



# Tabelas de Espalhamento

Prof. Dr. Wesin Ribeiro Alves

# Neste capítulo

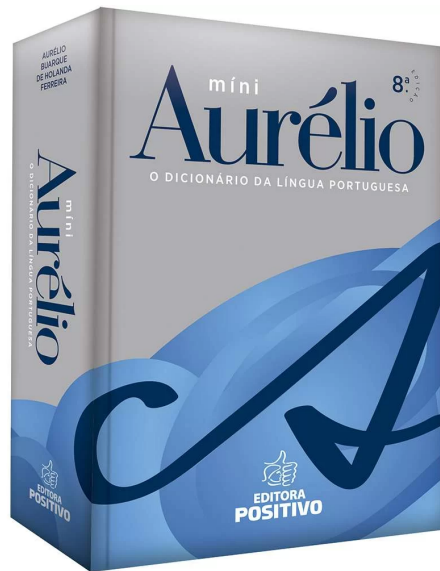
- ❑ Introdução
- ❑ Tabelas de endereço direto
- ❑ Tabelas de espalhamento
- ❑ Função Hash
- ❑ Endereçamento Aberto
- ❑ Hash perfeito
- ❑ revisão

# Introdução

Tabela de espalhamento é uma estrutura de dados eficaz para implementar **dicionários**.

Ela generaliza a noção mais simples de arranjo comum fazendo o uso do **endereçamento direto**.

Normalmente, a tabela de espalhamento é usada em situações onde precisa-se apenas de operações **inserir, buscar e remover**.



# Dicionários

A implementação das operações é trivial.

DIRECT-ADDRESS-SEARCH( $T, k$ )

1    **return**  $T[k] = x$

DIRECT-ADDRESS-INSERT( $T, x$ )

1     $T[x.chave] = x$

DIRECT-ADDRESS-DELETE( $T, x$ )

1     $T[x.chave] = \text{NIL}$

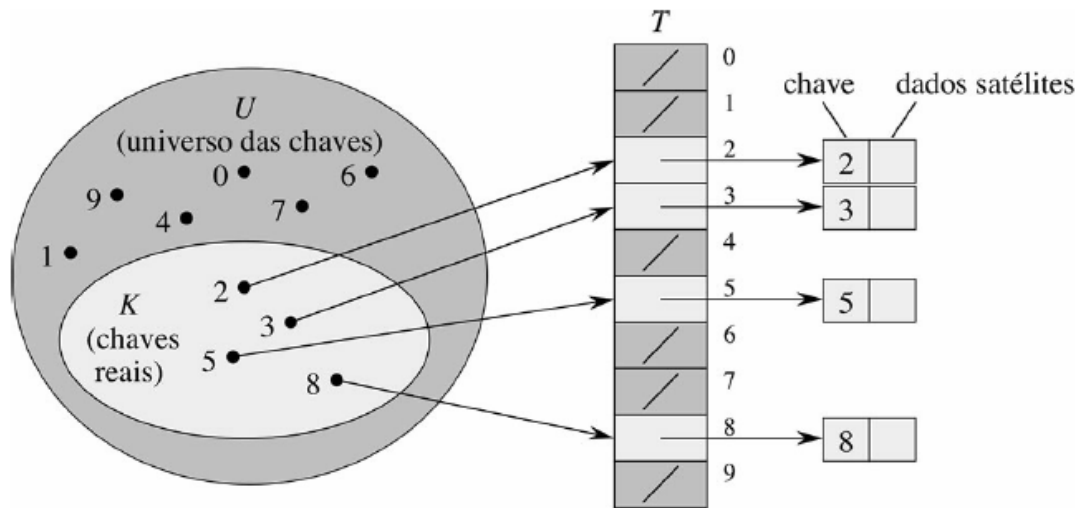
O tempo médio para  
implementação é  $O(1)$ .

# Tabelas de endereçamento direto

O endereçamento direto é uma técnica simples que funciona bem quando o número de chaves é razoavelmente pequeno.

