

# Tabela de Espalhamento (Tabela de Hash ou *Hash Table*)

Prof. Martín Vigil

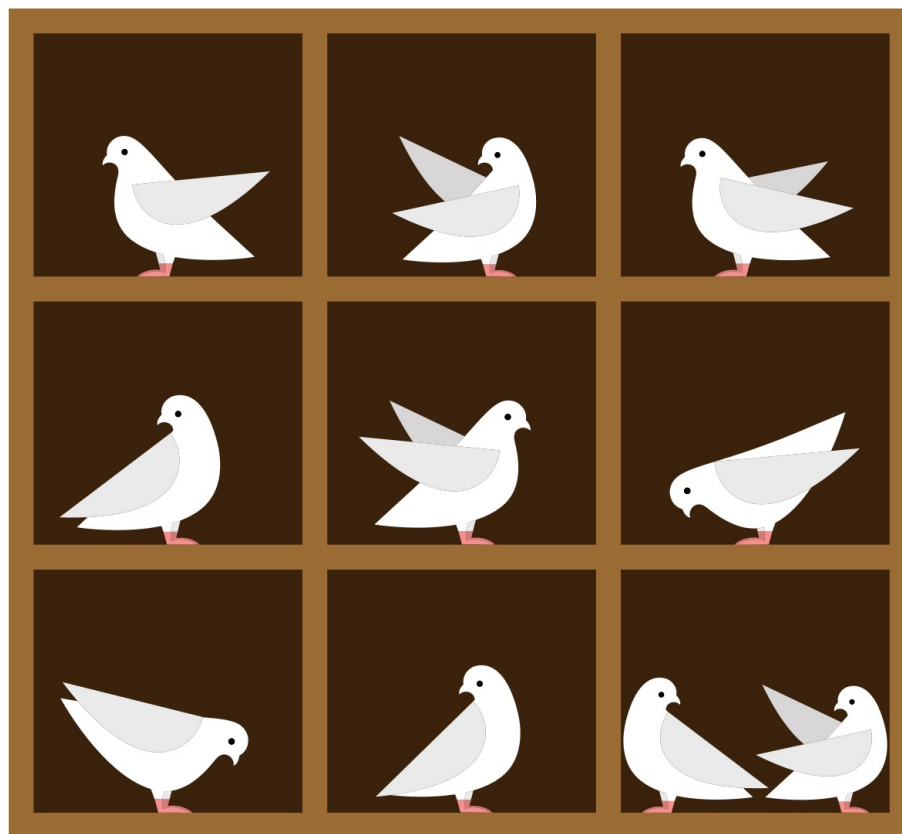
Adaptado de Ednaldo Pizzolato e  
Katia Guimarães

# **CONCEITOS PRELIMINARES**

# Tabela ASCII

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
32	20	40	[space]	64	40	100	@	96	60	140	`
33	21	41	!	65	41	101	A	97	61	141	a
34	22	42	"	66	42	102	B	98	62	142	b
35	23	43	#	67	43	103	C	99	63	143	c
36	24	44	\$	68	44	104	D	100	64	144	d
37	25	45	%	69	45	105	E	101	65	145	e
38	26	46	&	70	46	106	F	102	66	146	f
39	27	47	'	71	47	107	G	103	67	147	g
40	28	50	(	72	48	110	H	104	68	150	h
41	29	51	)	73	49	111	I	105	69	151	i
42	2A	52	*	74	4A	112	J	106	6A	152	j
43	2B	53	+	75	4B	113	K	107	6B	153	k
44	2C	54	,	76	4C	114	L	108	6C	154	l
45	2D	55	-	77	4D	115	M	109	6D	155	m
46	2E	56	.	78	4E	116	N	110	6E	156	n
47	2F	57	/	79	4F	117	O	111	6F	157	o
48	30	60	0	80	50	120	P	112	70	160	p
49	31	61	1	81	51	121	Q	113	71	161	q
50	32	62	2	82	52	122	R	114	72	162	r
51	33	63	3	83	53	123	S	115	73	163	s
52	34	64	4	84	54	124	T	116	74	164	t
53	35	65	5	85	55	125	U	117	75	165	u
54	36	66	6	86	56	126	V	118	76	166	v
55	37	67	7	87	57	127	W	119	77	167	w
56	38	70	8	88	58	130	X	120	78	170	x
57	39	71	9	89	59	131	Y	121	79	171	y
58	3A	72	:	90	5A	132	Z	122	7A	172	z
59	3B	73	;	91	5B	133	[	123	7B	173	{
60	3C	74	<	92	5C	134	\	124	7C	174	
61	3D	75	=	93	5D	135	]	125	7D	175	}
62	3E	76	>	94	5E	136	^	126	7E	176	~
63	3F	77	?	95	5F	137	_	127	7F	177	

# Princípio da Casa de Pombos



Colisão

# Princípio da Casa de Pombos

The image is a screenshot of the Facebook Events page. On the left sidebar, the 'Events' section is active, with a sub-menu containing 'Events', 'Calendar', 'Birthday', 'Discover', and 'Hosting'. The 'Birthday' option is highlighted with a mouse cursor. The main content area is divided into several sections. The top section is 'Today's Birthdays' for May 21, 2019. It lists two birthdays: one for a 22-year-old and another for a 35-year-old. A red bracket is drawn between these two entries, with the word 'Colisão' (Collision) written in red text next to it. Below this is the 'Recent Birthdays' section for Monday, May 20, 2019, showing a birthday for a 37-year-old. The bottom section is 'Upcoming Birthdays'. On the right side of the page, there is a 'RECENT BIRTHDAYS' section showing birthdays for 'Yesterday, May 20' and 'Saturday, May 18'. Below that is a 'Find Events' section with four buttons: 'Today', 'Tomorrow', 'This Week', and 'Choose Date'. At the bottom right, there are links for 'Causes' and 'Music'.

