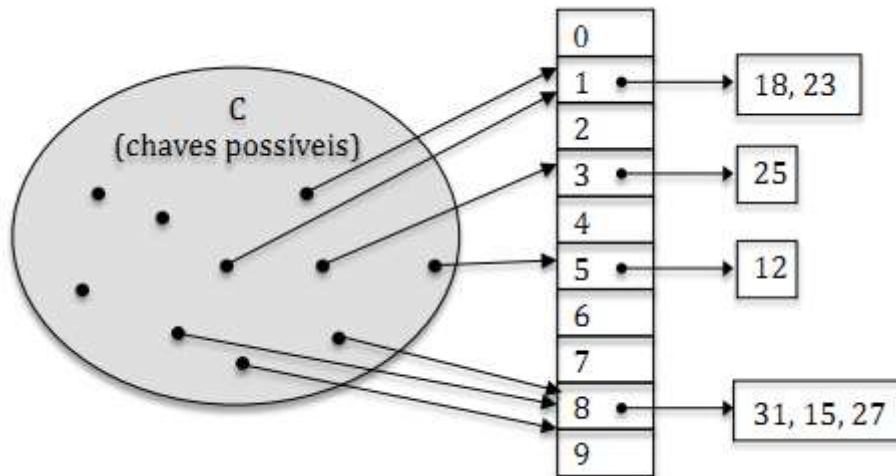
- Ser eficiente.
- Distribuir todos os elementos uniformemente por todas as posições da tabela.





Tabela Hash

- ▶ Quando duas chaves k_1 e k_2 , com $k_1 \neq k_2$, são tais que $h(k_1) = h(k_2)$, diz-se haver uma **colisão**.
- ▶ Colisões são inevitáveis em Tabelas Hash, já que o número de chaves possíveis é sempre maior que o número de entradas da tabela.



Tratamento de Colisões



Tratamento de Colisões

- ▶ Como as colisões são inevitáveis, é preciso arrumar formas de tratá-las.
- ▶ Resolução de colisões por encadeamento
- ▶ Endereçamento Aberto





