

# AED III

## Tabela Hash

Ciência da Computação – IFSULDEMINAS

Primeiro Semestre de 2014

# Roteiro

- 1 Introdução
- 2 Hashing - Método de Pesquisa
- 3 Colisões
- 4 Considerações Finais

# Contextualização

- Dado um conjunto de pares (**chave, valor**):
  - ▶ determinar se há uma chave e um valor associado
  - ▶ inserir um novo par no conjunto
  - ▶ remover um par do conjunto
- Estruturas que podem ser usadas:
  - ▶ tabelas simples (vetores ou listas)
  - ▶ árvores de busca
  - ▶ tabelas de espalhamento (hash)

# Contextualização

- Dado um conjunto de pares (**chave, valor**):
  - ▶ determinar se há uma chave e um valor associado
  - ▶ inserir um novo par no conjunto
  - ▶ remover um par do conjunto
- Estruturas que podem ser usadas:
  - ▶ tabelas simples (vetores ou listas)
  - ▶ árvores de busca
  - ▶ tabelas de espalhamento (hash)

# Contextualização

- Os métodos de pesquisa vistos até agora buscam informações armazenadas com base na comparação de suas chaves.
- Para obtermos algoritmos eficientes, armazenamos os elementos ordenados e tiramos proveito dessa ordenação.
- Conclusão: os algoritmos mais eficientes de busca mostrados até o momento demandam esforço computacional  $O(\log n)$ , quando usamos uma tabela hash, esta pode realizar tais operações em tempo esperado  $O(1)$ .
- Veremos agora, o método de pesquisa conhecido como **hashing** (**tabela de dispersão, espalhamento, indexação, escrutínio ou método de cálculo de endereço**). Na maioria dos casos é possível encontrar a chave com apenas UMA OPERAÇÃO de LEITURA.

# Contextualização

- Os métodos de pesquisa vistos até agora buscam informações armazenadas com base na comparação de suas chaves.
- Para obtermos algoritmos eficientes, armazenamos os elementos ordenados e tiramos proveito dessa ordenação.
- Conclusão: os algoritmos mais eficientes de busca mostrados até o momento demandam esforço computacional  $O(\log n)$ , quando usamos uma tabela hash, esta pode realizar tais operações em tempo esperado  $O(1)$ .
- Veremos agora, o método de pesquisa conhecido como **hashing** (**tabela de dispersão, espalhamento, indexação, escrutínio ou método de cálculo de endereço**). Na maioria dos casos é possível encontrar a chave com apenas UMA OPERAÇÃO de LEITURA.

# Contextualização

- Os métodos de pesquisa vistos até agora buscam informações armazenadas com base na comparação de suas chaves.
- Para obtermos algoritmos eficientes, armazenamos os elementos ordenados e tiramos proveito dessa ordenação.
- Conclusão: os algoritmos mais eficientes de busca mostrados até o momento demandam esforço computacional  $O(\log n)$ , quando usamos uma tabela hash, esta pode realizar tais operações em tempo esperado  $O(1)$ .
- Veremos agora, o método de pesquisa conhecido como **hashing** (**tabela de dispersão, espalhamento, indexação, escrutínio ou método de cálculo de endereço**). Na maioria dos casos é possível encontrar a chave com apenas UMA OPERAÇÃO de LEITURA.

# Contextualização

- Os métodos de pesquisa vistos até agora buscam informações armazenadas com base na comparação de suas chaves.
- Para obtermos algoritmos eficientes, armazenamos os elementos ordenados e tiramos proveito dessa ordenação.
- Conclusão: os algoritmos mais eficientes de busca mostrados até o momento demandam esforço computacional  $O(\log n)$ , quando usamos uma tabela hash, esta pode realizar tais operações em tempo esperado  $O(1)$ .
- Veremos agora, o método de pesquisa conhecido como **hashing (tabela de dispersão, espalhamento, indexação, escrutínio ou método de cálculo de endereço)**. Na maioria dos casos é possível encontrar a chave com apenas UMA OPERAÇÃO de LEITURA.

