

# Fundamentos de análisis y diseño de algoritmos

Tablas Hash

Abril 2018

Tablas de direccionamiento directo

Tablas hash

Funciones hash

Método de división

Método de multiplicación

# Tablas Hash

```
programa1(int n)
```

```
  x ← 1
```

```
  var1 ← n
```

```
  var2 ← 0
```

```
  while (x < n)
```

```
    var2 ← var2 + var1
```

```
    x ← x+1
```

```
  print x
```

Los compiladores llevan una tabla de símbolos cuya llave son los identificadores de las variables

x	1
var1	10
var2	0
n	10
z	5

Var2  
z?

1000  
            
O(n)

# Tablas Hash

```
programa1(int n)
```

```
  x ← 1
```

```
  var1 ← n
```

```
  var2 ← 0
```

```
  while (x < n)
```

```
    var2 ← var2 + var1
```

```
    x ← x+1
```

```
  print x
```

Los compiladores llevan una tabla de símbolos cuya llave son los identificadores de las variables

0		→	x	1
1		→	var1	10
2		→	var2	0
3		→	n	10
4				
5				
6				
7				
8				
9				

# Tablas Hash

```
programa1(int n)
```

```
  x ← 1
```

```
  var1 ← n
```

```
  var2 ← 0
```

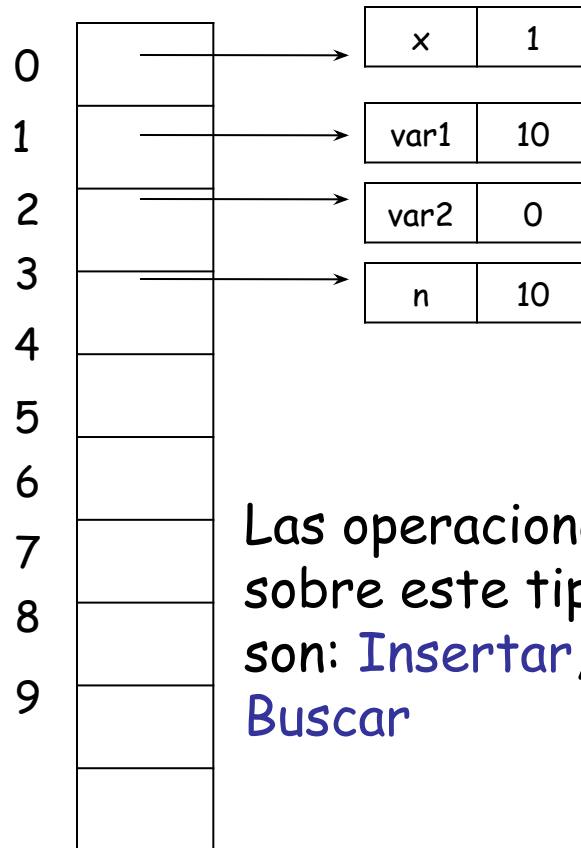
```
  while (x < n)
```

```
    var2 ← var2 + var1
```

```
    x ← x+1
```

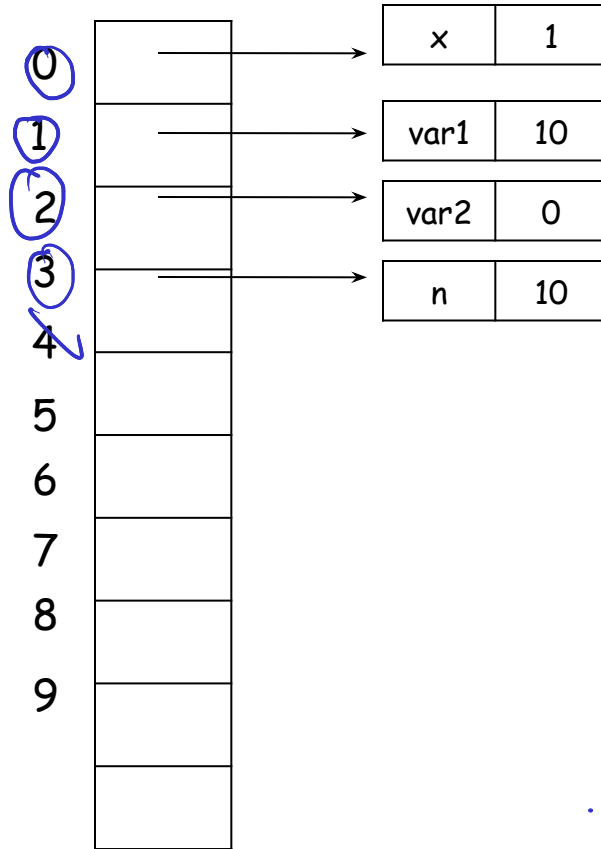
```
  print x
```

Los compiladores llevan una tabla de símbolos cuya llave son los identificadores de las variables



Las operaciones básicas sobre este tipo de tablas son: **Insertar**, **Borrar** y **Buscar**