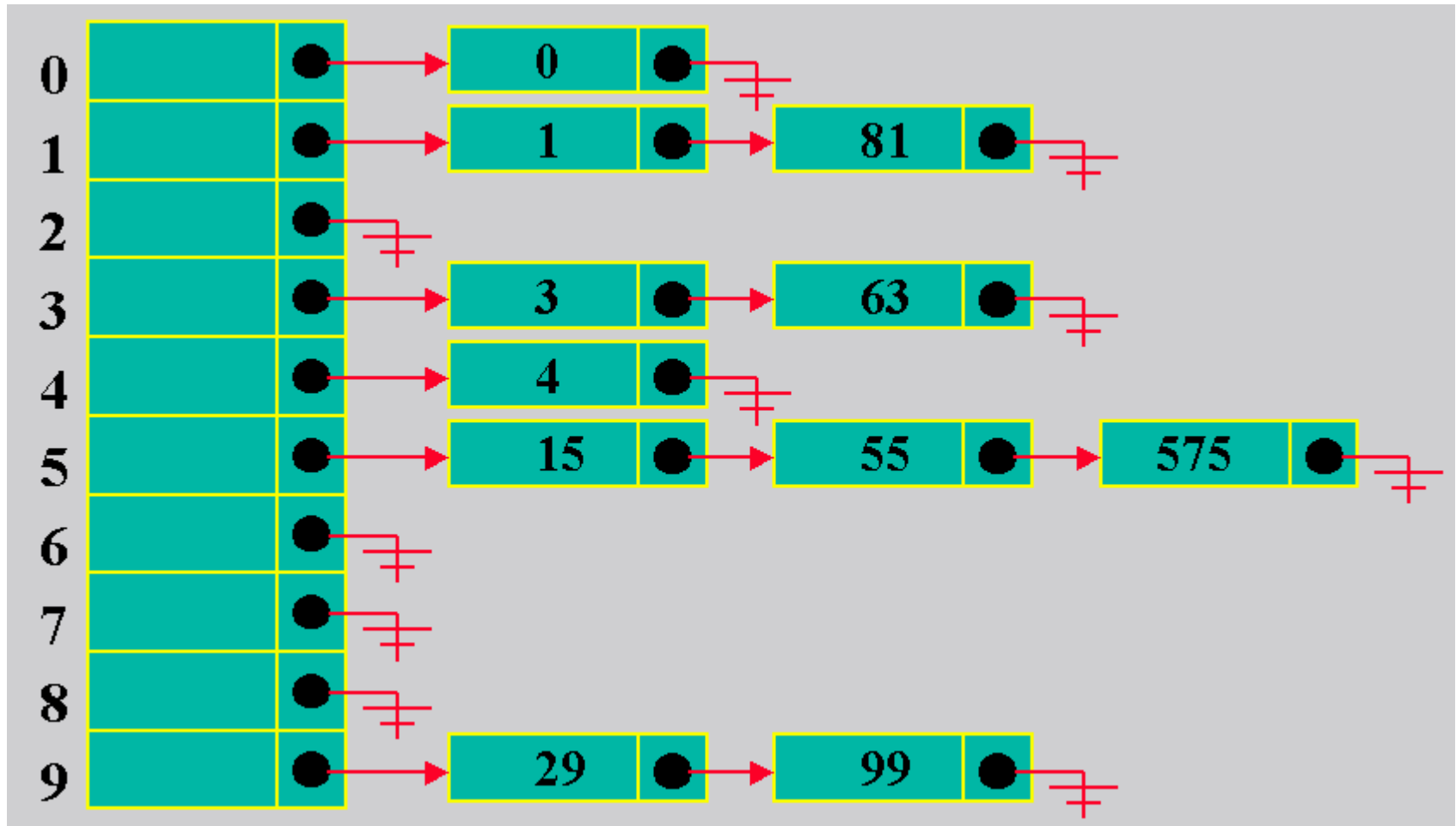
Encadeamento

- ▶ Também chamada de Resolução de colisões por meio de **listas encadeadas** ou *separate chaining* ou Hash Aberto.
- ▶ Para cada índice h da tabela há uma lista encadeada que armazena todos os objetos que a função de dispersão leva em h .
 - ▶ Essa solução é muito boa se cada uma das "listas de colisão" for curta.
 - ▶ Por quê?





Encadeamento





Encadeamento

- ▶ **Exemplo :**
- ▶ Dada uma tabela com 8 entradas utilizada para acomodar registros cujas chaves são valores pertencentes ao conjunto dos números naturais.
- ▶ Dada a função de dispersão **Resto da Divisão**