Search

Home Create

Events

Events

Calendar

Birthday

Discover

Hosting

+ Create Event

Today's Birthdays

May 21, 2019

22 years old

35 years old

Colisão

RECENT BIRTHDAYS

Yesterday, May 20

Saturday, May 18

Find Events

Today

Tomorrow

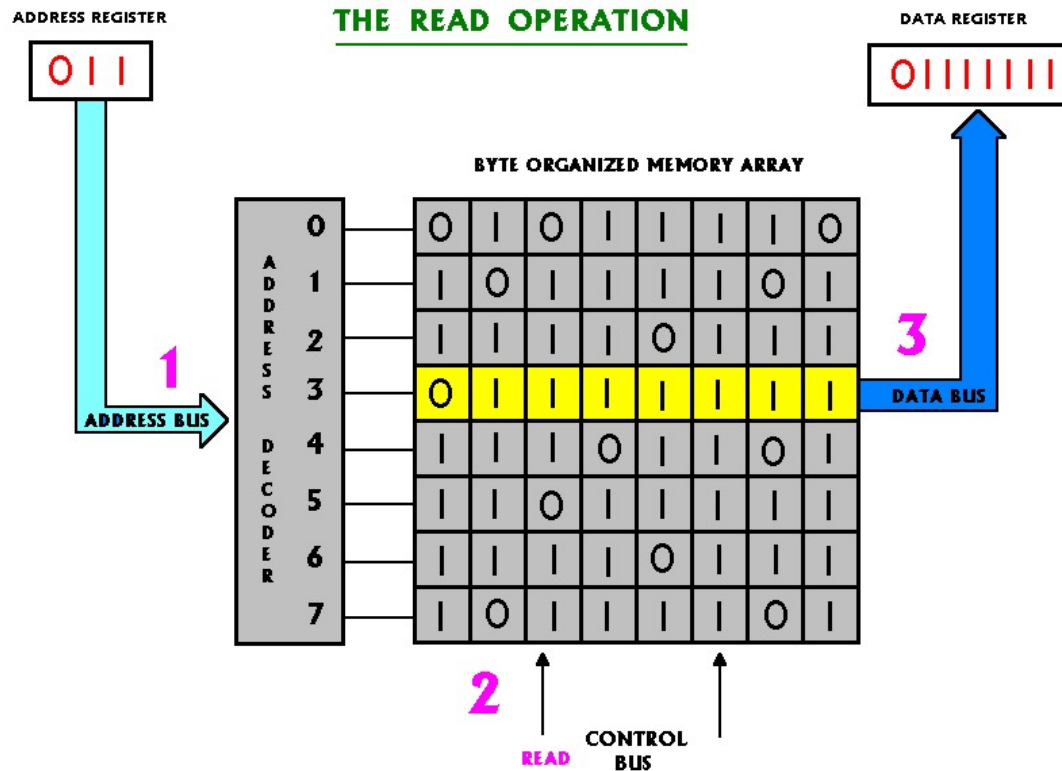
This Week

Choose Date

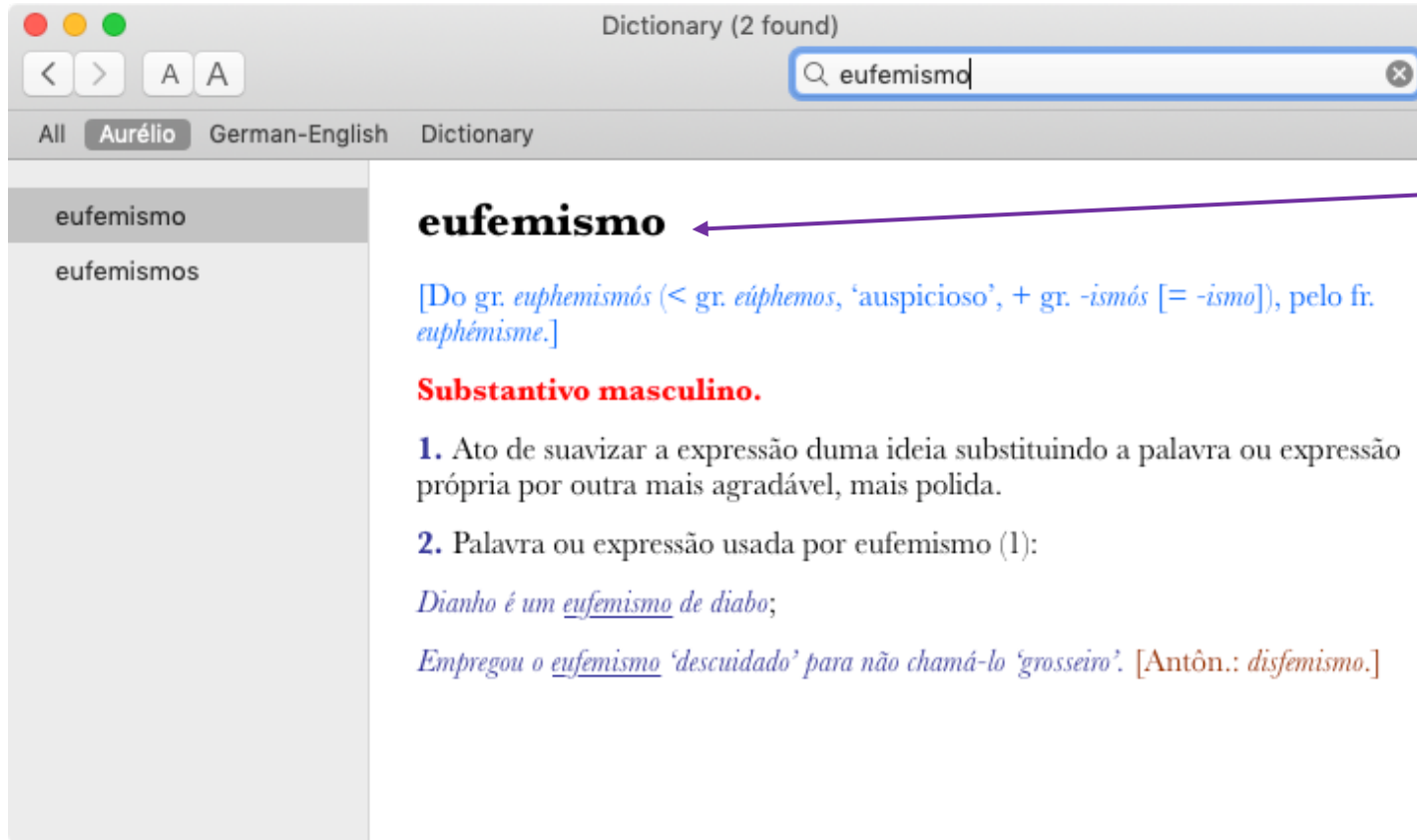
Causes

Music

# Acesso memória RAM custa $O(1)$



# Dicionário



Palavra (chave)

Significado  
(valor)

# Dicionário

- Conjunto de pares (chave, valor) que permite
  - busca pela chave
  - inserção e remoção de pares
- Exemplo: { (1, “José”), (2, “Maria”), ... }
- Exemplo: JSON

```
{  
  "symbol": "LTCBTC",  