# Contextualização

- Em algumas aplicações, é necessário obter o valor com poucas comparações, logo, é preciso saber a posição em que o elemento se encontra, sem precisar varrer todas as chaves.
- A estrutura com tal propriedade é chamada de **tabela hash**.

0	1	2	3	4	5	6	7
64			11	20	45		7

- Como sabemos se 11, 20 e 45 está no vetor?

# Definição de Hash

## Definição de hash

- Hash é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo **dicionário**.
- Dicionários são estruturas especializadas em prover as operações de **inserção, pesquisa e remoção** e que **não admitem repetições**.
- A ideia central do Hash é utilizar uma **função** (Função de Hashing), aplicada sobre parte da informação (**chave**), para retornar o **índice** onde a informação deve ou deveria estar armazenada.
- A estrutura de dados Hash é comumente chamada de **Tabela Hash**.

# Definição de Hash

## Definição de hash

- Hash é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo **dicionário**.
- **Dicionários** são estruturas especializadas em prover as operações de **inserção, pesquisa e remoção** e que **não admitem repetições**.
- A ideia central do Hash é utilizar uma **função** (Função de Hashing), aplicada sobre parte da informação (**chave**), para retornar o **índice** onde a informação deve ou deveria estar armazenada.
- A estrutura de dados Hash é comumente chamada de **Tabela Hash**.

# Definição de Hash

## Definição de hash

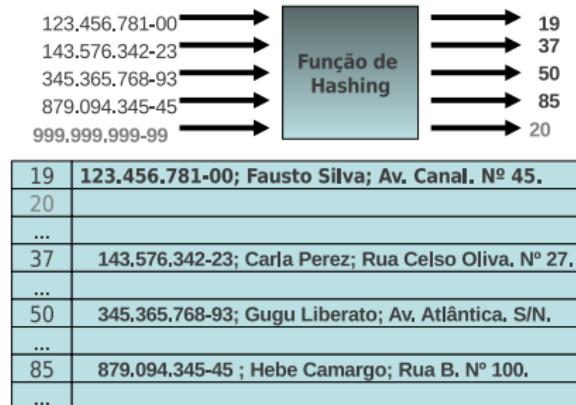
- Hash é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo **dicionário**.
- **Dicionários** são estruturas especializadas em prover as operações de **inserção, pesquisa e remoção** e que **não admitem repetições**.
- A ideia central do Hash é utilizar uma **função** (Função de Hashing), aplicada sobre parte da informação (**chave**), para retornar o **índice** onde a informação deve ou deveria estar armazenada.
- A estrutura de dados Hash é comumente chamada de **Tabela Hash**.

# Definição de Hash

## Definição de hash

- Hash é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo **dicionário**.
- **Dicionários** são estruturas especializadas em prover as operações de **inserção, pesquisa e remoção** e que **não admitem repetições**.
- A ideia central do Hash é utilizar uma **função** (Função de Hashing), aplicada sobre parte da informação (**chave**), para retornar o **índice** onde a informação deve ou deveria estar armazenada.
- A estrutura de dados Hash é comumente chamada de **Tabela Hash**.

# Definição de Hash



# Tabela Hash

## Tabela Hash

- Em Computação a **Tabela Hash** é uma estrutura de dados especial, que armazena as informações desejadas associando **chaves de pesquisa** a estas **informações**.
- Objetivo:** a partir de uma chave, fazer uma busca rápida e obter o valor desejado.
- A ideia central por trás da construção de uma Tabela Hash é identificar, na chave de busca, quais as partes que são significativas.

	chave	dados
1		
2		
X	K	registro
n-1		
n		

# Tabela Hash

## Tabela Hash

- Em Computação a **Tabela Hash** é uma estrutura de dados especial, que armazena as informações desejadas associando **chaves de pesquisa** a estas **informações**.
- Objetivo:** a partir de uma chave, fazer uma busca rápida e obter o valor desejado.
- A ideia central por trás da construção de uma Tabela Hash é identificar, na chave de busca, quais as partes que são significativas.