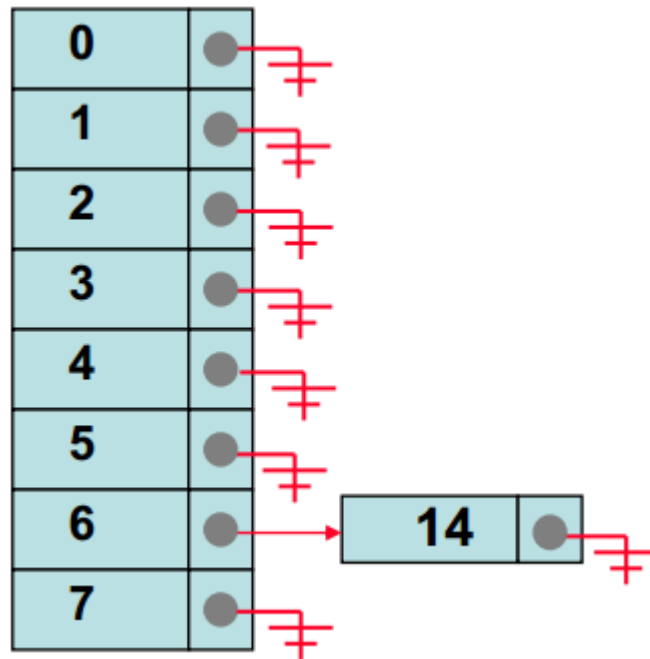
$$H(c) = c \bmod m$$

0	●	≠
1	●	≠
2	●	≠
3	●	≠
4	●	≠
5	●	≠
6	●	≠
7	●	≠



Encadeamento

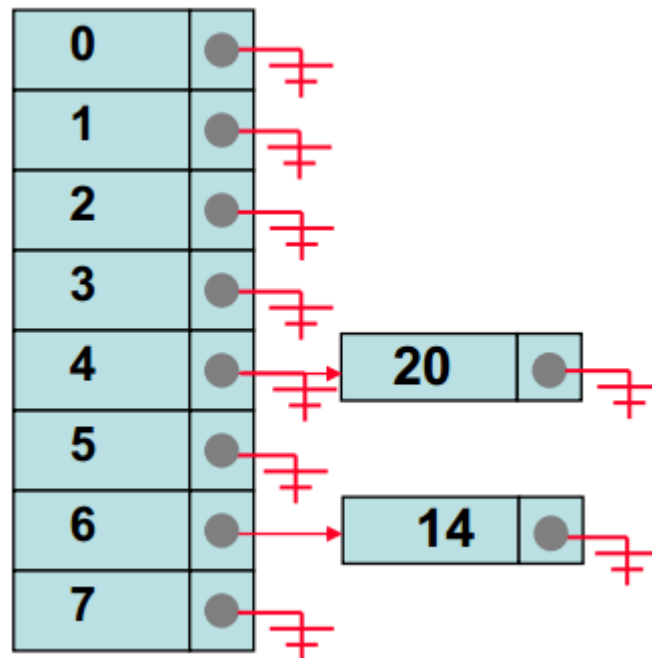
- ▶ **Exemplo** : Inserir 14
- ▶ $14 \bmod 8 = 6$





