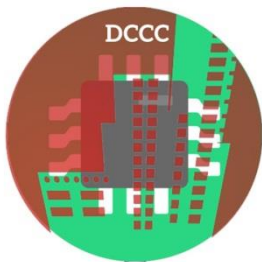


# *AEDII - Algoritmos e Estruturas de Dados II*

## Aula 02 – Hashing



Departamento de Computação  
de Construção Civil

**Prof. Aléssio Miranda Júnior**  
**[alessio@timoteo.cefetmg.br](mailto:alessio@timoteo.cefetmg.br)**  
**2Q-2019**



15/08/2019

## Sobre a busca de dados/chaves

## ***Busca em tabelas (vetores/arrays)***

Para se resolver os problemas de busca, inserção e remoção em uma tabela com  $n$  elementos, há várias maneiras:

- **Busca sequencial:**  
acesso em  $O(n)$

## ***Busca em tabelas (vetores/arrays)***

Para se resolver os problemas de busca, inserção e remoção em uma tabela com **n** elementos, há várias maneiras:

➤ **Busca sequencial:**

acesso em  $O(n)$

➤ **Busca Binária:**

acesso em  $O(\log_2(n))$

ainda lento se **n** é grande ( $\log_2(1000000) = 20$ )

# ***Busca em tabelas (vetores/arrays)***

Para se resolver os problemas de busca, inserção e remoção em uma tabela com  $n$  elementos, há várias maneiras:

➤ **Busca sequencial:**

acesso em  $O(n)$

➤ **Busca Binária:**

acesso em  $O(\log_2(n))$

ainda lento se  $n$  é grande ( $\log_2(1000000) = 20$ )

➤ **Uso de árvores balanceadas  
(estudaremos ainda...):**

➤ acesso em  $O(\log(n))$

➤ acesso bem melhor do que na busca sequencial!

➤ Em uma árvore B, o acesso é  $O(\log_k(n))$ , onde  $k$  é o tamanho da folha (veremos no final do curso)

## *Busca em tabelas (vetores/arrays)*

**Não é possível achar um método mais rápido?**

- $O(n)$  não é ruim em memória, mas mesmo um método com complexidade logarítmica é custoso.
- Acesso a disco é 1.000.000 vezes mais lento do que em memória.

# *Busca em tabelas (vetores/arrays)*

**Sim!**

## **Tabelas de dispersão (Hash tables)**

- Busca com acesso médio igual a  $O(1)$ .

# *Busca em tabelas (vetores/arrays)*

**Tabelas de  
dispersão**

=

**Tabelas de  
espalhamento**

=

**Hash  
tables**

- Busca com acesso médio igual a  $O(1)$ .
- Isso significa “constante” na média.  
No pior caso é  $O(n)$ .  
Na medida do possível uma constante pequena.
- São estruturas de dados eficientes para implementar um dicionário.