$T$  é uma tabela de endereços diretos  $T=[0..m-1]$ , na qual, cada posição corresponde a uma chave no universo  $U = \{0..m-1\}$ .

Podemos armazenar o objeto na própria posição da tabela e assim economizar espaço.

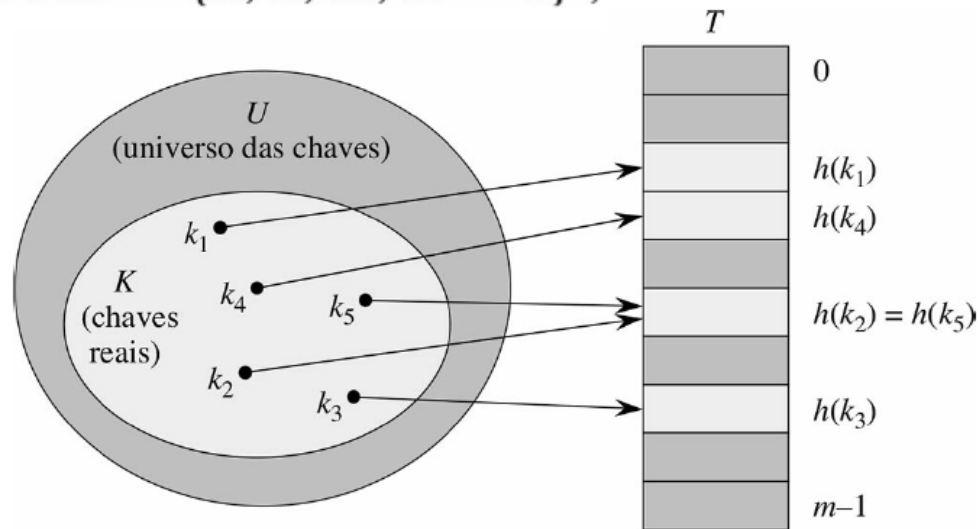


# Tabelas de espalhamento

A função hash  $h$  pode ser usada para calcular a posição da chave  $k$  no arranjo.

Quando o conjunto  $K$  de chaves armazenadas em um **dicionário** é muito menor que o universo  $U$  de todas as chaves possíveis, uma **tabela de espalhamento** requer armazenamento muito menor que uma tabela de endereços diretos.

$$h : U \rightarrow \{0, 1, \dots, m - 1\} ,$$



# O que é uma colisão?

São chaves mapeadas para a mesma posição!!!

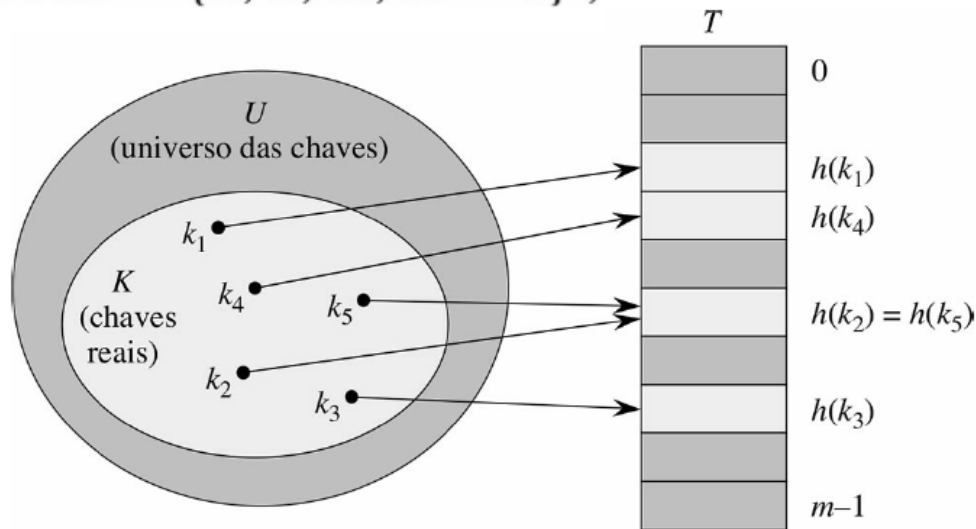
Repara na alocação da chave  $k_2$  e  $k_3$ .