	chave	dados
1		
2		
X	K	registro
n-1		
n		

# Tabela Hash

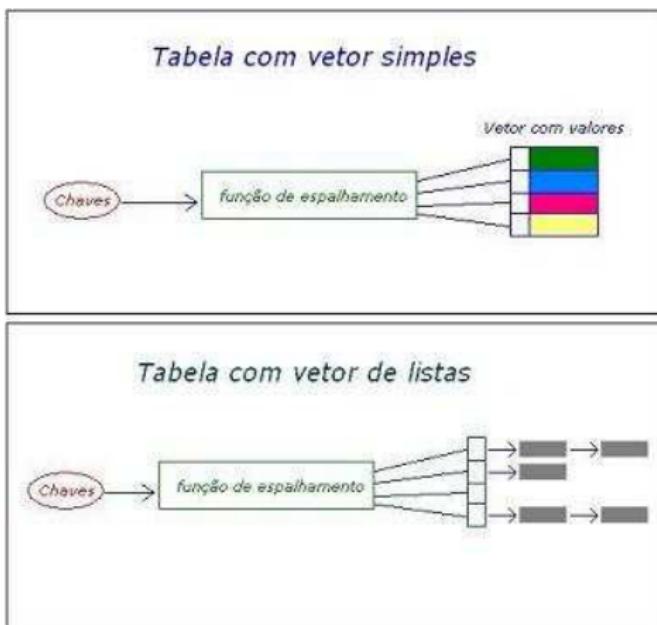
## Tabela Hash

- Em Computação a **Tabela Hash** é uma estrutura de dados especial, que armazena as informações desejadas associando **chaves de pesquisa** a estas **informações**.
- Objetivo:** a partir de uma chave, fazer uma busca rápida e obter o valor desejado.
- A ideia central por trás da construção de uma Tabela Hash é identificar, na chave de busca, quais as partes que são significativas.

	chave	dados
1		
2		
X	K	registro
n-1		
n		

# Como representar tabela hash?

- **Vetor:** cada posição do vetor guarda uma informação.
- **Vetor + Lista Encadeada:** o vetor contém ponteiros para as listas que representam as informações.



# Função de Hashing

## Função de Hashing

- A Função de Hashing é a responsável por gerar um índice a partir de uma determinada chave.
- O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis.
- Características desejáveis: eficiência e bom espalhamento.
- A função de Hashing é extremamente importante, pois ela é responsável por distribuir as informações pela Tabela Hash.
- A implementação da função de Hashing tem influência direta na eficiência das operações sobre o Hash.

# Função de Hashing

## Função de Hashing

- A Função de Hashing é a responsável por gerar um índice a partir de uma determinada chave.
- O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis.
- Características desejáveis: eficiência e bom espalhamento.
- A função de Hashing é extremamente importante, pois ela é responsável por distribuir as informações pela Tabela Hash.
- A implementação da função de Hashing tem influência direta na eficiência das operações sobre o Hash.

# Função de Hashing

## Função de Hashing

- A Função de Hashing é a responsável por gerar um índice a partir de uma determinada chave.
- O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis.
- Características desejáveis: eficiência e bom espalhamento.
- A função de Hashing é extremamente importante, pois ela é responsável por distribuir as informações pela Tabela Hash.
- A implementação da função de Hashing tem influência direta na eficiência das operações sobre o Hash.

# Função de Hashing

## Função de Hashing

- A Função de Hashing é a responsável por gerar um índice a partir de uma determinada chave.
- O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis.
- Características desejáveis: eficiência e bom espalhamento.
- A função de Hashing é extremamente importante, pois ela é responsável por distribuir as informações pela Tabela Hash.
- A implementação da função de Hashing tem influência direta na eficiência das operações sobre o Hash.

# Função de Hashing

## Função de Hashing

- A Função de Hashing é a responsável por gerar um índice a partir de uma determinada chave.
- O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis.
- Características desejáveis: eficiência e bom espalhamento.
- A função de Hashing é extremamente importante, pois ela é responsável por distribuir as informações pela Tabela Hash.
- A implementação da função de Hashing tem influência direta na eficiência das operações sobre o Hash.