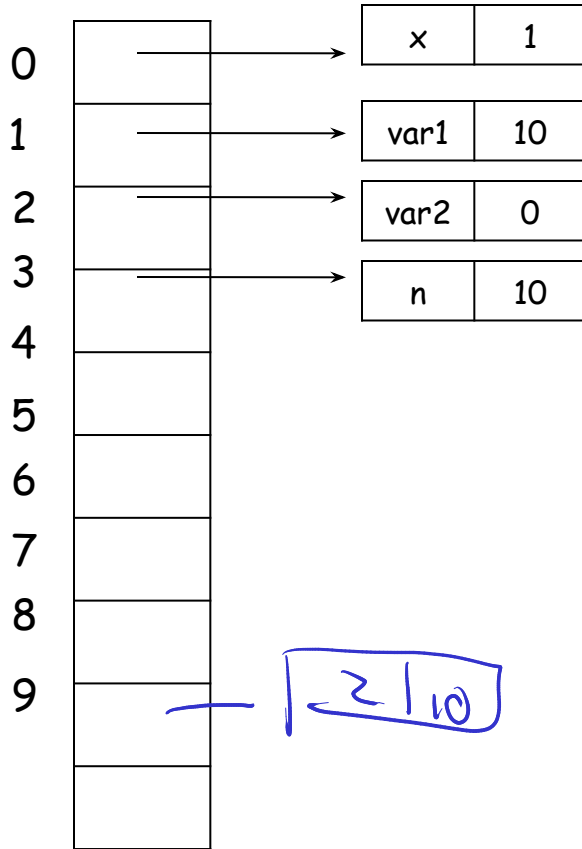
# Tablas Hash



¿Qué tan costoso puede ser **insertar** un par (llave, valor) en la tabla?

En qué posición de la tabla se debería almacenar un nuevo dato?

# Tablas Hash

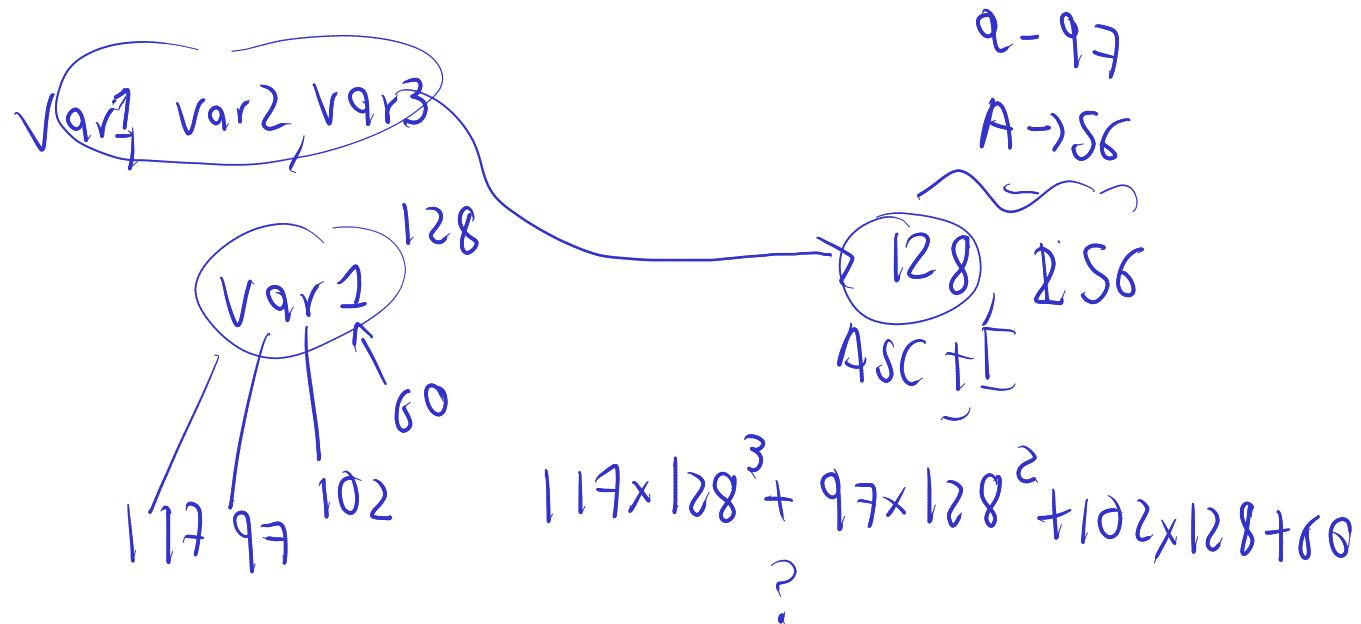


¿Qué tan costoso puede ser  
buscar un par (llave, valor) en  
la tabla?