Após a função hash, essas chaves foram mapeadas para a **mesma posição**.

Portanto, houve uma **colisão**.

Isso é uma situação inevitável pois  $|U| > m$

$$h : U \rightarrow \{0, 1, \dots, m - 1\} ,$$

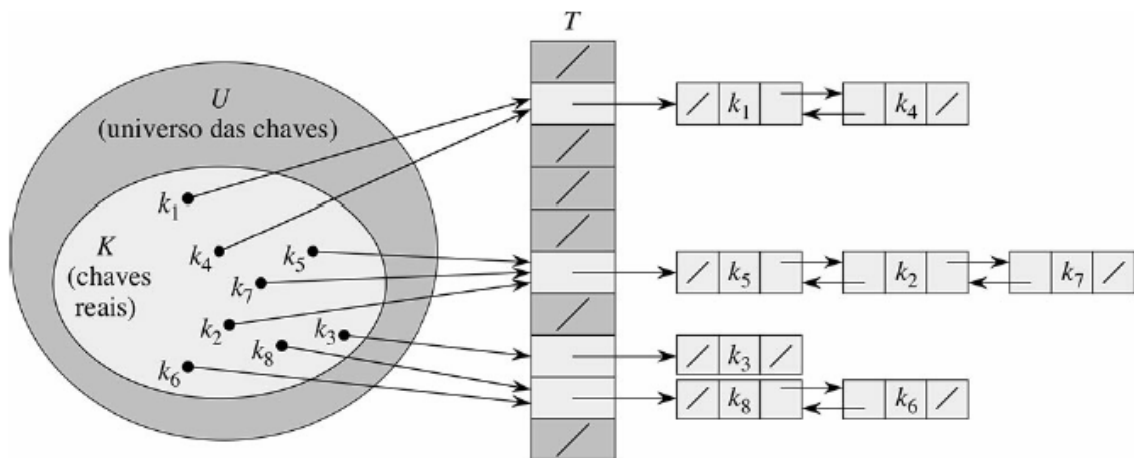


# Resolução de colisões por encadeamento

Utiliza uma lista encadeada é a maneira mais simples de resolver uma colisão.

Todos os elementos resultantes do hash vão para a **mesma posição** em uma lista encadeada.

As operações de dicionário são fáceis de implementar.





# Análise da tabela hash com encadeamento

- Complexidade de tempo
  - Inserção:  $O(1)$
  - Remoção  $O(1)$  se usar LDE
  - Busca  $\theta(1 + \alpha)$

# Fator de carga

- $n$  = número de elementos na tabela
- $m$  = capacidade da tabela
- $\alpha \Rightarrow$  fator de carga  $= n/m$
- $X_{ij} = I\{h(k_i) = h(k_j)\}$
- Assumindo o hash uniforme

$$\Pr \{h(k_i) = h(k_j)\} = 1/m$$

$$E[X_{ij}] = 1/m.$$

$$\begin{aligned} E\left[\frac{1}{n} \sum_{i=1}^n \left(1 + \sum_{j=i+1}^n X_{ij}\right)\right] \\ &= \frac{1}{n} \sum_{i=1}^n \left(1 + \sum_{j=i+1}^n E[X_{ij}]\right) \\ &= \frac{1}{n} \sum_{i=1}^n \left(1 + \sum_{j=i+1}^n \frac{1}{m}\right) \\ &= 1 + \frac{1}{nm} \left(\sum_{i=1}^n n - \sum_{i=1}^n i\right) \\ &= 1 + \frac{1}{nm} \left(n^2 - \frac{n(n+1)}{2}\right) \\ &= 1 + \frac{n-1}{2m} \\ &= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n} \end{aligned}$$

# Implementação

## *Resolução de colisões por encadeamento.*

CHAINED-HASH-INSERT( $T, x$ )

1 insere  $x$  no início da lista  $T[h(x.chave)]$

CHAINED-HASH-SEARCH( $T, k$ )

1 procura um elemento com a chave  $k$  na lista  $T[h(k)]$

CHAINED-HASH-DELETE( $T, x$ )

1 elimina  $x$  da lista  $T[h(x.chave)]$

# Funções hash

Existem três tipos de esquemas para criação de boas funções hash: por divisão, por multiplicação e por hash universal.