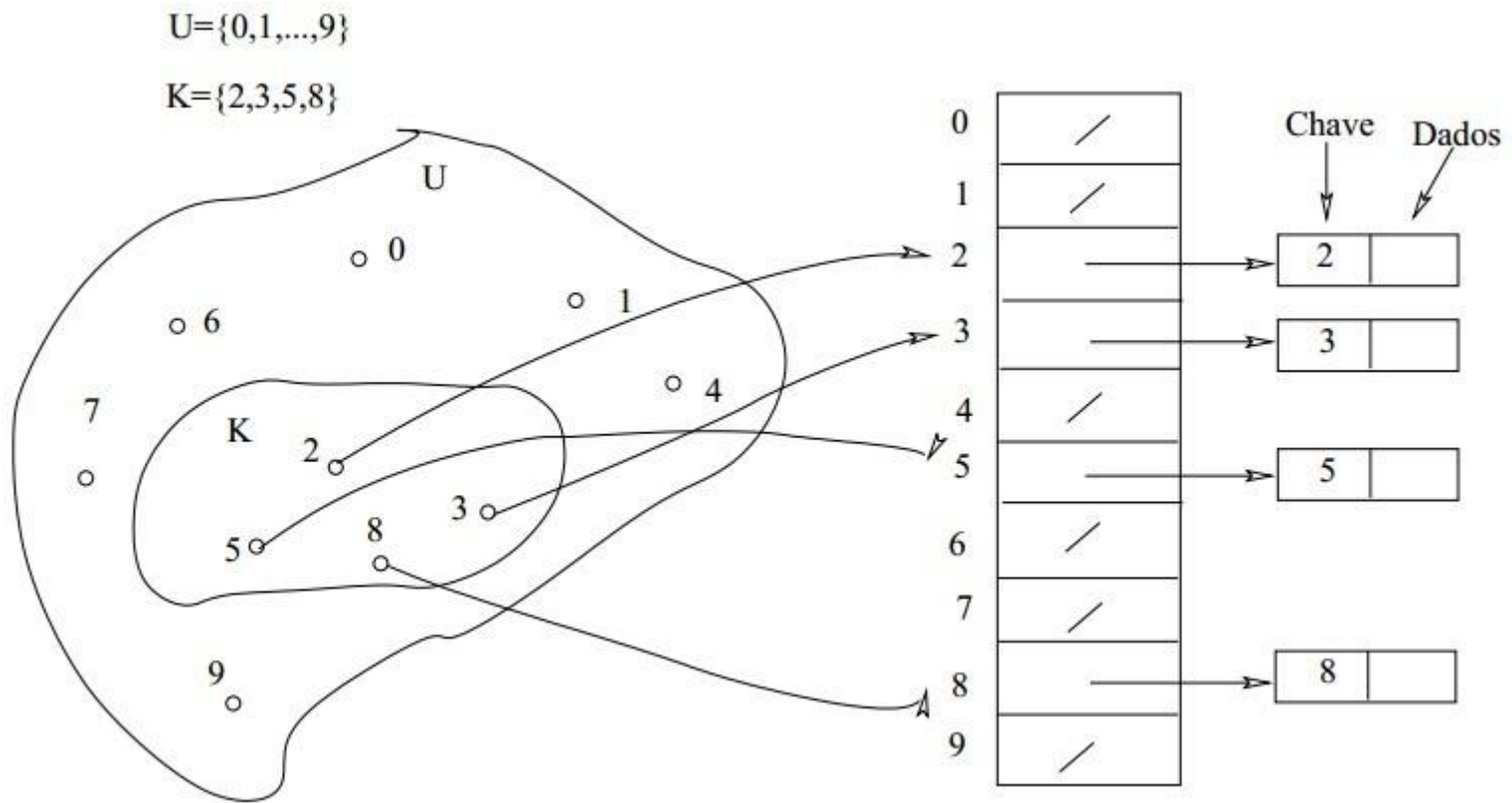
## **HASHING (tabelas de dispersão)**

## ***(1) Acesso/Endereçamento direto***

- Considere que os valores das chaves sejam: 0, 1, ...,  $m-1$ : Pode-se usar diretamente o valor de cada chave em seu índice de tabela (cada chave  $x$  armazenada no compartimento  $x$ )
- O endereçamento direto se baseia no fato de que podemos examinar uma posição qualquer em  $O(1)$ . Isto é aplicável quando podemos alocar uma tabela com uma posição para cada chave possível

# (1) Acesso/Endereçamento direto

Técnica simples que funciona quando o universo de chaves  $U$  é razoavelmente pequeno.



Haveriam  $|U| - |K|$  compartimentos vazios ou espaços.

# (1) Acesso/Endereçamento direto

## Implementação

### Search

*Direct\_address\_search*( $T, k$ )  
return  $T[k]$

### Insert

*Direct\_address\_insert*( $T, x$ )  
 $T[\text{key}[x]] \leftarrow x$

### Delete

*Direct\_address\_delete*( $T, x$ )  
 $T[\text{key}[x]] \leftarrow \text{NIL}$

# *(1) Acesso/Endereçamento direto*

## Considerações

- A dificuldade de usar endereçamento direto é óbvia: A alocação de uma tabela **T** de tamanho **|U|** pode ser inviável para um universo muito grande.
- Endereçamento direto pode implicar em desperdício de memória.
- O conjunto de chaves **K** armazenadas na tabela pode ser muito menor do que o universo de chaves **U**:
  - Espaço utilizado:  $O(|U|)$
  - Tempo de busca:  $O(1)$

## *(2) Tabelas de dispersão/espalhamento*

- Através da aplicação de uma função conveniente (**função hash**), a chave é transformada em um endereço de tabela (endereço base).

$$\text{hash(chave)} \rightarrow \{0, 1, \dots, m-1\}$$

- $h(k)$  é uma função  $h$  que mapeia o universo  $U$  de chaves para entradas da tabela de espalhamento  $T[0, \dots, m-1]$ .

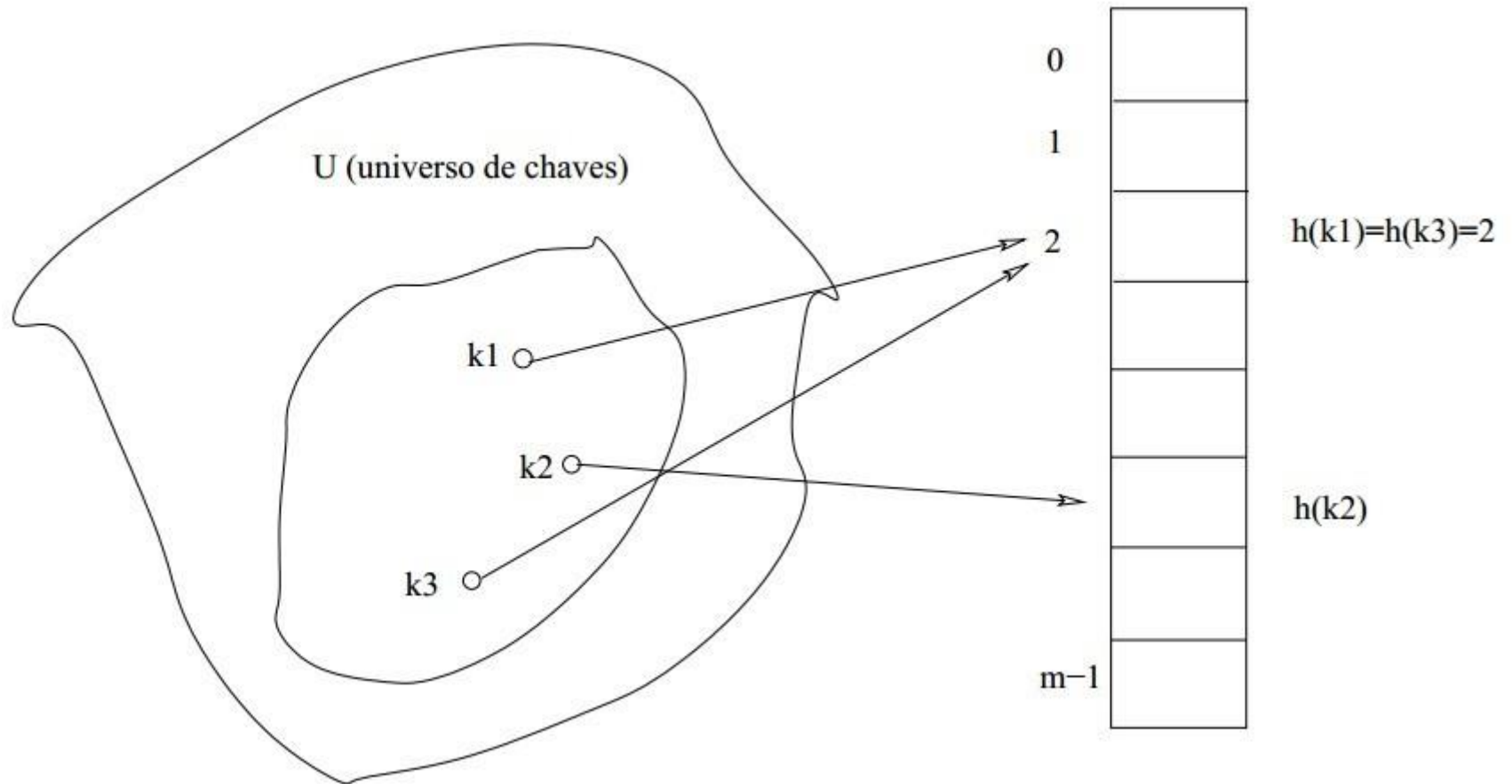
$$h : U \rightarrow \{0, 1, \dots, m - 1\}$$

## *(2) Tabelas de dispersão/espalhamento*

### Considerações

- Uma dificuldade dessa técnica é a possibilidade de duas chaves  $k_1$  e  $k_3$  serem mapeadas para a mesma posição na Tabela de Espalhamento.
- Uma **colisão entre 2 chaves** ocorre quando  $h(k_1)=h(k_3)$ .