# Tablas Hash

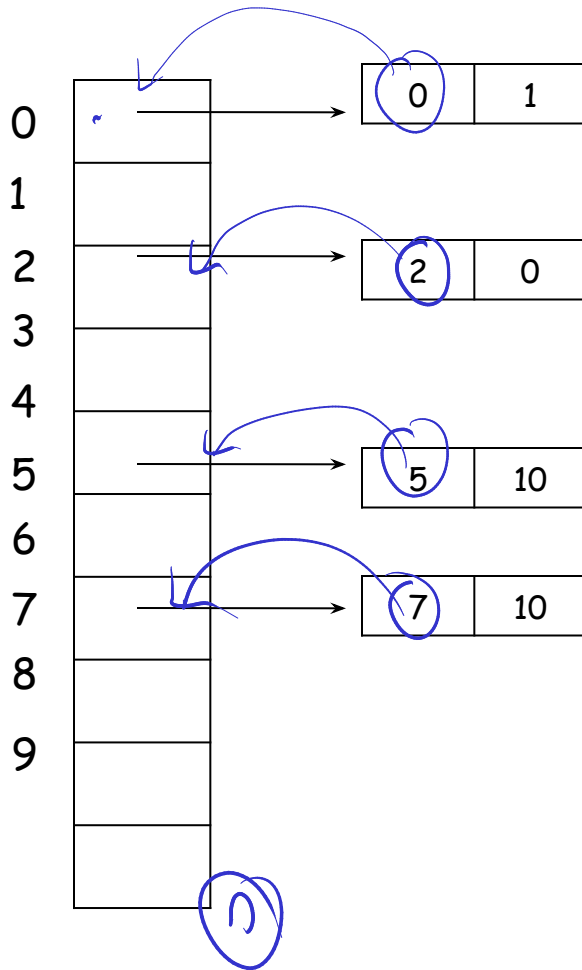
0		→	192	1
1		→	17	10
2		→	18	0
3		→	128	10
4				
5				
6				
7				
8				
9				

Las llaves se manejan como valores numéricos, en el caso de cadenas de caracteres, se convierten a un número entero utilizando código ASCII





# Tablas Hash



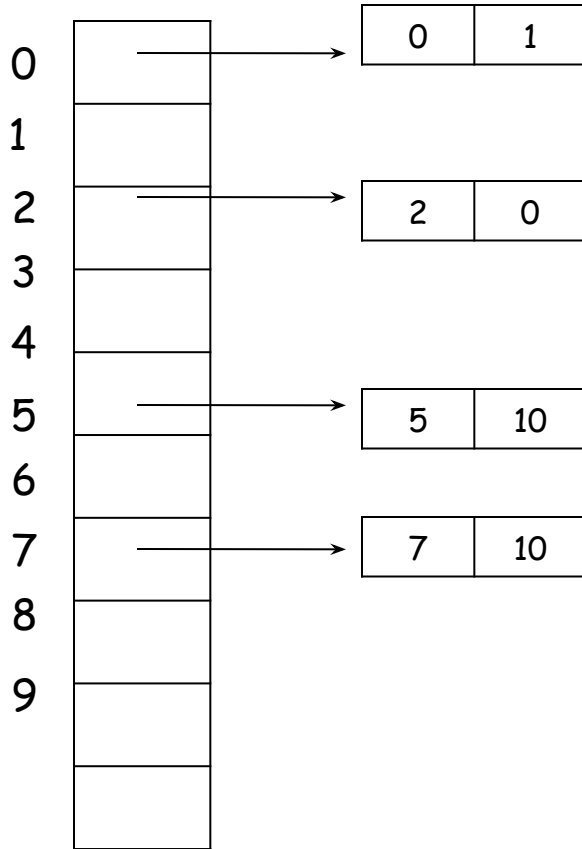
Una estrategia consiste en aprovechar que las llaves son numéricas y almacenar el par (llave, valor) en la posición "llave" de la tabla

Esta estrategia se conoce como  
**Tabla de direccionamiento directo**

¿Cuál es el tiempo de búsqueda ahora?

$O(1)$  7?  $O(1)$  18 | 12

# Tablas Hash



•DIRECT-ADDRESS-INSERT( $T, X$ )

$T[\text{key}[x]] \leftarrow x$

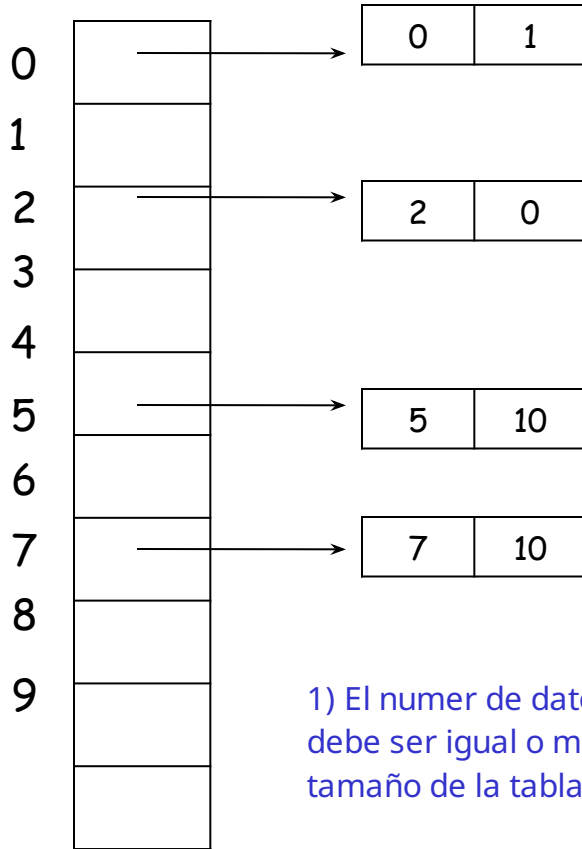
•DIRECT-ADDRESS-SEARCH( $T, k$ )

return  $T[k]$

•DIRECT-ADDRESS-DELETE( $T, k$ )

$T[\text{key}[x]] \leftarrow \text{nil}$

# Tablas Hash



1) El número de datos debe ser igual o menor al tamaño de la tabla

2) Las llaves deben estar entre 0 y  $n-1$

3) ¿Qué pasa si dos datos comparten la MISMA LLAVE?