Uma boa função hash satisfaz (aproximadamente) a premissa do **hashing uniforme simples**.

cada chave tem **igual probabilidade** de passar para qualquer das  $m$  posições por uma operação de hash.

Uma boa função hash **minimiza** a chance de pequenas variações nos símbolos passarem para a mesma posição após o hashing.

Uma boa abordagem deriva o valor hash de um modo que esperamos seja **independente** de quaisquer padrões que possam existir nos dados.

# Método da Divisão

$$h(k) = k \bmod m$$

- $k$  é a chave do tipo inteiro que queremos fazer o hash
- $m$  é o tamanho da tabela  $T$
- O valor de  $m$  não deve ser potência de 2
- Melhor escolha seria um número primo não muito próximo da potência de 2

# Método da multiplicação

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

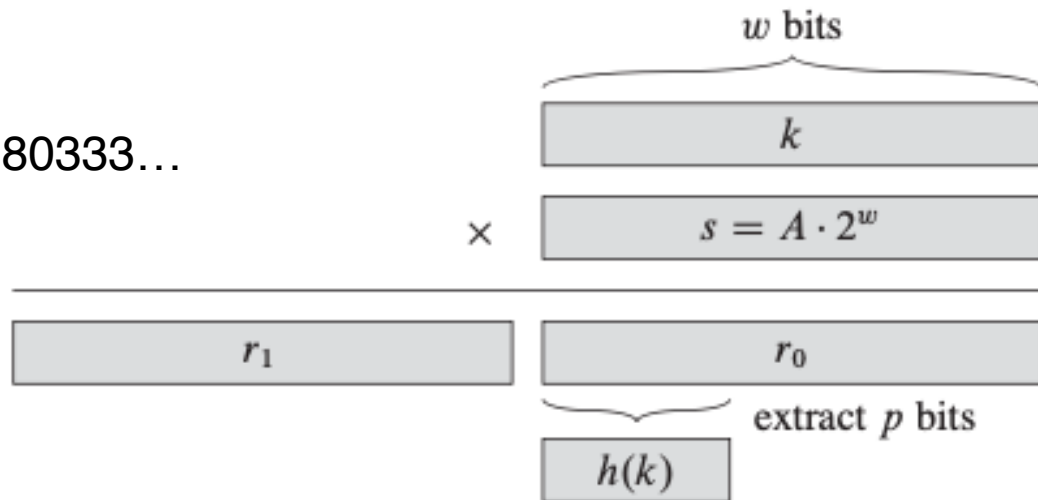
- $k$  é a chave do tipo inteiro que queremos fazer o hash
- $m$  é o tamanho da tabela  $T$
- O valor da constante  $A$  deve estar entre  $[0,1]$
- O valor de  $m$  pode ser potência de 2

# Método da multiplicação

$$h(k) = \lfloor m(k A \bmod 1) \rfloor$$

$$A = (\text{sqrt}(r) - 1) / 2 = 0,6180333\dots$$

$$0 < A < 1$$



Esquema do método da multiplicação em modo binário.  $h(k)$  corresponde aos  $p$  bits de mais alta ordem da metade inferior ( $r_0$ ) da multiplicação.

# Hash universal

- Dado uma coleção finita de funções hash  $H$
- Para cada par  $k, l$  contida em  $U$
- $|h(k) = h(l)| \sim |H| / m$

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m$$

$$\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}$$

$$0 < k < p-1$$

$$\Pr\{h_{ab}(k) = h_{ab}(l)\} \leq 1/m$$

A chance de colisão entre duas  
chaves distintas é menor ou igual a  $1/m$



## Exercícios - Resolução de colisões por encadeamento usando método da divisão

1. Insira as chaves {5, 28, 19, 15, 20, 33} em uma tabela T com 9 posições [0..8], utilizando a função hash:  $h(k) = k \bmod 9$ .
2. Se a i-ésima letra do alfabeto é representada pelo número i e a função dispersão  $h(\text{chave}) = \text{chave} \bmod M$  é utilizada para  $M = 7$ , então mostre o resultado da inserção das seguintes chaves na tabela: P E S Q U I S A.