## (2) Tabelas de dispersão/espalhamento





## *(2) Tabelas de dispersão/espalhamento*

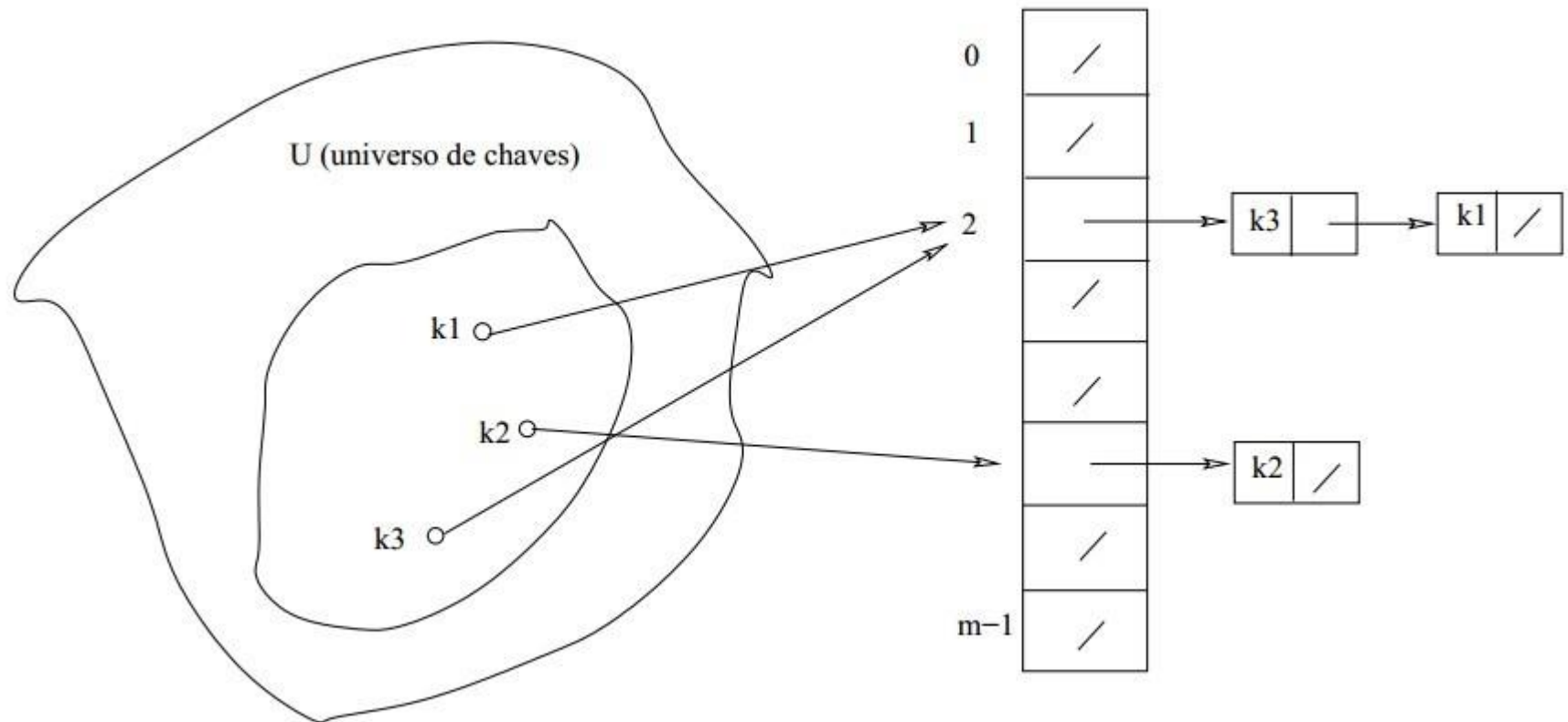
Duas formas de tratar colisões:

- Endereçamento aberto
- Resolução de colisões através de encadeamento

## (2) Tabelas de dispersão/espalhamento

Duas formas de tratar colisões:

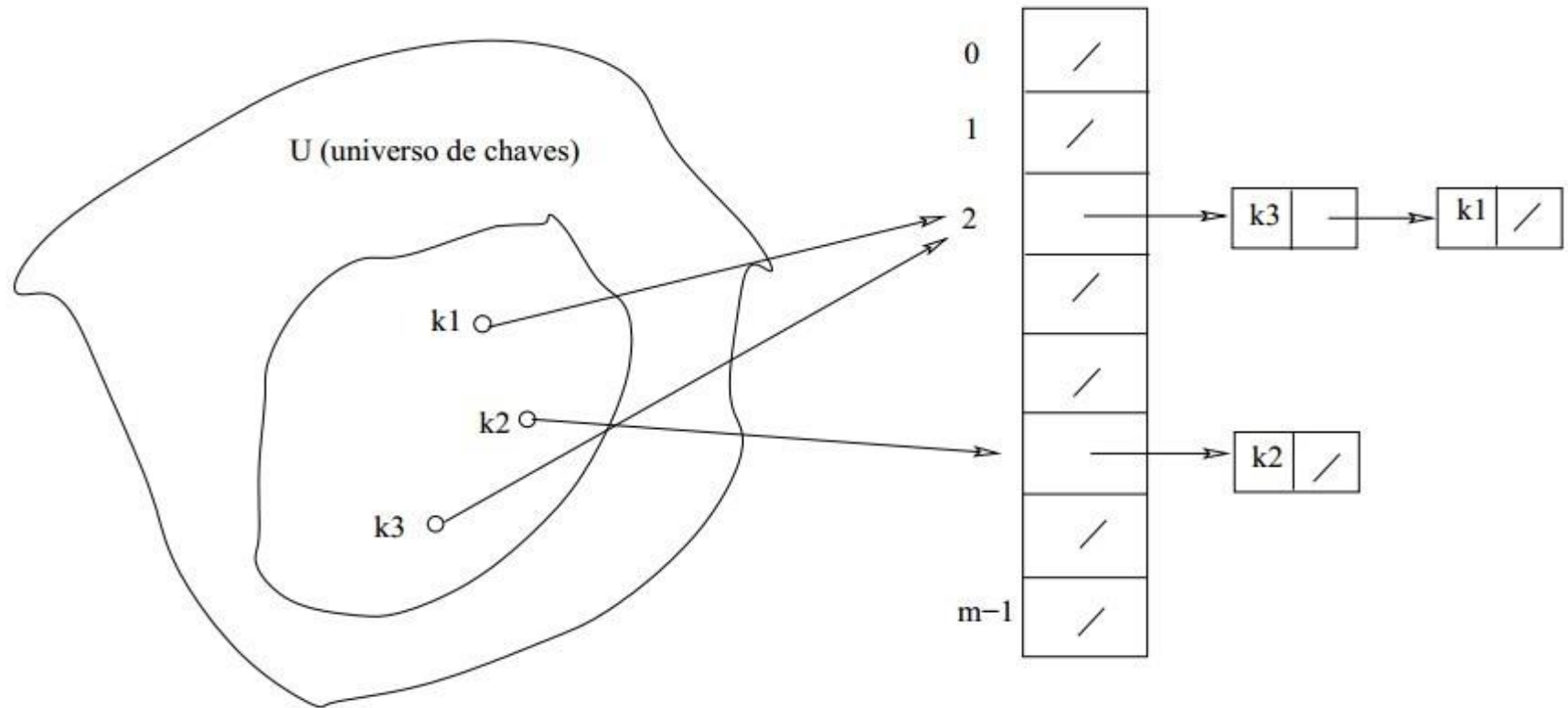
- Endereçamento aberto
- Resolução de colisões através de encadeamento



## (2) Tabelas de dispersão/espalhamento

Duas formas de tratar colisões:

- Endereçamento aberto
- Resolução de colisões através de encadeamento



Para evitar colisões usa-se uma função Hash que apresente “comportamento randômico”

# Tratamento de colisões porencadeamento

## Implementação

### Inserção

*Hash\_insert*( $T, x$ )

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

### Busca

*Hash\_search*( $T, k$ )

Procure por um elemento de chave  $k$   
na lista  $T[h(k)]$

# *Tratamento de colisões porencadeamento*

## Implementação

### Remoção

*Hash\_delete*( $T, x$ )

Delete  $x$  da lista  $T[h(key[x])]$

# *Tabelas de espalhamento*

- **Análise das operações**
- Inserção é executada em tempo  **$O(1)$** .
- Remoção de um elemento  **$x$**  é executada em tempo  **$O(1)$** .
- Busca leva tempo proporcional ao comprimento da lista.

# *Tabelas de espalhamento (TEs)*

## ➤ Considerações

- Seja **m** o número de entradas na TE.
  - Seja **n** o número de elementos armazenados na TE.
- Fator de carga  $\alpha$  é definido por  $\alpha = \frac{n}{m}$

# *Tabelas de espalhamento (TEs)*

## Considerações

- Seja **m** o número de entradas na TE.
- Seja **n** o número de elementos armazenados na TE
- Fator de carga  $\alpha$  definido por  $\alpha = \frac{n}{m}$

## Análise de espalhamento por encadeamento

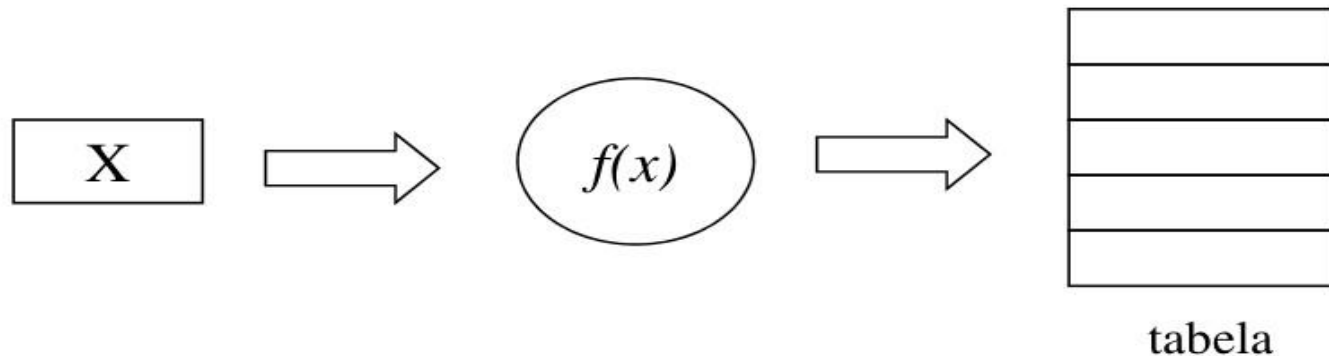
- No pior caso, o tempo de busca de uma chave **k** pode levar tempo  $\Theta(n)$  em uma TE.
- O tempo de busca em uma TE depende de quão bem a função de espalhamento **h distribui as chaves** entre as entradas de T.