Encadeamento

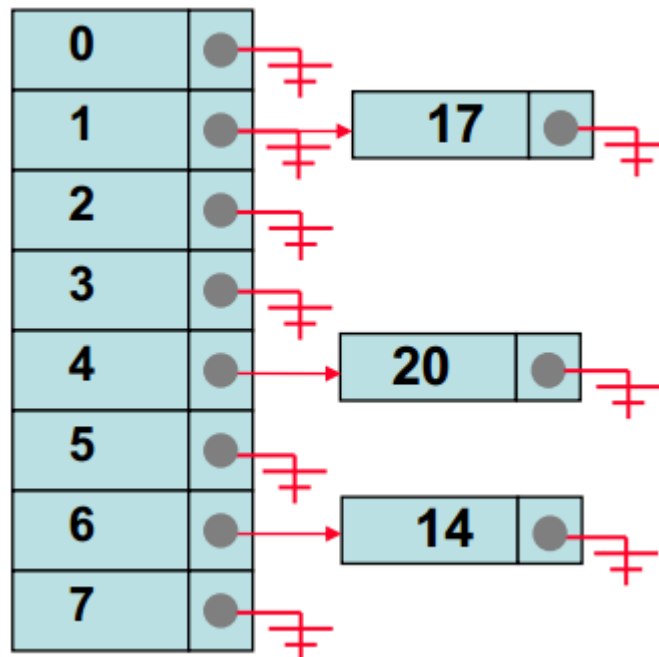
► Exemplo : Inserir 20





Encadeamento

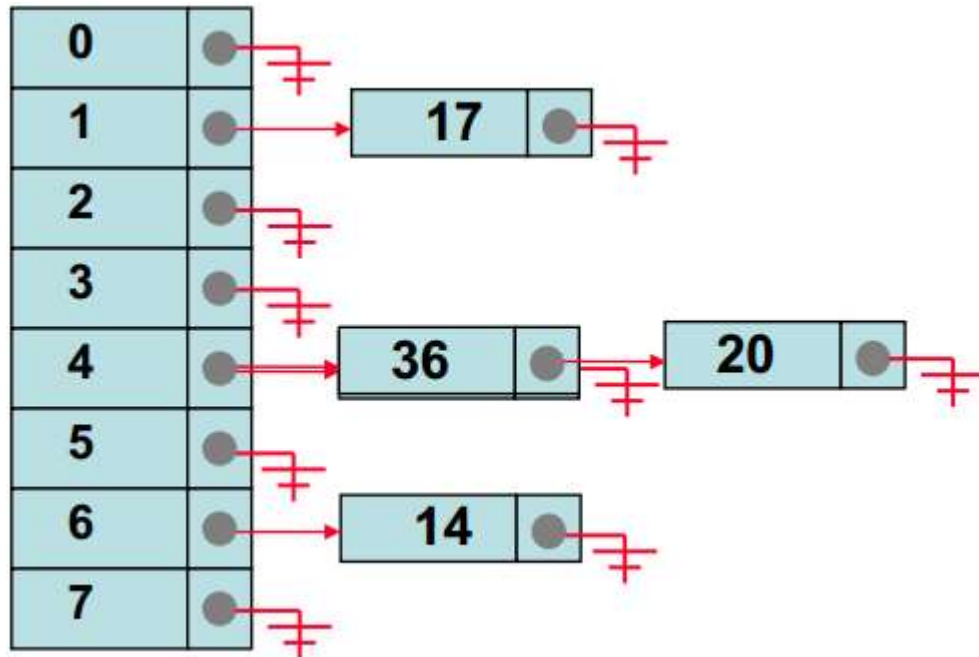
► Exemplo : Inserir 17





Encadeamento

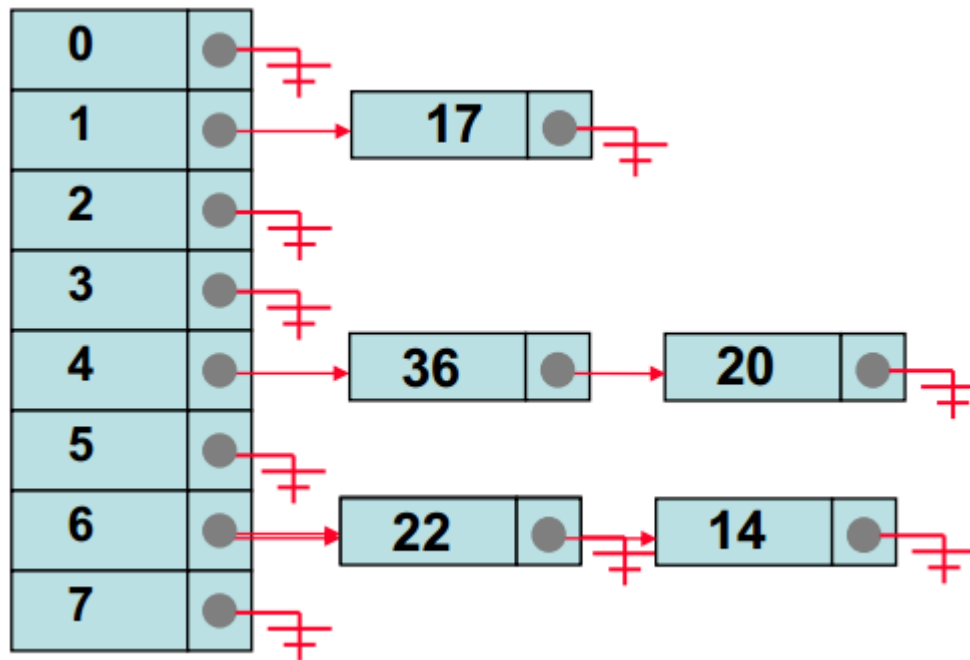
► Exemplo : Inserir 36





Encadeamento

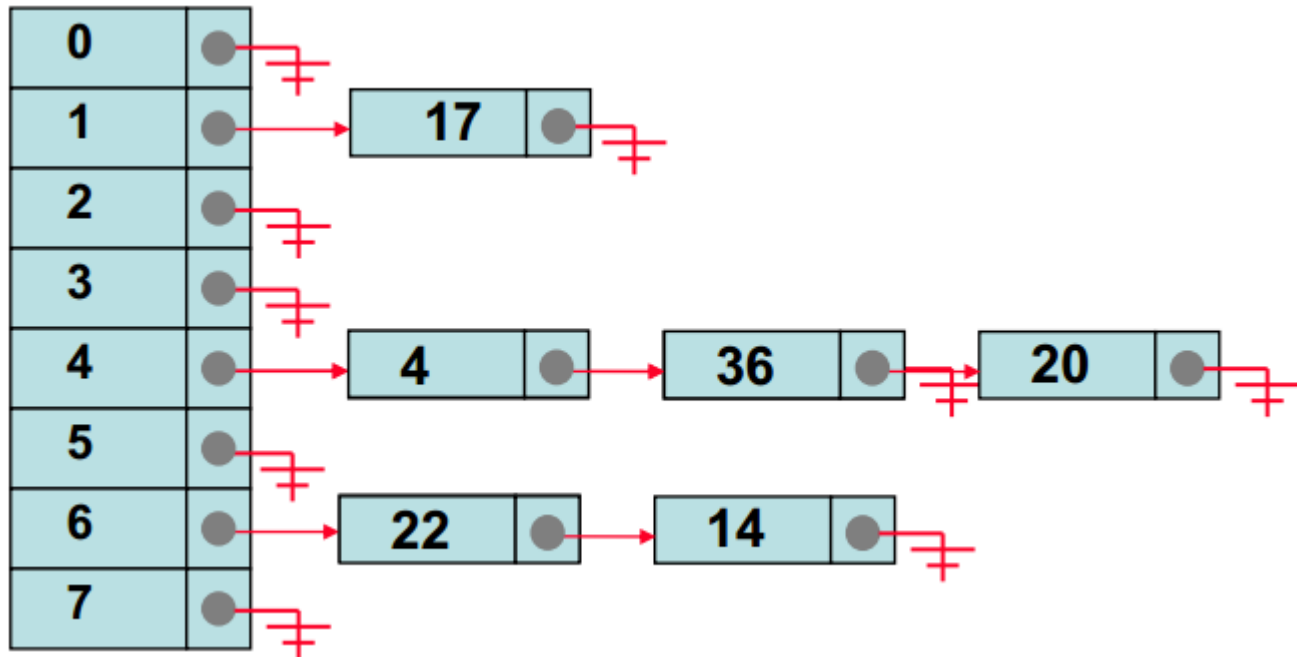
► Exemplo : Inserir 22





Encadeamento

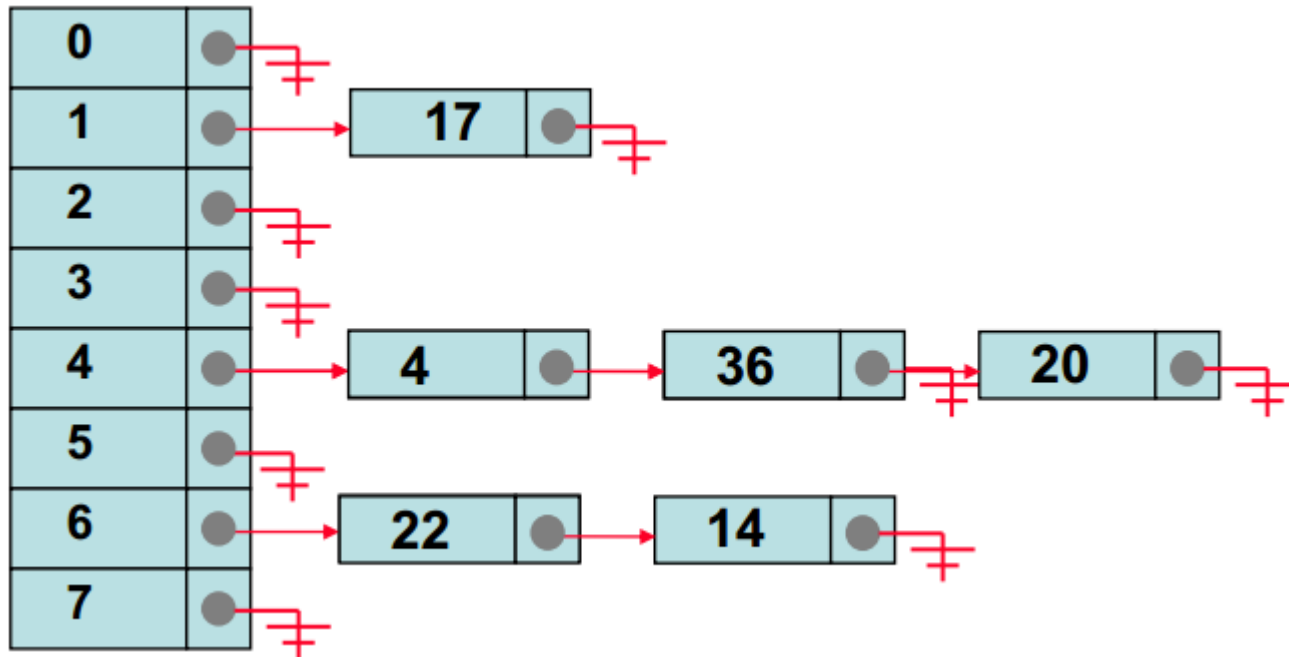
► Exemplo : Inserir 4