# Endereçamento Aberto

Todos os elementos ficam na própria tabela de espalhamento. Isto é, cada entrada da tabela contém um elemento do conjunto dinâmico ou nulo.

- ❑ Não usa lista ligada
- ❑ A tabela pode ficar cheia
- ❑ Evita por completo a utilização de ponteiros
- ❑ + memória, - colisões

# Sequência de sondagem

*Pseudo código para operações de inserção e busca.*

**HASH-INSERT**( $T, k$ )

1  $i = 0$

2 **repeat**  $j = h(k, i)$

3     **if**  $T[j] == \text{NIL}$

4          $T[j] = k$

5         **return**  $j$

6     **else**  $i = i + 1$

7 **until**  $i == m$

8 **error** “estouro da tabela”

**HASH-SEARCH**( $T, k$ )

1  $i = 0$

2 **repeat**

3      $j = h(k, i)$

4     **if**  $T[j] == k$

5         **return**  $j$

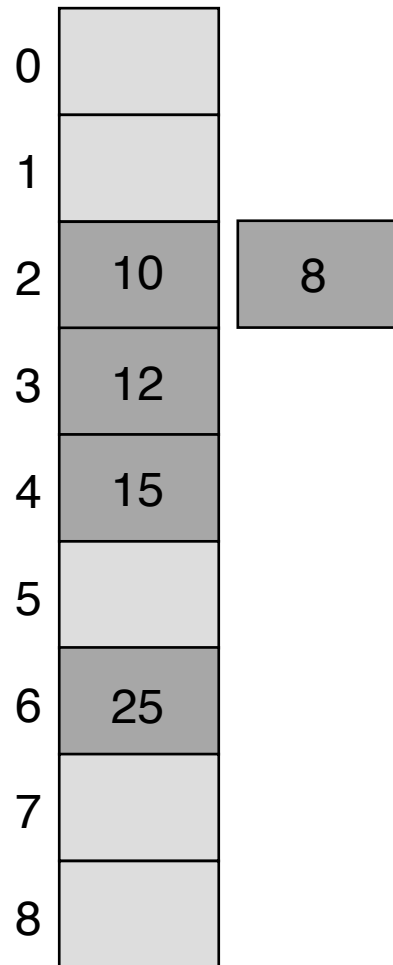
6      $i = i + 1$

7 **until**  $T[j] == \text{NIL}$  ou  $i == m$

8 **return** NIL

# Sondagem linear

- $h(k,i) = (h'(k)+i) \bmod m$
- $i=0,1,\dots,m-1$
- Fácil de implementar
- Sondagem inicial em  $T[h'(k)]$
- Problema de agrupamento primário (longas sequências de posições ocupadas)



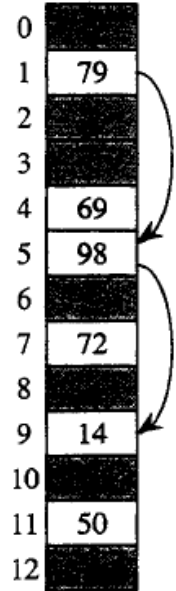
# Sondagem quadrática

- $h(k,i) = (h'(k) + c1*i + c2*i^2) \bmod m$
- Sondagem inicial em  $T[h'(k)]$
- $c1$  e  $c2$  são constantes
- $i=0,1\dots m-1$
- Problema de agrupamento secundário

0		
1		
2	10	8
3	12	
4	15	
5		
6	25	
7		
8		

# Hash duplo

- $h(k,i) = (h1(k) + i * h2(k)) \bmod m$
- $h1$  e  $h2$  são funções de hash auxiliares
- Sondagem inicial em  $T[h1(k)]$
- Melhor quando  $m$  é primo ou potência de 2