# Tabelas de espalhamento (TEs)

## Espalhamento uniforme

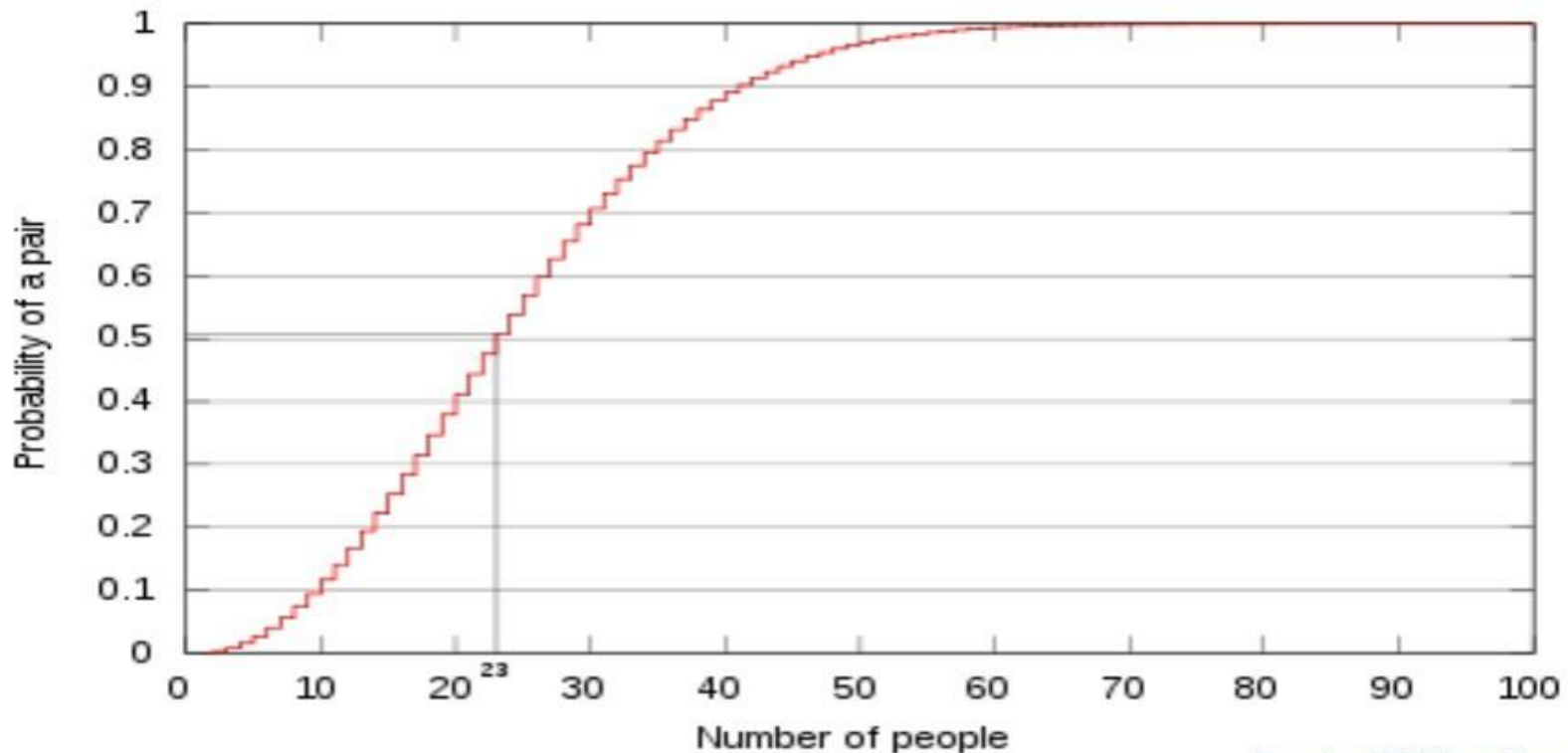
- Se a probabilidade de um elemento qualquer ser direcionado para uma entrada qualquer de  $T$  é uniforme, **independentemente dos demais elementos**, dizemos que a função de espalhamento  $h$  satisfaz a **condição de espalhamento uniforme**.



# Sobre o espalhamento uniforme

## Paradoxo do aniversário

Em um grupo de 23 pessoas ou mais pessoas, juntas ao acaso, existe uma chance maior do que 50% de que duas pessoas comemorem aniversário no mesmo dia.



Fonte: Wikipedia

Se for utilizar uma função Hash que enderece 23 chaves (randômicas) em uma tabela de 365 elementos, a **probabilidade de que haja colisão é maior do que 50%**

15/08/2019

## ***Função de espalhamento perfeito***

### ➤ **Função de Espalhamento Perfeito:**

Quando a função é capaz de evitar qualquer colisão (supondo-se determinados **n** e **m**).

## ***Função de espalhamento perfeito***

### ➤ **Função de Espalhamento Perfeito:**

Quando a função é capaz de evitar qualquer colisão (supondo-se determinados  $n$  e  $m$ ).

➤ Entretanto, é muito difícil se obter uma função de dispersão perfeita.

## *Função de espalhamento perfeito*

Suponha que você quisesse guardar:

- **n = 4000** registros, em
- **m = 5000** compartimentos.

Foi mostrado que todas as funções de espalhamento possíveis, somente **1** em  **$10^{120000}$**  é capaz de evitar todas as colisões.

## *Função de espalhamento perfeito*

Suponha que você quisesse guardar:

- **n = 4000** registros, em
- **m = 5000** compartimentos.

Foi mostrado que todas as funções de espalhamento possíveis, somente **1** em  **$10^{120000}$**  é capaz de evitar todas as colisões.

Para um número pequeno de chaves que mudam com pouca frequência é possível achar com mais facilidade uma função de espalhamento perfeita.

# *Função de espalhamento perfeito*

Suponha que você quisesse guardar:

- **n = 4000** registros, em
- **m = 5000** compartimentos.

Foi mostrado que todas as funções de espalhamento possíveis, somente **1** em  **$10^{120000}$**  é capaz de evitar todas as colisões.

Para um número pequeno de chaves que mudam com pouca frequência é possível achar com mais facilidade uma função de espalhamento perfeita.