Encadeamento

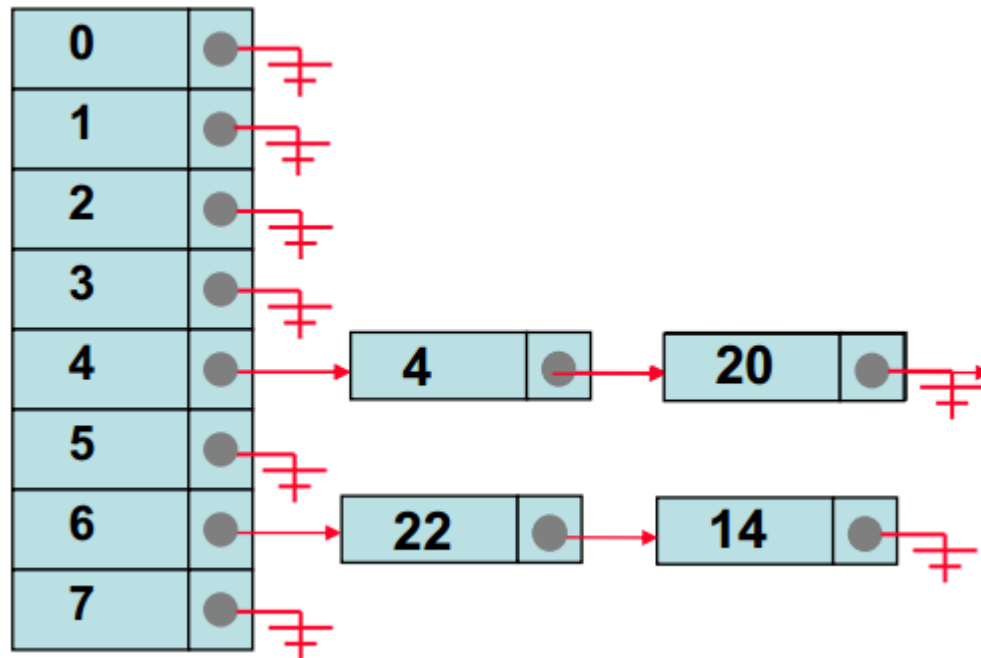
► Exemplo : Remover 17





Encadeamento

► Exemplo : Remover 36





Encadeamento

- ▶ Vantagens
- ▶ Desvantagens
- ▶ Complexidade
 - ▶ Inserção : $\theta(1)$
 - ▶ Remoção : $\theta(n)$
 - ▶ Pesquisa : $\theta(n)$





Tratamento de Colisões

- ▶ Como as colisões são inevitáveis, é preciso arrumar formas de tratá-las.
- ▶ Resolução de colisões por encadeamento
- ▶ **Endereçamento Aberto**





Endereçamento Aberto

- ▶ Neste tipo de implementação, a tabela hash é um vetor com m posições.
- ▶ Todas as chaves são armazenadas na própria tabela sem a necessidade de espaços extras ou ponteiros.
- ▶ Este método é aplicado quando o número de chaves a serem armazenadas é reduzido e as posições vazias na tabela são usadas para o tratamento de colisões.