## Exercícios - Resolução de colisões por endereçamento aberto usando sondagem linear

1. Insira as chaves {10, 22, 31, 4, 15, 28, 59} em uma tabela T com 11 posições [0..10], utilizando a função hash:  $h(k) = k \bmod 11$  com sondagem linear.
2. Se a i-ésima letra do alfabeto é representada pelo número i e a função dispersão  $h(\text{chave}) = \text{chave} \bmod M$  é utilizada para  $M = 7$ , então mostre o resultado da inserção das seguintes chaves na tabela: P E S Q U I S A.

## Exercícios - Resolução de colisões por endereçamento aberto usando sondagem quadrática

1. Insira as chaves {10, 22, 31, 4, 15, 28, 59} em uma tabela T com 11 posições [0..10], utilizando a função hash:  $h(k) = k \bmod 11$  com sondagem quadrática, sendo  $c1=1$  e  $c2=3$ .
2. Se a  $i$ -ésima letra do alfabeto é representada pelo número  $i$  e a função dispersão  $h(\text{chave}) = \text{chave} \bmod M$  é utilizada para  $M = 7$ , então mostre o resultado da inserção das seguintes chaves na tabela: P E S Q U I S A.



# Como eliminar um elemento usando endereçamento aberto?

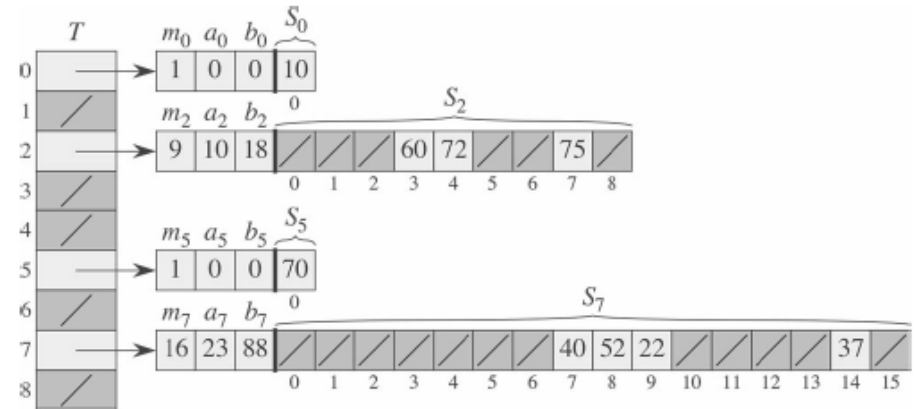
- O algoritmo de pesquisa (ou busca) percorre a mesma sequência de posições examinada pelo algoritmo de inserção quando a chave  $k$  foi inserida.
- Após a remoção, a posição não pode ser deixada como uma célula vazia, pois pode interferir nas buscas.
- A posição deve ser marcada de alguma maneira (com uma variável booleana, por exemplo) para que na busca possa-se saber que havia algo lá.

# O Hash perfeito

Ocorre quando forem exigidos  $O(1)$  acessos à memória para executar uma busca no pior caso.

## ❑ Dois níveis de **hash universal**

- Primeiro nível igual ao hashing com encadeamento
- Segundo nível usa uma tabela hashing secundário  $S_j$
- $m_j = n_j \wedge 2$
- Complexidade de espaço  $O(n)$



# Revisão (1/2)

- Tabelas de espalhamento implementam de forma eficiente dicionários de dados
- A função hash mapeia uma chave para uma espaço na tabela hash
- Boas funções hashes distribuem chaves de maneira a diminuir o número de colisões
- Uma das formas de resolver colisões é usar encadeamento

## Revisão (2/2)

- O fator de carga é uma propriedade que indica quanto a tabela corre o risco de sofrer colisão.
- O endereçamento aberto resolve colisões sem precisar de uma estrutura de dados auxiliar
- O endereçamento aberto utiliza o esquema de sondagem linear, quadrática ou hash duplo.
- O Hash perfeito utiliza duas tabelas hash com método hash universal