  "bidPrice": "4.00000000",  
  "bidQty": "431.00000000",  
  "askPrice": "4.00000200",  
  "askQty": "9.00000000"  
}
```



# **TABELAS DE ENDEREÇO DIRETO**

# Dicionário na forma de array de valores onde índice é chave numérica

[0]	
[1]	"José"
[2]	"Maria"
...	...
[n-1]	"Sascha"

# Análise da Tabela de Endereço Direto

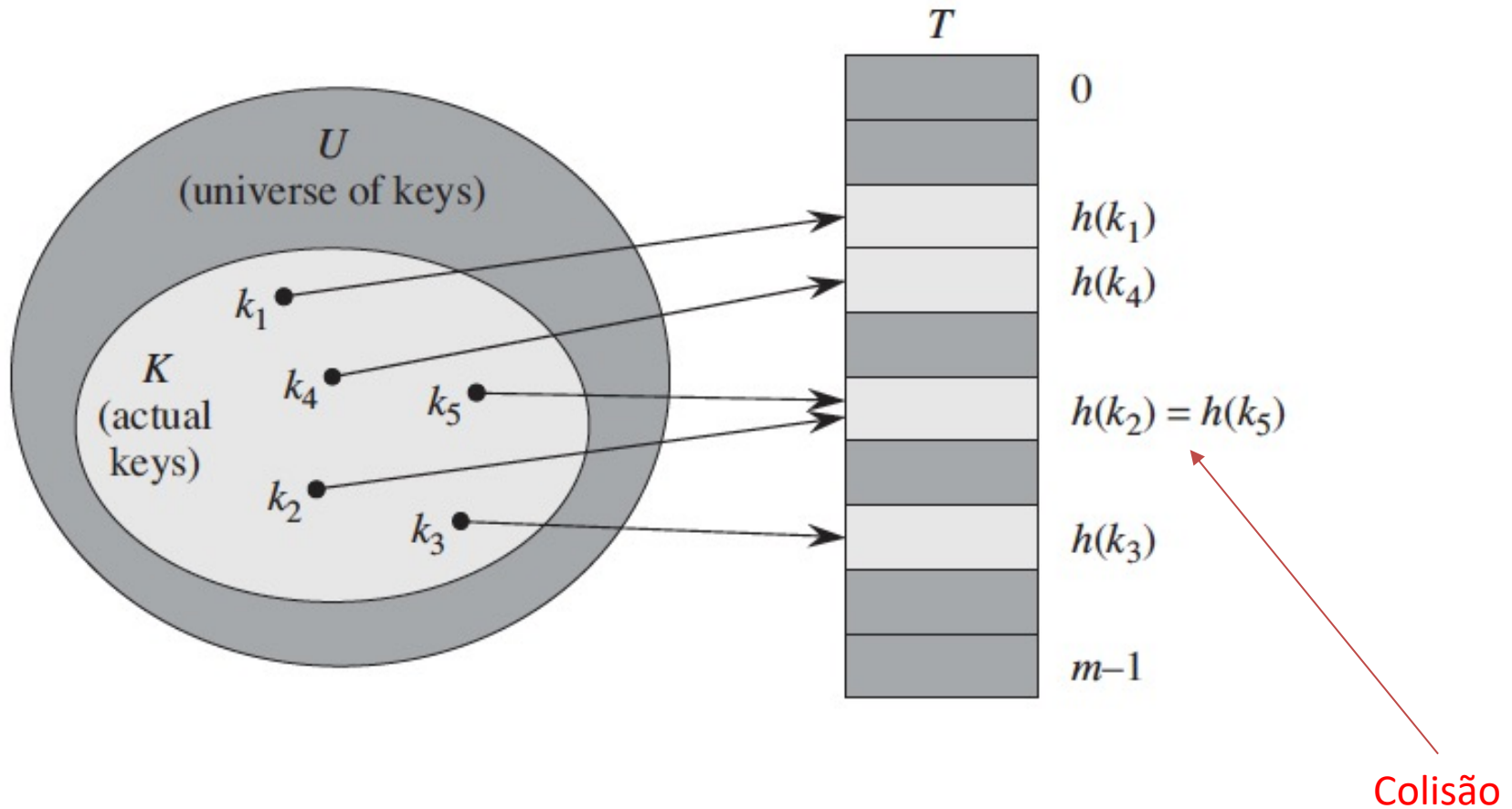
- Acesso  $O(1)$
- Dependendo do universo de chaves
  - A memória RAM pode ser **insuficiente**
  - Índices do array pode ser **inutilizados**
- Exemplo: Cadastro de Pessoa Física (CPF)
  - Formato AAA.BBB.CCC-DD =>  $10^{11}$  índices
  - Nem todo índice é CPF válido: ex: 123.456.789-10