• DIRECT-ADDRESS-INSERT( $T, X$ )

$T[\text{key}[x]] \leftarrow x$

• DIRECT-ADDRESS-SEARCH( $T, k$ )

return  $T[k]$

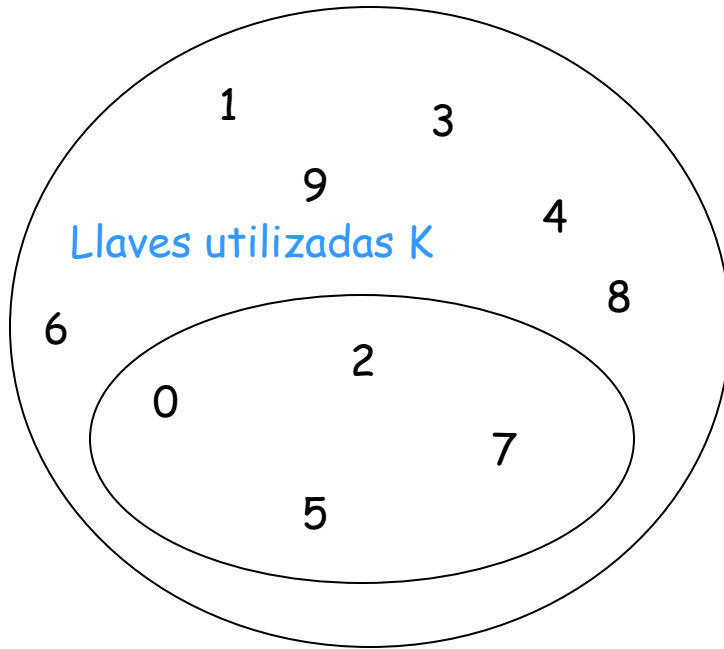
• DIRECT-ADDRESS-DELETE( $T, k$ )

$T[\text{key}[x]] \leftarrow \text{nil}$

• Todas estas operaciones toman tiempo constante  $O(1)$

# Tablas Hash

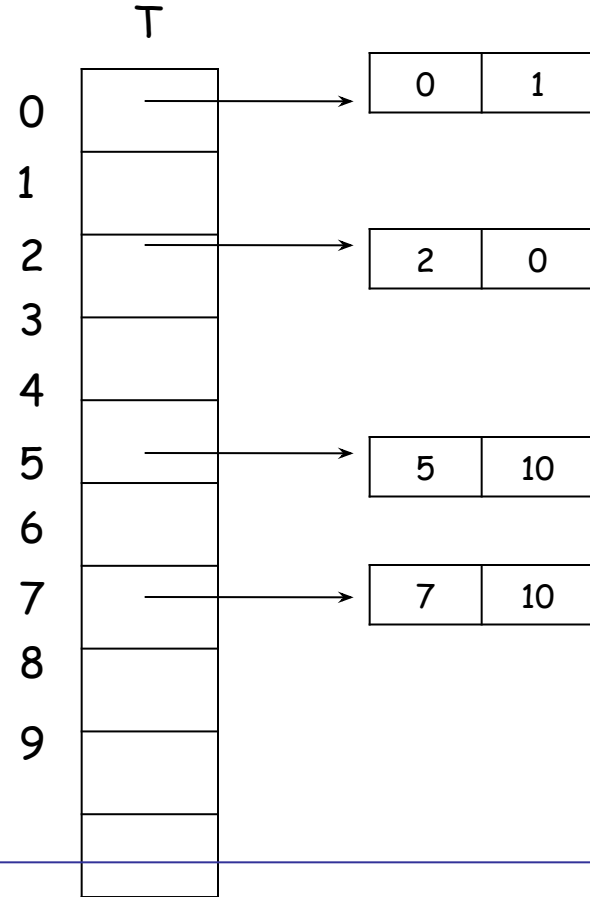
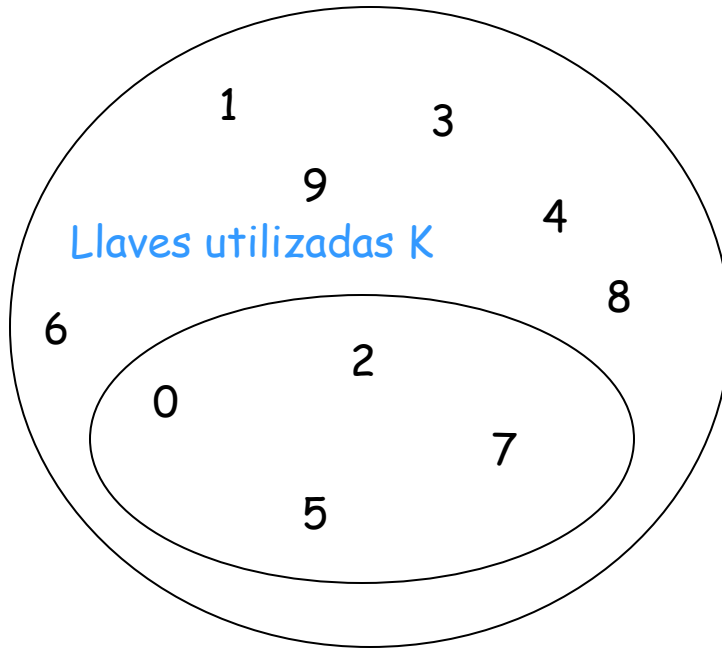
Universo de llaves U