# Função de Hashing

- **Método da divisão** (mais usado):

- ▶ Usa o resto da divisão

$$H(K) = K \bmod M$$

- ▶ Onde **K** é um inteiro correspondente à chave.
  - ▶ **M** é um inteiro correspondente ao tamanho da tabela.

- **As chaves não numéricas:**

$$K = \sum_{i=1}^n \text{Chave}[i] \times p[i]$$

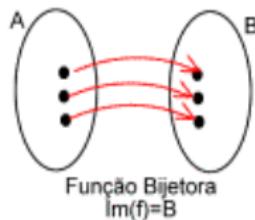
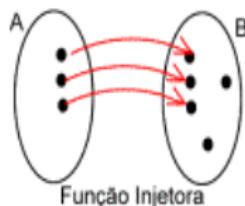
- ▶ **n** é o número de caracteres da chave.
  - ▶ **Chave[i]** corresponde à representação ASCII do i-ésimo caracter da chave.
  - ▶ **p[i]** é um inteiro de um conjunto de pesos gerado randomicamente.

# Função de Hashing

- Uma boa função hash (ou de hashing) deve apresentar duas propriedades básicas:
  - ▶ seu cálculo deve ser rápido
  - ▶ deve gerar poucas colisões
- Escolha de funções de hashings apropriadas tentam minimizar a probabilidade de ocorrência de colisões.

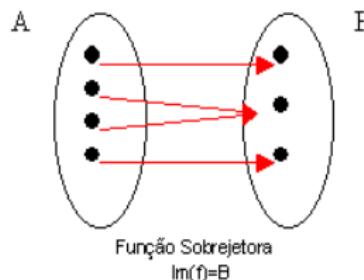
# Hashing Perfeito

- Para quaisquer chaves  $x$  e  $y$  diferentes e pertencentes a  $A$ , a função utilizada fornece saídas diferentes.



# Hashing Imperfeito

- Existe chaves  $x$  e  $y$  diferentes e pertencentes a  $A$ , onde a função Hash utilizada fornece saídas iguais.



## Exemplo - Hashing Imperfeito

- Suponha que queiramos armazenar as seguintes chaves: **C, H, A, V, E** e **S** em um vetor de  $P = 7$  posições conforme a seguinte função  $h(K) = K_{\text{ASCII}} \% P$ .
- Tabela ASCII: C (67); H (72); A (65); V (86); E (69) e S (83).

K	K <sub>ASCII</sub>	h(k)
C	67	4
H	72	2
A	65	2
V	86	2
E	69	6
S	83	6

# Considerações sobre funções de hashing

- Na prática funções de hashing perfeitas ou quase perfeitas:
  - ▶ são encontradas apenas onde a colisão não é tolerável;
  - ▶ quando conhecemos previamente o conteúdo a ser armazenado na tabela.
- Nas Tabelas Hash comuns a colisão é indesejável, pois diminui o desempenho do sistema.
- Por causa das colisões, muitas Tabelas Hash são aliadas com outras estruturas de dados:
  - ▶ Listas Encadeadas, Árvores Balanceadas, etc

# Considerações sobre funções de hashing

- Na prática funções de hashing perfeitas ou quase perfeitas:
  - ▶ são encontradas apenas onde a colisão não é tolerável;
  - ▶ quando conhecemos previamente o conteúdo a ser armazenado na tabela.
- Nas Tabelas Hash comuns a colisão é indesejável, pois diminui o desempenho do sistema.
- Por causa das colisões, muitas Tabelas Hash são aliadas com outras estruturas de dados:
  - ▶ Listas Encadeadas, Árvores Balanceadas, etc

# Considerações sobre funções de hashing

- Na prática funções de hashing perfeitas ou quase perfeitas:
  - ▶ são encontradas apenas onde a colisão não é tolerável;
  - ▶ quando conhecemos previamente o conteúdo a ser armazenado na tabela.
- Nas Tabelas Hash comuns a colisão é indesejável, pois diminui o desempenho do sistema.
- Por causa das colisões, muitas Tabelas Hash são aliadas com outras estruturas de dados:
  - ▶ Listas Encadeadas, Árvores Balanceadas, etc

# Colisões

- Ocorrem quando duas ou mais chaves geram o mesmo endereço na Tabela Hash.
- É comum ocorrer colisões.
- Principais causas:
  - ▶ O número de chaves possíveis é muito maior que o número de entradas disponíveis na tabela.
  - ▶ Não é possível garantir que as funções de hashing possuam um bom potencial de distribuição (espalhamento).