# **TABELA DE HASH**

# Tabela T de hash

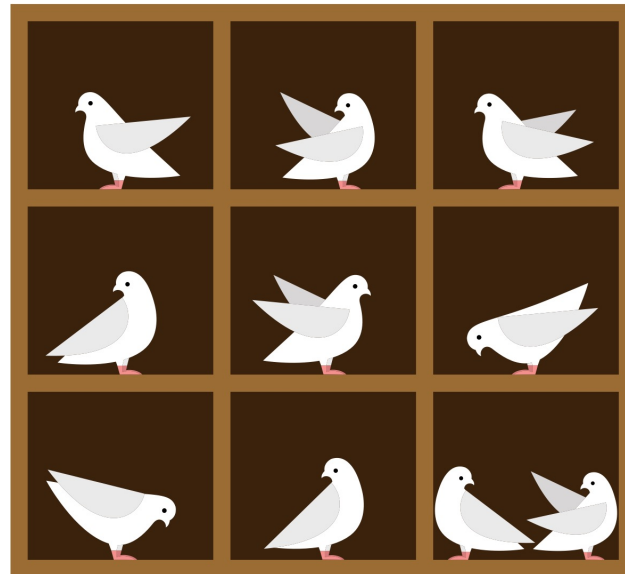
- Conjunto U de chaves  $\{k_0, k_1, \dots, k_{n-1}\}$
- Tabela T com índices  $\{0, 1, \dots, m-1\}$ ,  $m < n$
- Função ***h*** :  $U \rightarrow \{0, 1, \dots, m-1\}$  de *hash* **mapeia** uma chave  $k \in U$  para um índice  $h(k) \in T$