**Foco: Minimizar o número de colisões a um valor aceitável**

15/08/2019

**Funções Hash (de espalhamento):**  
→ **O método da divisão**



## *Método da divisão*

- Fácil, eficiente e largamente empregado.
- **A chave  $k$  é dividida pela dimensão da tabela  $m$ :**  
o resto da divisão é o endereço base:
  - $h(k) = k \bmod m$

Resulta em endereços no intervalo:  $[0, m-1]$ .

## *Método da divisão*

➤ Por exemplo, para  $m=12$ , e  $k=100$ :

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

$$h(100) = 100 \bmod 12 = 4$$

## ***Método da divisão***

Bons valores para  $m$  são números primos não muito próximos de potências de 2.

# *Método da divisão*

Bons valores para  $m$  são números primos

- **O motivo não é óbvio!**
- Ver os seguintes links para uma discussão a respeito:
  - <http://stackoverflow.com/questions/1145217/why-should-hash-functions-use-a-prime-number-modulus>
  - <https://computinglife.wordpress.com/2008/11/20/why-do-hash-functions-use-prime-numbers>

## *Método da divisão*

Suponha que desejamos alocar  $n=2000$  cadeias de 8 bits, e que não nos importamos em procurar em listas de tamanho médio 3.

**Qual seria o tamanho apropriado para a tabela T?**

## *Método da divisão*

Suponha que desejamos alocar  $n=2000$  cadeias de 8 bits, e que não nos importamos em procurar em listas de tamanho médio 3.

Qual seria o tamanho apropriado para a tabela T?

Então, podemos fazer  $m=701$ , pois este é um número primo próximo de  $2000/3$ , então

$$\alpha = \frac{n}{m} = \frac{2000}{701} \approx 3$$

# Método da divisão

$n=2000, m=701$

$$h(k) = k \bmod 701$$

|     |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 2   | 3   | 5   | 7   | 11  | 13  | 17  | 19  | 23  | 29  |
| 31  | 37  | 41  | 43  | 47  | 53  | 59  | 61  | 67  | 71  |
| 73  | 79  | 83  | 89  | 97  | 101 | 103 | 107 | 109 | 113 |
| 127 | 131 | 137 | 139 | 149 | 151 | 157 | 163 | 167 | 173 |
| 179 | 181 | 191 | 193 | 197 | 199 | 211 | 223 | 227 | 229 |
| 233 | 239 | 241 | 251 | 257 | 263 | 269 | 271 | 277 | 281 |
| 283 | 293 | 307 | 311 | 313 | 317 | 331 | 337 | 347 | 349 |
| 353 | 359 | 367 | 373 | 379 | 383 | 389 | 397 | 401 | 409 |
| 419 | 421 | 431 | 433 | 439 | 443 | 449 | 457 | 461 | 463 |
| 467 | 479 | 487 | 491 | 499 | 503 | 509 | 521 | 523 | 541 |
| 547 | 557 | 563 | 569 | 571 | 577 | 587 | 593 | 599 | 601 |
| 607 | 613 | 617 | 619 | 631 | 641 | 643 | 647 | 653 | 659 |
| 661 | 673 | 677 | 683 | 691 | 701 | 709 | 719 | 727 | 733 |
| 739 | 743 | 751 | 757 | 761 | 769 | 773 | 787 | 797 | 809 |
| 811 | 821 | 823 | 827 | 829 | 839 | 853 | 857 | 859 | 863 |
| 877 | 881 | 883 | 887 | 907 | 911 | 919 | 929 | 937 | 941 |
| 947 | 953 | 967 | 971 | 977 | 983 | 991 | 997 |     |     |

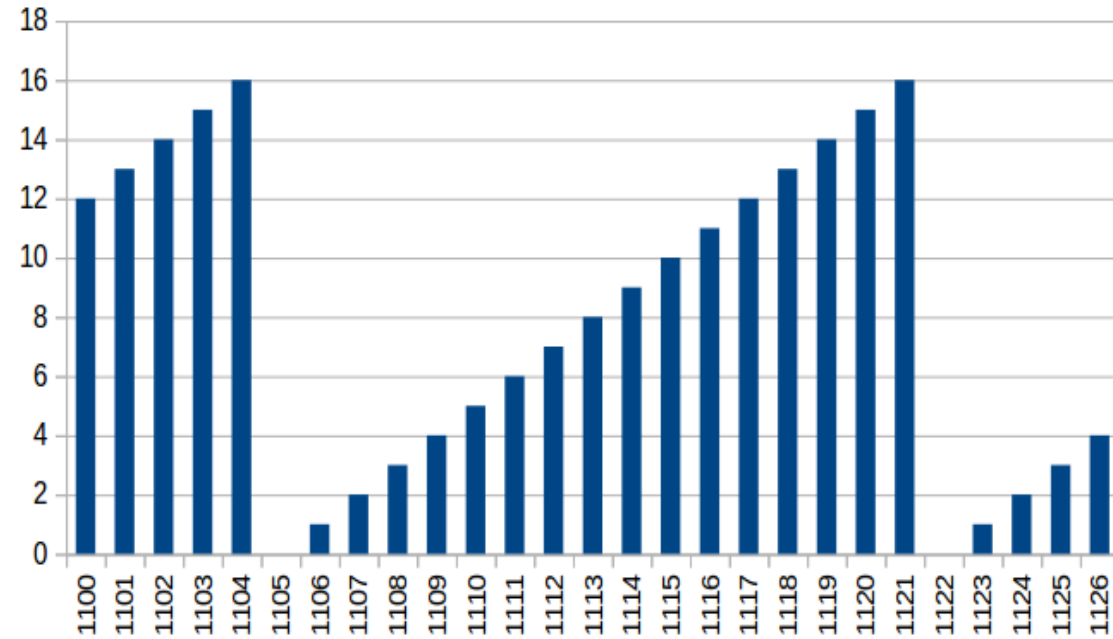
Tabela de primos

# Método da divisão

m=

17

| k    | h(k) |
|------|------|
| 1100 | 12   |
| 1101 | 13   |
| 1102 | 14   |
| 1103 | 15   |
| 1104 | 16   |
| 1105 | 0    |
| 1106 | 1    |
| 1107 | 2    |
| 1108 | 3    |
| 1109 | 4    |
| 1110 | 5    |
| 1111 | 6    |
| 1112 | 7    |
| 1113 | 8    |
| 1114 | 9    |
| 1115 | 10   |
| 1116 | 11   |
| 1117 | 12   |
| 1118 | 13   |
| 1119 | 14   |
| 1120 | 15   |
| 1121 | 16   |
| 1122 | 0    |
| 1123 | 1    |
| 1124 | 2    |
| 1125 | 3    |
| 1126 | 4    |