Endereçamento Aberto

- ▶ Quando uma chave x é endereçada na posição $h(x)$ e esta já está ocupada, outras posições vazias na tabela são procuradas para armazenar x .
- ▶ Caso nenhuma seja encontrada, a tabela está totalmente preenchida e x não pode ser armazenada.
- ▶ Há duas maneiras de efetuar a busca por uma posição livre para armazenar : sondagem linear e sondagem quadrática.





Endereçamento Aberto

- ▶ Sondagem Linear

$$h'(x) = (h(x) + j) \bmod m$$

- ▶ $1 \leq j \leq m$ $h(x) = x \bmod m$

- ▶ O objetivo é armazenar a chave no endereço consecutivo $h(x)+1, h(x)+2, \dots$, até encontrar uma posição vazia.





Endereçamento Aberto

► Sondagem Linear – Exemplo

Índice	Situação	Chave
0	L	
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	





Endereçamento Aberto

► Sondagem Linear – Inserir 16

Índice	Situação	Chave
0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	





Endereçamento Aberto

► Sondagem Linear – Inserir 23

Índice	Situação	Chave
0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23





Endereçamento Aberto

► Sondagem Linear – Inserir 41

Índice	Situação	Chave
0	O	16
1	O	41
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23





Endereçamento Aberto

► Sondagem Linear – Inserir 25

Índice	Situação	Chave
0	O	16
1	O	41
2	O	25
3	L	
4	L	
5	L	
6	L	
7	O	23





Endereçamento Aberto

► Sondagem Linear – Inserir 39

Índice	Situação	Chave
0	O	16
1	O	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23





Endereçamento Aberto

▶ Sondagem Linear

- ▶ A operação de remoção é delicada
- ▶ Não se pode remover de fato uma chave do endereço, pois haveria perda da sequência de tentativas.
- ▶ Com isso, cada endereço da tabela é marcado como livre (L), ocupado (O) ou removido (R).
 - ▶ Livre quando a posição ainda não foi usada,
 - ▶ Ocupado quando uma chave está armazenada
 - ▶ Removido quando armazena uma chave que já foi removida.





Endereçamento Aberto

► Sondagem Linear – Remove 41

Índice	Situação	Chave
0	O	16
1	R	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23





Endereçamento Aberto

► Sondagem Linear – Remover 25

Índice	Situação	Chave
0	O	16
1	R	41
2	R	25
3	O	39
4	L	
5	L	
6	L	
7	R	23





Tabelas Hash – Vantagens

- ▶ **Simplicidade**

- ▶ É muito fácil de imaginar um algoritmo para implementar hashing.

- ▶ **Escalabilidade**

- ▶ Podemos adequar o tamanho da tabela de hashing ao n esperado em nossa aplicação.

- ▶ **Eficiência para n grandes**

- ▶ Para trabalharmos com problemas envolvendo $n = 1.000.000$ de dados, podemos imaginar uma tabela de *hashing* com 2.000 entradas, onde temos uma divisão do espaço de busca da ordem de $n/2.000$ de imediato.

- ▶ **Aplicação imediata a arquivos**

- ▶ Os métodos de *hashing*, tanto de endereçamento aberto como fechado, podem ser utilizados praticamente sem nenhuma alteração em um ambiente de dados persistentes utilizando arquivos em disco.





Tabelas Hash – Desvantagens

- ▶ Dependência da escolha de função de *hashing*
- ▶ Tempo médio de acesso é ótimo somente em uma faixa
 - ▶ A complexidade linear implica em um crescimento mais rápido em relação a n do que as árvores, p.ex.
- ▶ Existe uma faixa de valores de n , determinada por b , onde o *hashing* será muito melhor do que uma árvore.
 - ▶ Fora dessa faixa é pior.





Exercício

- ▶ Desenhe a tabela *hash* de tamanho 11 resultante do uso da função de *hash* para as chaves definidas abaixo, assumindo que colisões são tratadas por :
 - ▶ encadeamento
 - ▶ hash linear

$$h(k) = (2k + 5) \bmod 11$$

- ▶ 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, e 5