# Tabela T de hash



# Colisões em tabelas de hash

- Se há mais chaves inseridas que índices em T, haverá colisão

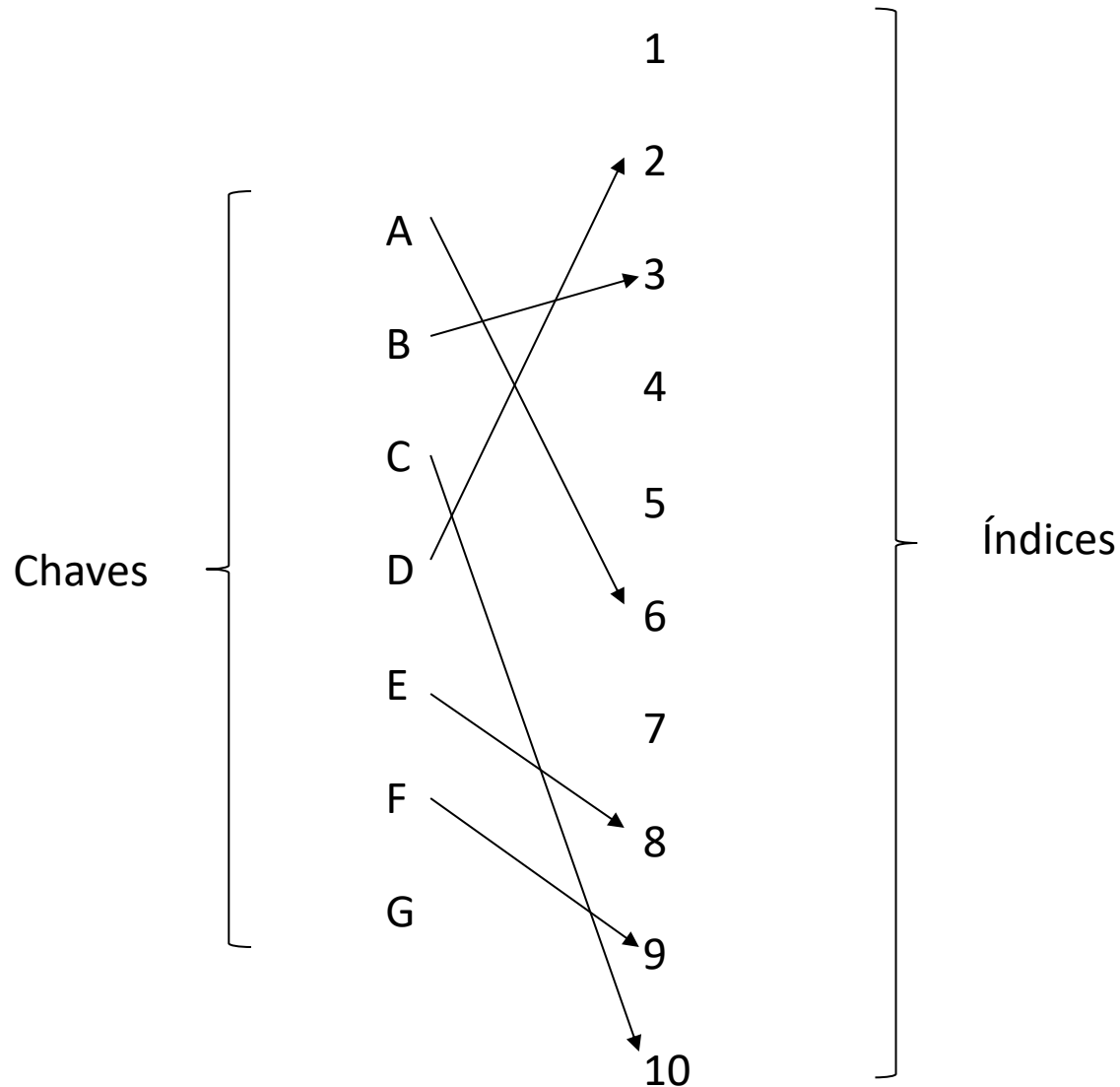


Colisão

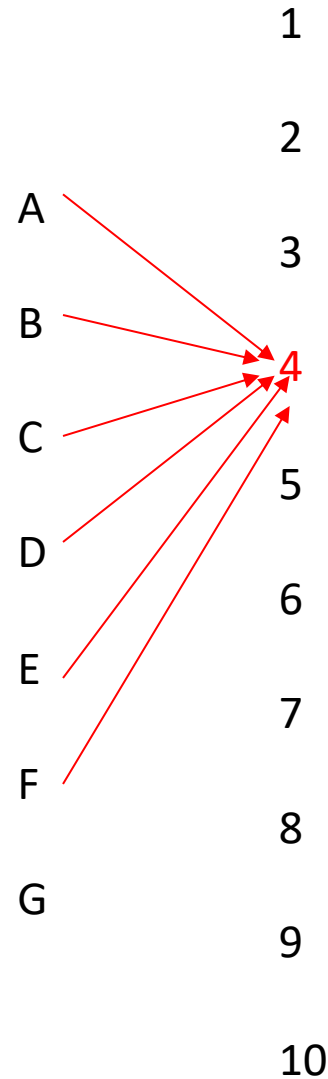
# **FUNÇÕES DE HASH**



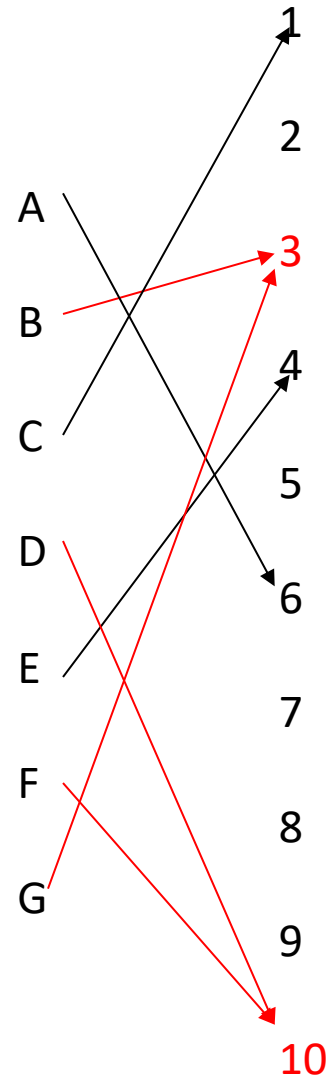
# Função ótima: *nenhuma* colisão



# Função péssima: *muitas* colisões



# Função boa: *algumas* colisões



# Características de uma boa função

- Distribui chaves *uniformemente* entre índices
- Ajustada de acordo com a distrib. das chaves

# Traduzindo chaves não inteiras

- Chaves podem ter a forma: “Maria”, “pt\_0”
- Cada character é um número da Tabela ASCII
- Geralmente traduzi-las para  $l=\{...,-1,0,1,...\}$
- Exemplo: traduzir `std::string` para `std::size_t`
  - Seja string  $s$  um array de char  $s[0], s[1], ..., s[m-1]$
  - Tradução:  $s[0]*31^{(m-1)} + s[1]*31^{(m-2)} + ... + s[m-1]$
  - Ex: “hello world”  $\Rightarrow$  88006926820958896
  - Ex: “dec0006”  $\Rightarrow$  91734818568
  - Ex: “UFSC”  $\Rightarrow$  2602145

# Função módulo (resto da divisão)

- $h(k) = k \bmod m$ , onde  $m$  é
  - o número de índices na tabela  $T$  de hash
  - *preferencialmente* **primo**

# Função módulo ( $m=7$ )

- Conjunto  $U$  de chaves *uniformemente* distrib.
- $U=\{0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, \dots\}$

Índice	0	1	2	3	4	5	6	7
Chave	0	1	2	3	4	5	6	
	7	8	9	10	11	12	13	
	14	...						