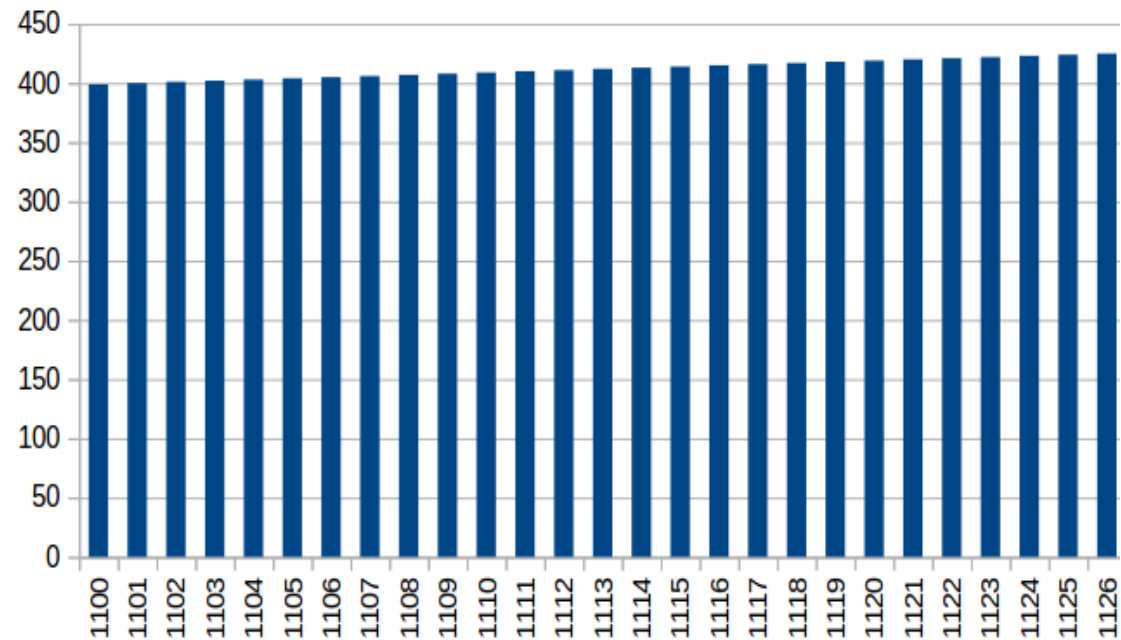


# Método da divisão

m=

701

| k    | h(k) |
|------|------|
| 1100 | 399  |
| 1101 | 400  |
| 1102 | 401  |
| 1103 | 402  |
| 1104 | 403  |
| 1105 | 404  |
| 1106 | 405  |
| 1107 | 406  |
| 1108 | 407  |
| 1109 | 408  |
| 1110 | 409  |
| 1111 | 410  |
| 1112 | 411  |
| 1113 | 412  |
| 1114 | 413  |
| 1115 | 414  |
| 1116 | 415  |
| 1117 | 416  |
| 1118 | 417  |
| 1119 | 418  |
| 1120 | 419  |
| 1121 | 420  |
| 1122 | 421  |
| 1123 | 422  |
| 1124 | 423  |
| 1125 | 424  |
| 1126 | 425  |



# Método da divisão

Se o elemento for uma string?

```
/******  
função de espalhamento exemplo  
devolve um inteiro entre 0 e 50  
*****/  
int f(char *s){  
    int k = strlen(s);  
    int i, j = 0;  
    for(i=0; i<k; i++) j+=s[i];  
    return j % 51;  
}
```

## Funções Hash (de espalhamento):

→ **Método da multiplicação**

## ***Método da multiplicação***

O método utiliza uma constante **A** ( $0 < A < 1$ ), sendo **h(k)** calculado como:

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

## ***Método da multiplicação***

O método utiliza uma constante **A** ( $0 < A < 1$ ), sendo **h(k)** calculado como:

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

$$(kA \bmod 1) = kA - \lfloor kA \rfloor$$

## ***Método da multiplicação***

O método utiliza uma constante **A** ( $0 < A < 1$ ), sendo **h(k)** calculado como:

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

$$(kA \bmod 1) = kA - \lfloor kA \rfloor$$

A vantagem deste método é que o valor de **m** não é crítico como no método de divisão.

- Mas a escolha de uma constante A adequada é crítica

# *Método da multiplicação*

$$m = 1000$$
$$A = 0,5$$

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

Determine os resultados de  $h(k)$  para:  
 $k = \{1100, 1101, 1102, 1103, 1104, 1105\}$

# *Método da multiplicação*

$m = 1000$   
 $A = 0,5$

$$h(1100) = 0$$

$$h(1101) = 500$$

$$h(1102) = 0$$

$$h(1103) = 500$$

$$h(1104) = 0$$

$$h(1105) = 500$$

$$h(1106) = 0$$

$$h(1107) = 500$$

.

.

.

.

.

.

.



# Método da multiplicação

$m = 1000$

$A = 0,5$

$h(1100) = 0$

$h(1101) = 500$

$h(1102) = 0$

$h(1103) = 500$

$h(1104) = 0$

$h(1105) = 500$

$h(1106) = 0$

$h(1107) = 500$

.

.

.

.

AAAAAAARRRRRRRRRR.GGGGGGGGGGHHHHHHH!!!!!!!!!!!!

# Método da multiplicação

➤ Alguns valores de **A** são melhores do que outros, em particular a razão áurea

$$m = 1000$$

$$A = 0,6180339887...$$

$$h(1100) = 837$$

$$h(1101) = 455$$

$$h(1102) = 73$$

$$h(1103) = 691$$

$$h(1104) = 309$$

$$h(1105) = 927$$

$$h(1106) = 545$$

$$h(1107) = 163$$

.