0		→	0	1
1				
2		→	2	0
3				
4				
5		→	5	10
6				
7		→	7	10
8				
9				

# Tablas Hash

Universo de llaves U



$U = \{0, 1, \dots, m-1\}$ , donde  $|U| = m$

La tabla de direccionamiento directo T, se puede ver como un arreglo  $T[0, \dots, m-1]$  donde cada posición, o slot, corresponde a una llave en U

# Tablas Hash

---

## Tabla de direccionamiento directo T

Considere  $K=\{1,2,3,4,5\}$  el conjunto de llaves actuales,  $U=\{0,1,\dots,9\}$  y las siguientes operaciones:

DIRECT-ADDRESS-INSERT(T,2)

DIRECT-ADDRESS-INSERT(T,4)

DIRECT-ADDRESS-INSERT(T,3)

DIRECT-ADDRESS-INSERT(T,1)

Muestre el contenido de la tabla de direccionamiento directo

# Tablas Hash

---

Describa un procedimiento para encontrar el elemento máximo de una tabla  $T$  de tamaño  $m$ . Indique su complejidad

# Tablas Hash

---

Considere el caso en el que tuviese que almacenar 1000 datos utilizando una tabla de direccionamiento directo

$$|U| = 10 \quad |K| = 1000$$

¿Qué pasa si  $|K| \ll |U|$ ?

$$|U| = 10^6 \quad |K| = 10$$



# Tablas Hash

---

Considere el caso en el que tuviese que almacenar 1000 datos utilizando una tabla de direccionamiento directo

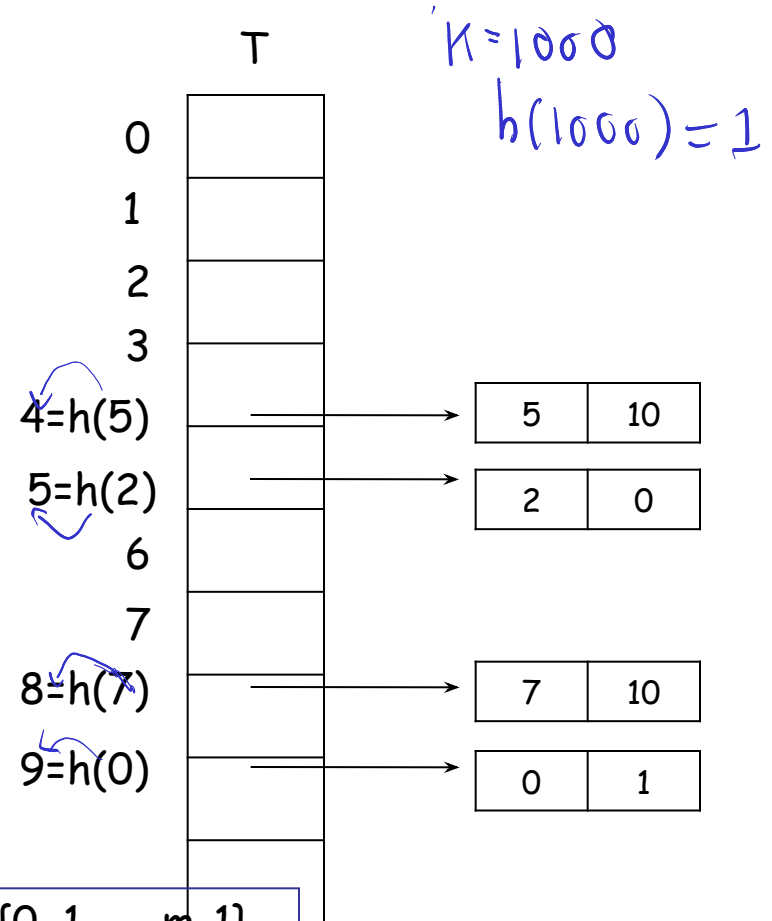
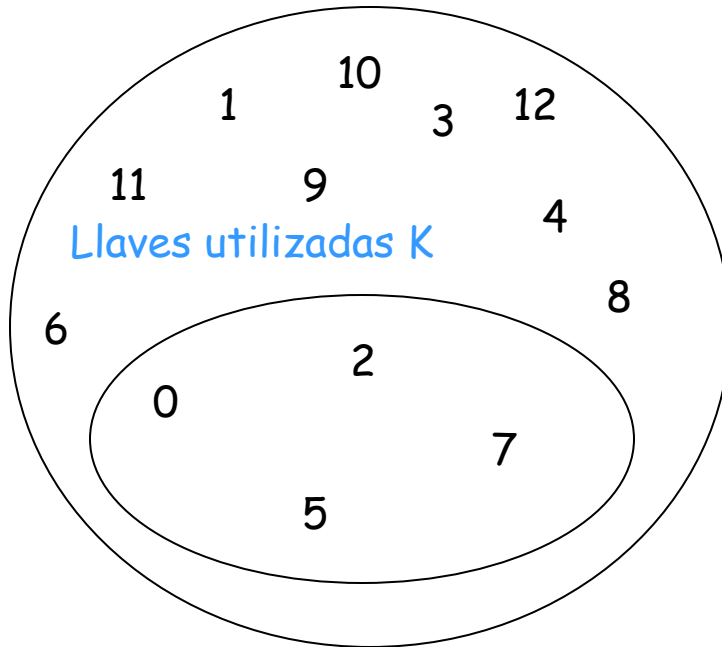
¿Qué pasa si  $|K| \ll |U|$ ?