Colisões

# Função módulo (primo vs **composto**)

- Conjunto U de chaves **não** *uniformemente* distrib.
- $U=\{12, 15, 18, 24, 27, 28, 30, 36, 39, 42\}$

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12
Chave	12			15			18						
	24			27	28		30						
	36			39			42						

m=12 (**composto**)  
9 colisões

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chave	39	27	15	42	30	18				24	36		12	
			28											

m=13 (**primo**)  
2 colisões



# Função módulo (primo vs composto)

- Chaves que têm um fator em comum com o número  $m$  de índices serão mapeadas para um índice que é múltiplo desse fator.

Chave	Fatores da chave	Fatores de 12	Fatores em comum	Chave % 12
15	5,3	2,2,3	3	3
27	3,3,3		3	3
28	2,2,7		2,2	4
39	3,13		3	3
18	2,3,3		2,3	6
30	5,2,3		2,3	6
42	2,2,2,2,3		2,3	6

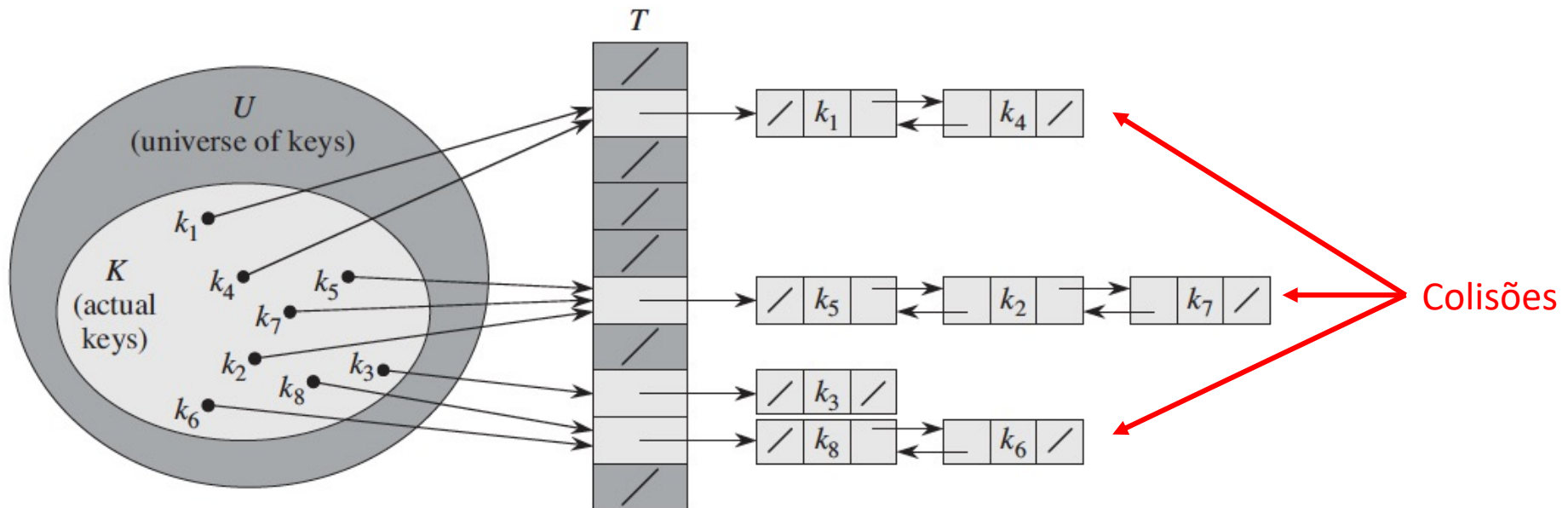
# Função módulo (primo vs composto)

- Chaves que têm um fator em comum com o número  $m$  de índices serão mapeadas para um índice que é múltiplo desse fator.

Chave	Fatores da chave	Fatores de 13	Fatores em comum	Chave % 13
15	5,3	1,13	-	2
27	3,3,3		-	1
28	2,2,7		-	2
39	3,13		13	0
18	2,3,3		-	5
30	5,2,3		-	4
42	2,2,2,2,3		-	3

**TRATANDO COLISÕES**

# Endereçamento fechado



# Fator $\alpha = n/m$ de carga

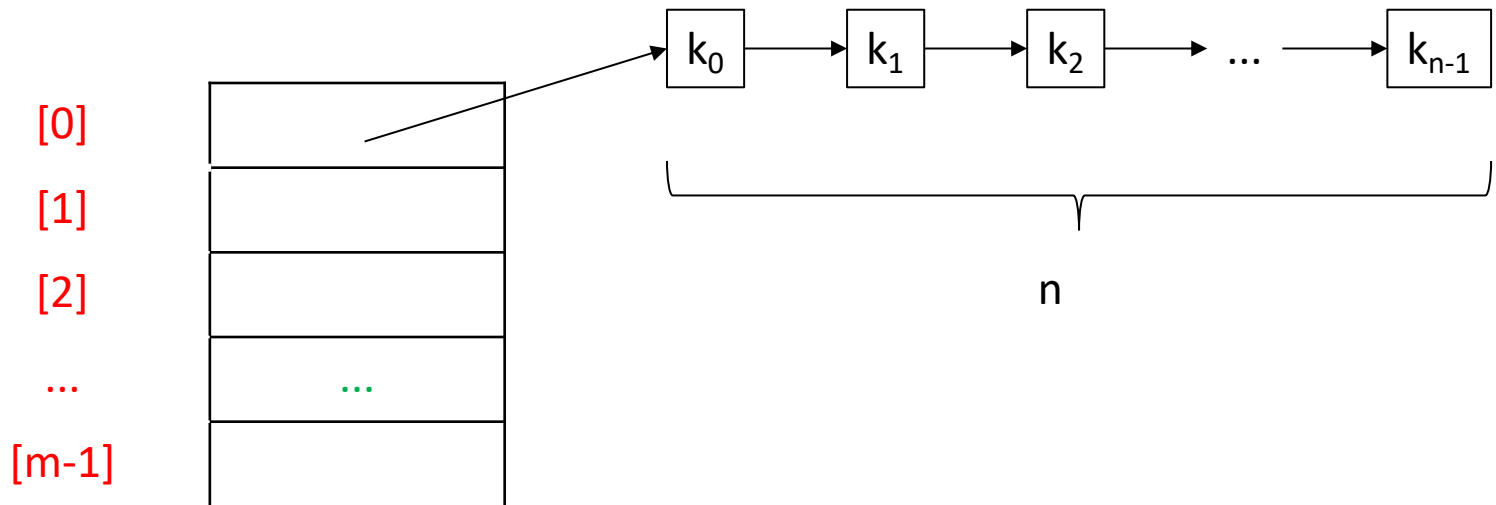
- Seja uma tabela T de hash contendo
  - m índices, cada um com uma lista para colisões
  - n chaves
- Fator  $\alpha = n/m$  indica *média* de chaves por lista