# Colisões

- Ocorrem quando duas ou mais chaves geram o mesmo endereço na Tabela Hash.
- É comum ocorrer colisões.
- Principais causas:
  - ▶ O número de chaves possíveis é muito maior que o número de entradas disponíveis na tabela.
  - ▶ Não é possível garantir que as funções de hashing possuam um bom potencial de distribuição (espalhamento).

# Colisões

- Ocorrem quando duas ou mais chaves geram o mesmo endereço na Tabela Hash.
- É comum ocorrer colisões.
- **Principais causas:**
  - ▶ O número de chaves possíveis é muito maior que o número de entradas disponíveis na tabela.
  - ▶ Não é possível garantir que as funções de hashing possuam um bom potencial de distribuição (espalhamento).

# Tratamento de colisões

- Um bom método de resolução de colisões é essencial, não importando a qualidade da função de hashing.
- Há diversas técnicas de resolução de colisão.
- Técnicas mais comuns
  - ▶ Encadeamento (Hashing Aberto);
  - ▶ Endereçamento Aberto (Rehash ou Hashing Fechado);

# Tratamento de colisões

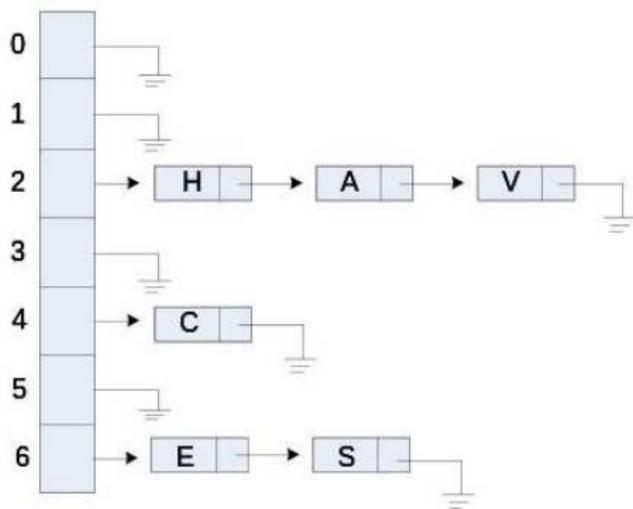
- Um bom método de resolução de colisões é essencial, não importando a qualidade da função de hashing.
- Há diversas técnicas de resolução de colisão.
  - Técnicas mais comuns
    - ▶ Encadeamento (Hashing Aberto);
    - ▶ Endereçamento Aberto (Rehash ou Hashing Fechado);

# Tratamento de colisões

- Um bom método de resolução de colisões é essencial, não importando a qualidade da função de hashing.
- Há diversas técnicas de resolução de colisão.
- Técnicas mais comuns
  - ▶ Encadeamento (Hashing Aberto);
  - ▶ Endereçamento Aberto (Rehash ou Hashing Fechado);

# Encadeamento

- A informação é armazenada em estruturas encadeadas fora da Tabela Hash. Ou seja, mantém numa lista ligada das chaves que levam a um mesmo índice na tabela de hashing.
- **Exemplo:** Suponha que queiramos armazenar as seguintes chaves: **C**, **H**, **A**, **V**, **E** e **S** em um vetor de **P = 7** posições conforme a seguinte função  $f(K) = K_{\text{ASCII}} \% P$ .