Los requerimientos de memoria pueden llegar a ser de  $O(|U|)$  aun cuando no se utilicen todos los slots

Las **tablas hash** ofrecen un mecanismo para asignar la posición de almacenamiento para las llaves, de tal forma que los requerimientos de memoria pueden ser de  $O(|K|)$

# Tablas Hash

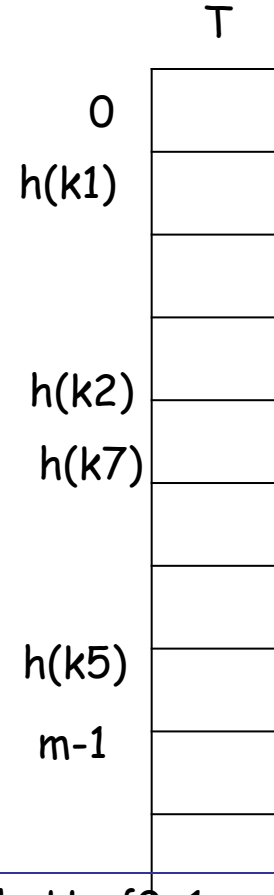
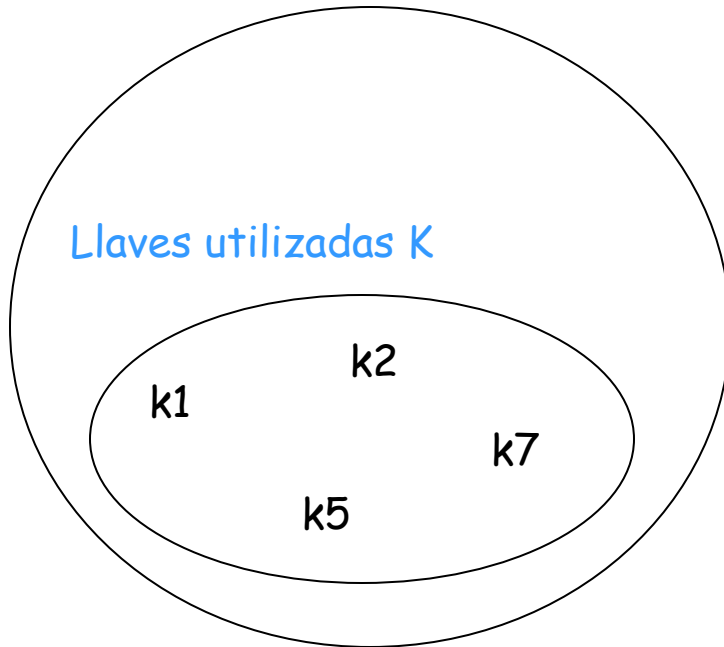
Universo de llaves  $U$ , ahora  $|U| > m$