# Custo do Endereçamento Fechado

- Custo total = custo função  $h$  + custo das listas
- Custo da função  $h(k) = k \bmod m$  é  $O(1)$
- Custo das listas depende do ***caso***

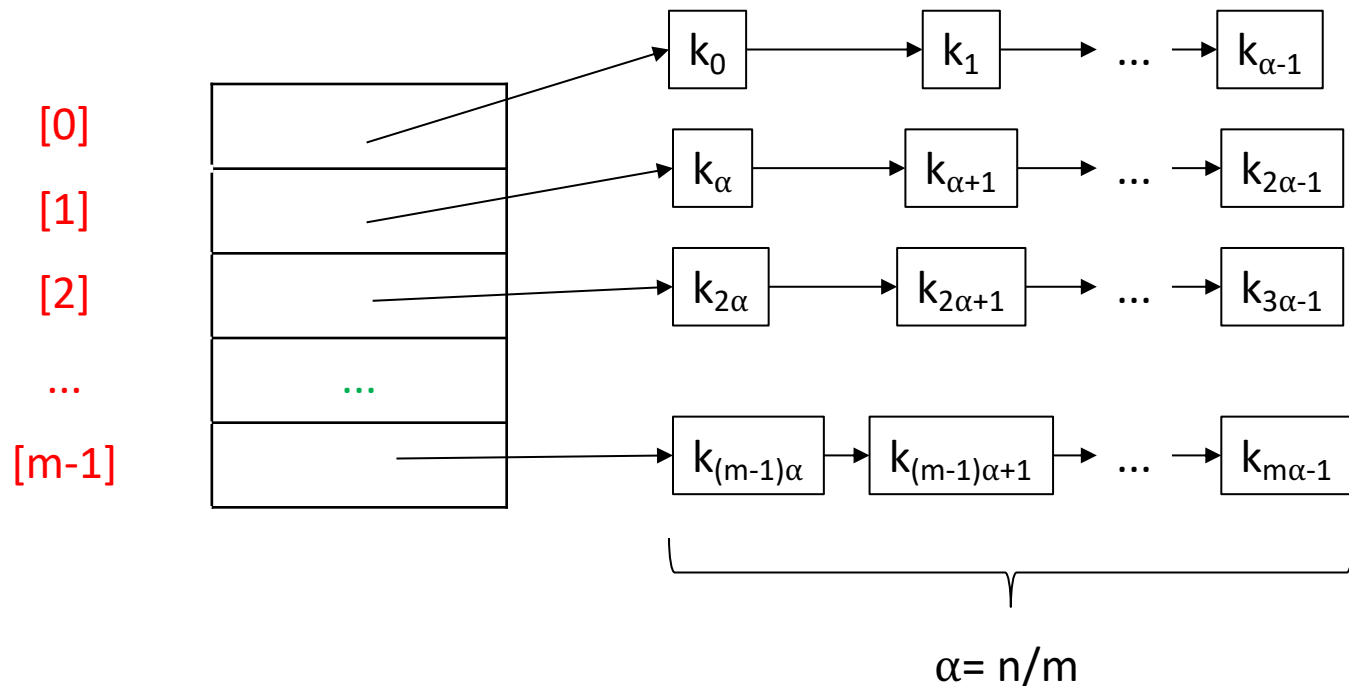
# Custo do Endereçamento Fechado

- Custo das listas no *pior* caso:  $O(1+n) = O(n)$



# Custo do Endereçamento Fechado

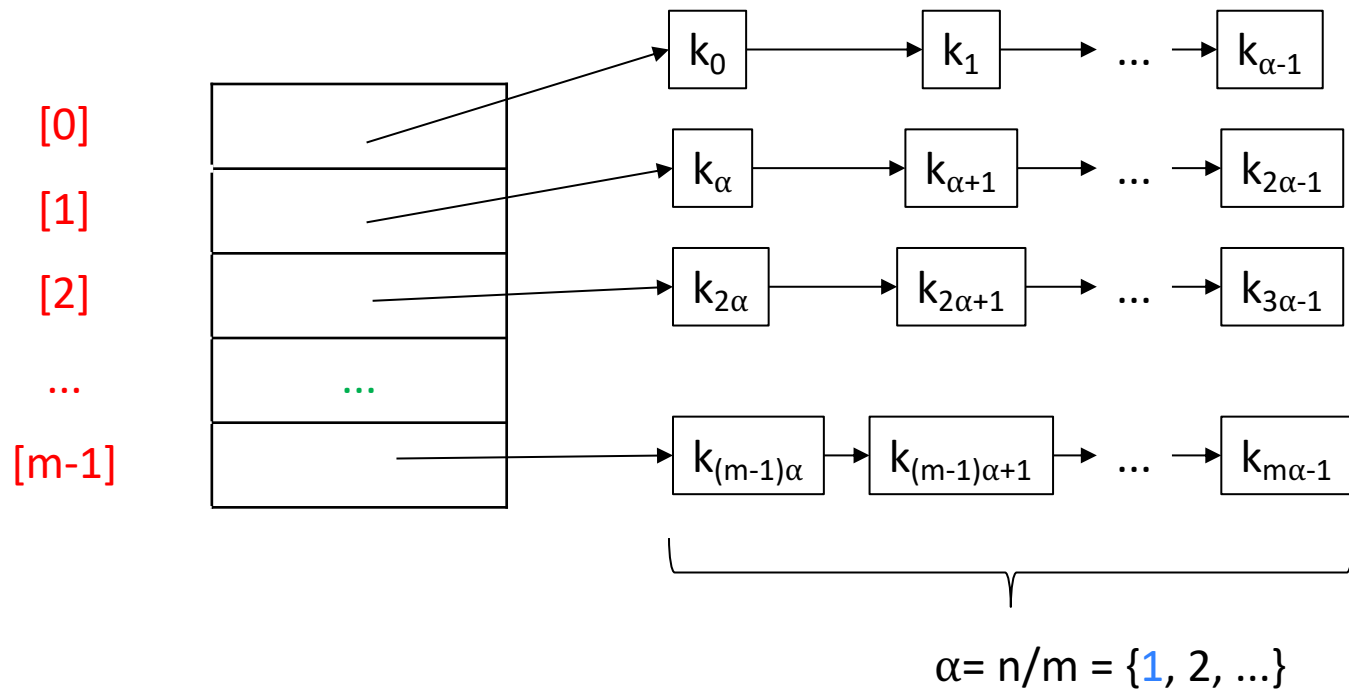
- Custo das listas no caso *médio*:  $O(1+\alpha) \sim O(n)$



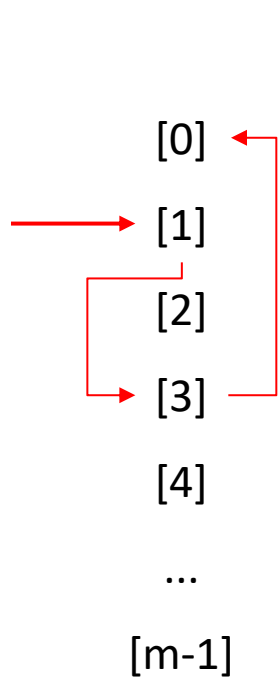


# Custo do Endereçamento Fechado

- Custo das listas no *melhor* caso:  $O(1+1) = O(1)$



# Endereçamento Aberto: índices **alternativos** invés de listas



	Chave	Removida?
[0]	$k_9$	0
[1]	$k_1$	0
[2]		0
[3]	$k_5$	0
[4]	$k_7$	1
...	...	
[m-1]		

Chave  $k_9$  é inserida na índice vazio 0  
após tentarmos índice 1 e 3 ocupados  
(3 iterações, no máximo  $m$ )

$$\alpha = n/m \leq 1$$

A busca pela chave  $k_9$  deverá  
visitar os índices 1, 3 e 0  
(3 iterações, no máximo  $m$ )

# Função $h$ gera **sequência** de índices

- Conjunto de chaves  $U = \{k_0, k_1, \dots, k_{n-1}\}$
- Tabela  $T$  com índices  $\{0, 1, \dots, m-1\}$ ,  **$m < n$**