# Encadeamento - Operações

## Busca

Hash-Search( $T$ ,  $k$ )

procure um elemento com chave  $k$   
na lista  $T[h(k)]$  e devolva seu ponteiro

## Inserção

Hash-Insert( $T$ ,  $x$ )

insira  $x$  na cabeça da lista  $T[h(x)]$

## Remoção

Hash-Delete( $T$ ,  $x$ )

remova  $x$  da lista  $T[h(x)]$

# Endereçamento

- A estratégia é utilizar o próprio espaço da tabela que ainda não foi ocupado para armazenar a chave que gerou a colisão.
- Utilizado quando o número de registros a serem armazenados na tabela pode ser estimado.
- **ReHash linear:** a posição na tabela é dada por:

$$rh(k, i) = (k + i) \% M$$

- ▶  $M$  é o tamanho da Tabela Hash
- ▶  $k$  é a chave
- ▶  $i$  é o índice de reaplicação do Hash

# Endereçamento

- A estratégia é utilizar o próprio espaço da tabela que ainda não foi ocupado para armazenar a chave que gerou a colisão.
- Utilizado quando o número de registros a serem armazenados na tabela pode ser estimado.
- **ReHash linear:** a posição na tabela é dada por:

$$rh(k, i) = (k + i) \% M$$

- ▶  $M$  é o tamanho da Tabela Hash
- ▶  $k$  é a chave
- ▶  $i$  é o índice de reaplicação do Hash

## Endereçamento

- A estratégia é utilizar o próprio espaço da tabela que ainda não foi ocupado para armazenar a chave que gerou a colisão.
- Utilizado quando o número de registros a serem armazenados na tabela pode ser estimado.
- **ReHash linear:** a posição na tabela é dada por:

$$rh(k, i) = (k + i) \% M$$

- ▶ **M** é o tamanho da Tabela Hash
- ▶ **k** é a chave
- ▶ **i** é o índice de reaplicação do Hash

## Exemplo - Rehash Linear

- Suponha que queiramos inserir os caracteres da palavra **CHAVES** utilizando o método de resolução de colisões Hash Linear.
- O número de entradas da Tabela Hash a ser criada é 7.

K	K <sub>ASCII</sub>	i <sub>1</sub> = h(k)	i <sub>2</sub> = rh(k)	i <sub>3</sub> = rh(k)	i <sub>4</sub> = rh(k)
C	67	4	-	-	-
H	72	2	-	-	-
A	65	2	3	-	-
V	86	2	3	4	5
E	69	6	-	-	-
S	83	6	0	-	-

# Exemplo - Rehash Linear

- Tabela Hash:

S
H
A
C
V
E

# Endereçamento

## Características

- A função hash calcula uma posição para uma chave.
- Se a posição está livre a chave é armazenada.
- Caso contrário, verifica-se a posição seguinte.
- A busca por uma posição continua até uma posição livre seja encontrada ou até que a capacidade da tabela seja esgotada.
- **Obs:** neste tipo de tratamento considera-se a tabela como uma estrutura circular, onde a primeira posição sucede a última posição.

# Endereçamento

## Características

- A função hash calcula uma posição para uma chave.
- Se a posição está livre a chave é armazenada.
- Caso contrário, verifica-se a posição seguinte.
- A busca por uma posição continua até uma posição livre seja encontrada ou até que a capacidade da tabela seja esgotada.
- **Obs:** neste tipo de tratamento considera-se a tabela como uma estrutura circular, onde a primeira posição sucede a última posição.

# Endereçamento

## Características

- A função hash calcula uma posição para uma chave.
- Se a posição está livre a chave é armazenada.
- Caso contrário, verifica-se a posição seguinte.
- A busca por uma posição continua até uma posição livre seja encontrada ou até que a capacidade da tabela seja esgotada.
- **Obs:** neste tipo de tratamento considera-se a tabela como uma estrutura circular, onde a primeira posição sucede a última posição.

# Endereçamento

## Características

- A função hash calcula uma posição para uma chave.
- Se a posição está livre a chave é armazenada.
- Caso contrário, verifica-se a posição seguinte.
- A busca por uma posição continua até uma posição livre seja encontrada ou até que a capacidade da tabela seja esgotada.
- Obs: neste tipo de tratamento considera-se a tabela como uma estrutura circular, onde a primeira posição sucede a última posição.