Las tablas hash utilizan una función  $h: U \rightarrow \{0, 1, \dots, m-1\}$

# Tablas Hash

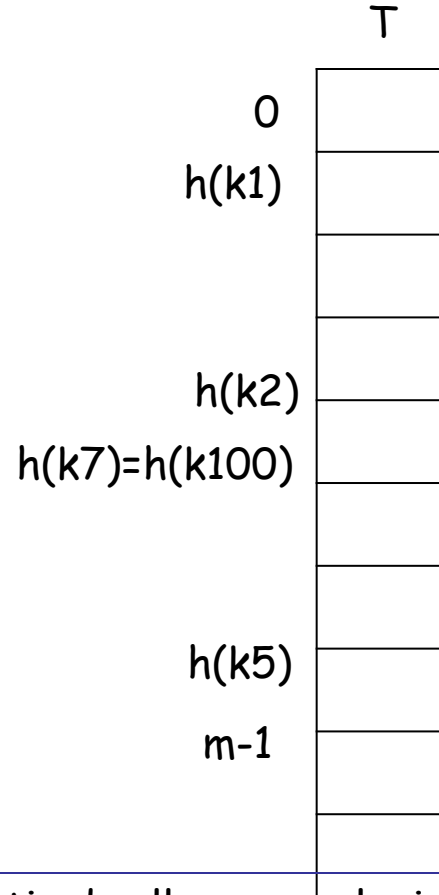
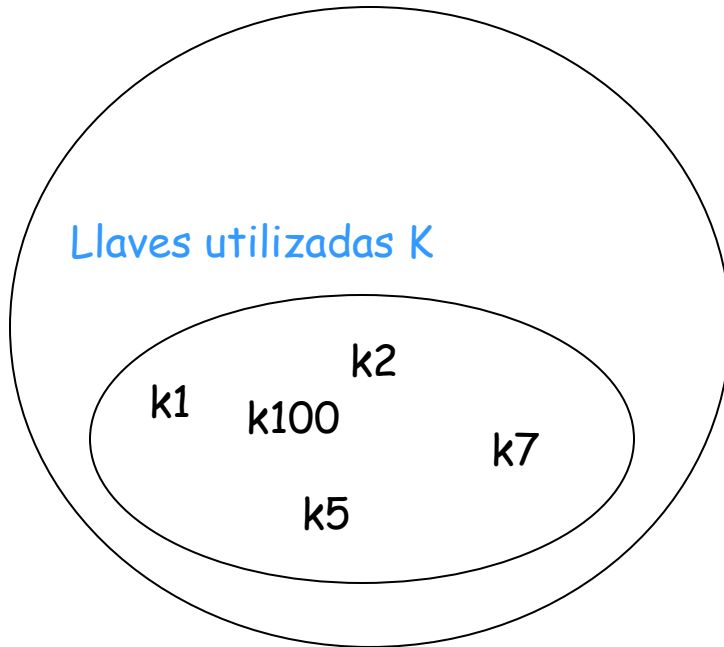
Universo de llaves  $U$ , ahora  $|U| > m$



Las tablas hash utilizan una función  $h: U \rightarrow \{0, 1, \dots, m-1\}$   
La tarea de  $h$  es asignar el slot a cada llave

# Tablas Hash

Universo de llaves  $U$ , ahora  $|U| > m$



Como  $|U| > m$ , pueden existir dos llaves en el mismo slot, esto se llama una **colisión**

# Tablas Hash

---

Tabla hash (suponga que  $\text{key}(x)=x$  y  $m=5$ )

Sea  $h(1)=1$ ,  $h(4)=1$ ,  $h(2)=3$ ,  $h(5)=3$ ,  $h(3)=4$

`HASH-INSERT(T,1)`

`HASH-INSERT(T,2)`

`HASH-INSERT(T,3)`

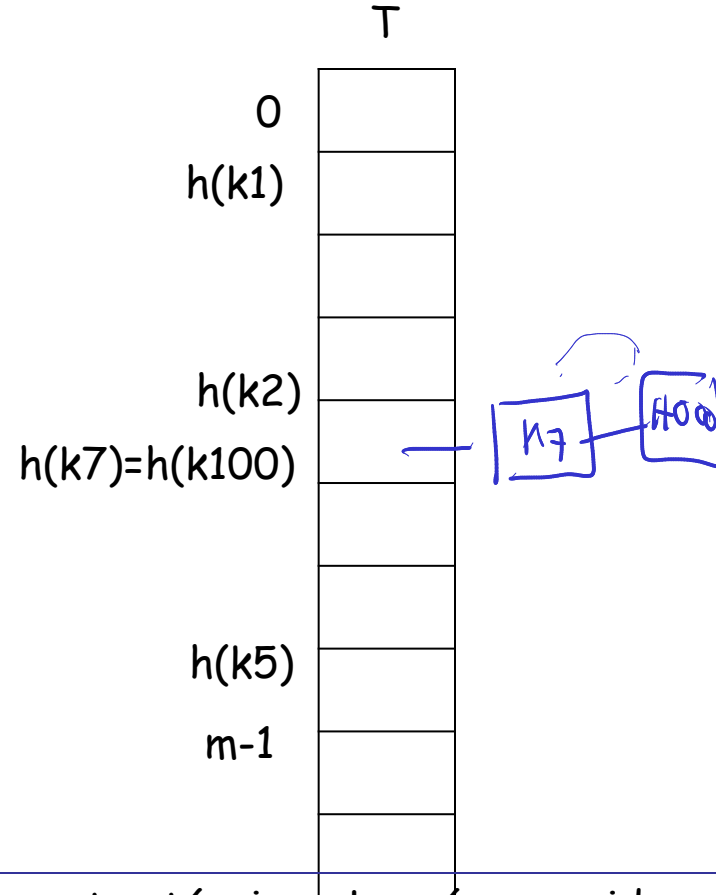
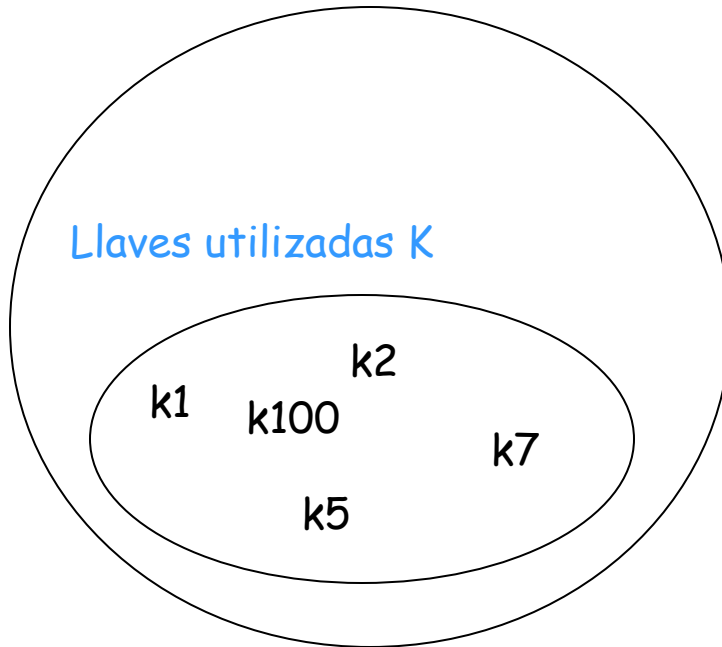
`HASH-INSERT(T,4)`