.

.

.

$$A \approx (\sqrt{5} - 1)/2 = 0,6180339887$$

.

.

**Funções Hash:**  
→ **Método da dobra**

# Método da dobra

**Quebre a chave em partes e combine-as de algum modo**

## **Exemplo:**

Se as chaves são números de 8 dígitos, e a tabela hash tem 1000 entradas, quebre a chave em 3 números:

- Número 1: 3 dígitos
- Número 2: 3 dígitos
- Número 3: 2 dígitos

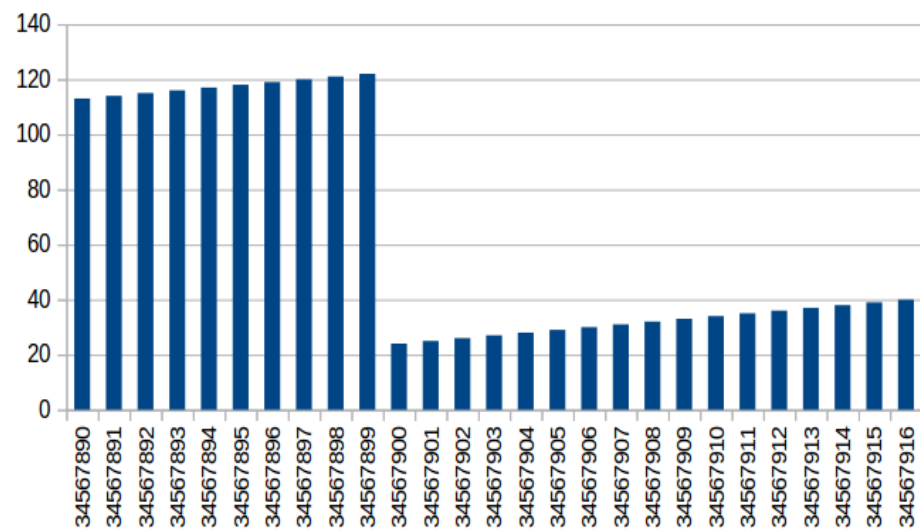
Some-os, considerando os 3 últimos dígitos da soma para compor a chave.

$$73419883 \rightarrow 734 + 198 + 83 = 1015 \rightarrow 015 \rightarrow h(k) = 15$$

# Método da dobra

n= 1000

| k        | d1  | d2  | d3 | soma | h(k) |
|----------|-----|-----|----|------|------|
| 34567890 | 345 | 678 | 90 | 1113 | 113  |
| 34567891 | 345 | 678 | 91 | 1114 | 114  |
| 34567892 | 345 | 678 | 92 | 1115 | 115  |
| 34567893 | 345 | 678 | 93 | 1116 | 116  |
| 34567894 | 345 | 678 | 94 | 1117 | 117  |
| 34567895 | 345 | 678 | 95 | 1118 | 118  |
| 34567896 | 345 | 678 | 96 | 1119 | 119  |
| 34567897 | 345 | 678 | 97 | 1120 | 120  |
| 34567898 | 345 | 678 | 98 | 1121 | 121  |
| 34567899 | 345 | 678 | 99 | 1122 | 122  |
| 34567900 | 345 | 679 | 00 | 1024 | 24   |
| 34567901 | 345 | 679 | 01 | 1025 | 25   |
| 34567902 | 345 | 679 | 02 | 1026 | 26   |
| 34567903 | 345 | 679 | 03 | 1027 | 27   |
| 34567904 | 345 | 679 | 04 | 1028 | 28   |
| 34567905 | 345 | 679 | 05 | 1029 | 29   |
| 34567906 | 345 | 679 | 06 | 1030 | 30   |
| 34567907 | 345 | 679 | 07 | 1031 | 31   |
| 34567908 | 345 | 679 | 08 | 1032 | 32   |
| 34567909 | 345 | 679 | 09 | 1033 | 33   |
| 34567910 | 345 | 679 | 10 | 1034 | 34   |
| 34567911 | 345 | 679 | 11 | 1035 | 35   |
| 34567912 | 345 | 679 | 12 | 1036 | 36   |
| 34567913 | 345 | 679 | 13 | 1037 | 37   |
| 34567914 | 345 | 679 | 14 | 1038 | 38   |
| 34567915 | 345 | 679 | 15 | 1039 | 39   |
| 34567916 | 345 | 679 | 16 | 1040 | 40   |



## *Chaves como números naturais*

- Até aqui consideramos o caso da chave ser um número natural.

$k = \text{'gato'}$

$h(k) = ?$

## *Chaves como números naturais*

**Se não for este o caso, ela deve ser convertida primeiro para um número natural.**

Por exemplo, se a chave é uma cadeia de caracteres podemos interpreta-la como um natural em uma base conveniente (26 se levarmos em consideração apenas letras minúsculas):

$$gato = 7 \cdot 26^3 + 1 \cdot 26^2 + 20 \cdot 26^1 + 15 = 124243.$$

## *Chaves como números naturais*

**Se não for este o caso, ela deve ser convertida primeiro para um número natural.**

Por exemplo, se a chave é uma cadeia de caracteres podemos interpreta-la como um natural em uma base conveniente (26 se levarmos em consideração apenas letras maiúsculas):

$$gato = 7 \cdot 26^3 + 1 \cdot 26^2 + 20 \cdot 26^1 + 15 = 124243.$$

Palavras mais longas representam um desafio maior, pois o valor calculado pode ser muito grande (e o cálculo pode levar muito tempo).

Neste caso, podemos escolher algumas letras para representar a palavra, como a primeira, a do meio e a penúltima



**Para finalizar...**

## *Para finalizar*

### **Vantagens:**

- Algoritmo simples e eficiente para inserção, remoção e busca.

### **Desvantagens:**

- Nenhuma garantia 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.

### **Desafio:**

- Pense na criação de uma função hash “universal”