## Características

- A função hash calcula uma posição para uma chave.
- Se a posição está livre a chave é armazenada.
- Caso contrário, verifica-se a posição seguinte.
- A busca por uma posição continua até uma posição livre seja encontrada ou até que a capacidade da tabela seja esgotada.
- **Obs:** neste tipo de tratamento considera-se a tabela como uma estrutura circular, onde a primeira posição sucede a última posição.

# Endereçamento - Operações

## Busca

```
Hash-Search(T, k)
```

```
    i = 0
```

```
repeat
```

```
    j = h(k, i)
```

```
    if T[j] == k
```

```
        return j
```

```
    i = i + 1
```

```
until T[j] == NIL or i == m
```

```
return NIL
```

# Endereçamento - Operações

## Inserção

```
Hash-Insert(T, k)
    i = 0
    repeat
        j = h(k, i)
        if T[j] = NIL or T[j] = DELETED
            T[j] = k
            return j
        else i = i + 1
    until i == m
    error "Estouro da tabela hash"
```

# Endereçamento - Operações

## Remoção

```
Hash-Delete(T, k)
    i = 0
    repeat
        j = h(k, i)
        if T[j] == k
            T[j] = DELETED
            return j;
        i = i + 1
    until T[j] == NIL or i == m
```

# Limitações

- O Hash é uma estrutura de dados do tipo dicionário:
  - ▶ Não permite armazenar elementos repetidos.
  - ▶ Não permite recuperar elementos sequencialmente (ordenado).
- Para otimizar a função Hash é necessário conhecer a natureza e domínio da chave a ser utilizada.
- No pior caso a complexidade das operações pode ser  $O(n)$ . Caso em que todos os elementos inseridos colidirem.

# Limitações

- O Hash é uma estrutura de dados do tipo dicionário:
  - ▶ Não permite armazenar elementos repetidos.
  - ▶ Não permite recuperar elementos sequencialmente (ordenado).
- Para otimizar a função Hash é necessário conhecer a natureza e domínio da chave a ser utilizada.
- No pior caso a complexidade das operações pode ser  $O(n)$ . Caso em que todos os elementos inseridos colidirem.

# Limitações

- O Hash é uma estrutura de dados do tipo dicionário:
  - ▶ Não permite armazenar elementos repetidos.
  - ▶ Não permite recuperar elementos sequencialmente (ordenado).
- Para otimizar a função Hash é necessário conhecer a natureza e domínio da chave a ser utilizada.
- No pior caso a complexidade das operações pode ser  $O(n)$ . Caso em que todos os elementos inseridos colidirem.

# Vantagens x Desvantagens

- **Vantagens:**

- ▶ Algoritmos simples e eficientes para inserção, retirada e busca.
- ▶ Alta eficiência no custo de pesquisa, que é  $O(1)$  para o caso médio.

- **Desvantagens:**

- ▶ Não há garantias de balanceamento:
  - ★ Espaço sub-utilizado nas tabelas.
  - ★ O grau de espalhamento é sensível à função de hashing utilizada e ao tipo de informação usada como chave.

# Vantagens x Desvantagens

- **Vantagens:**

- ▶ Algoritmos simples e eficientes para inserção, retirada e busca.
- ▶ Alta eficiência no custo de pesquisa, que é  $O(1)$  para o caso médio.

- **Desvantagens:**

- ▶ Não há garantias de balanceamento:
  - ★ Espaço sub-utilizado nas tabelas.
  - ★ O grau de espalhamento é sensível à função de hashing utilizada e ao tipo de informação usada como chave.

# Exercícios

- Ilustre a organização final de uma Tabela Hash após a inserção das seguintes chaves: 35, 99, 27, 18, 65, 45. Considere a tabela com tamanho 6 e que os números possíveis de chaves estão no intervalo entre 1 a 100. Faça o tratamento de colisões utilizando encadeamento e endereçamento.