- $h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$  gera a **sequência**  $h(k, 0), h(k, 1), \dots, h(k, m-1)$  de índices
- A **sequência** é permutação de  $\{0, 1, \dots, m-1\}$

# Função $h$ por *double hashing*

- Sequência aproxima seleção randômica
- $h_1$  e  $h_2$  são funções de hash auxiliares

$$h(k,i) = ( h_1(k) + i * h_2(k) ) \bmod m$$

Base (parte fixa)



Deslocamento  
(parte incremental)



# Função $h$ por *double hashing*

- $h_2(k)$  relativ. primo ao tamanho  $m$  da tabela garante que toda tabela seja percorrida.
- Estratégia #1:  $m=2^w$  e  $h_2(k)$  ímpar
  - Exemplo:  $m=32$  e  $h_2(k)=2*k+1$
- Estratégia #2:  $m$  primo e  $0 < h_2(k) < m$ ,  $h_2 \in I$ 
  - Exemplo:  $m=31$  e  $h_2(k)=1+ (k \bmod 30)$

# Custo End. Aberto + double hashing

- Fator  $\alpha = n/m < 1$
- Custo: número de iterações  $\leq m$
- Número máximo de iterações:  $1/(1-\alpha)$ 
  - Exemplos:
    - $\alpha = 0.5 \Rightarrow 2$  tentativas
    - $\alpha = 0.9 \Rightarrow 10$  tentativas
  - Ver Cormen, Teorema 11.6

# Resumo: Tabelas de Hash

- Podem ser mais eficientes que árvores, listas
- Colisões são comuns e afetam eficiência
- End. aberto: simples mas custo  $O(n)$
- End. fechado: complexo mas eficiente