`HASH-INSERT(T,5)`

Muestre la tabla hash

# Tablas Hash

Universo de llaves  $U$ , ahora  $|U| > m$



Las colisiones se tratan con diferentes técnicas. La más conocida es la **resolución de colisiones por encadenamiento**

# Tablas Hash

Universo de llaves  $U$ , ahora  $|U| > m$

Llaves utilizadas  $K$

$k_1$   $k_2$   $k_{100}$   $k_7$   $k_5$

$$O(n) \rightarrow \frac{\infty}{O(1)}$$

$$O(1)$$

$$O(\infty)$$

$k_{100}$

$T$

0

$h(k_1)$

$h(k_2)$

$h(k_5)$

$m-1$

$k_1$

$k_2$

$k_7$

$k_5$

$k_{100}$

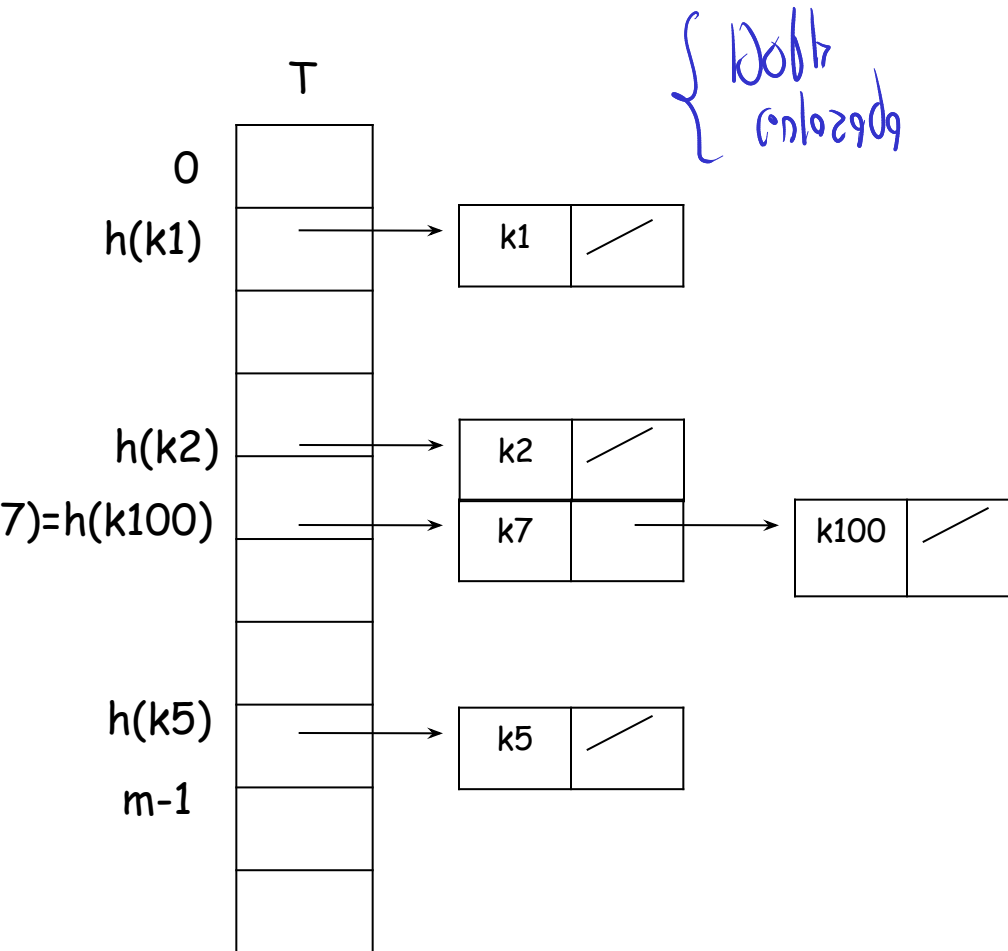
Las colisiones se tratan con diferentes técnicas. La más conocida es la **resolución de colisiones por encadenamiento**

Cada slot  $T[j]$  tiene una lista encadenada de todas las llaves cuyo valor hash es  $j$

1000  
1000  
1000

$$1000 \cdot \infty = \frac{1000}{10}$$

# Tablas Hash



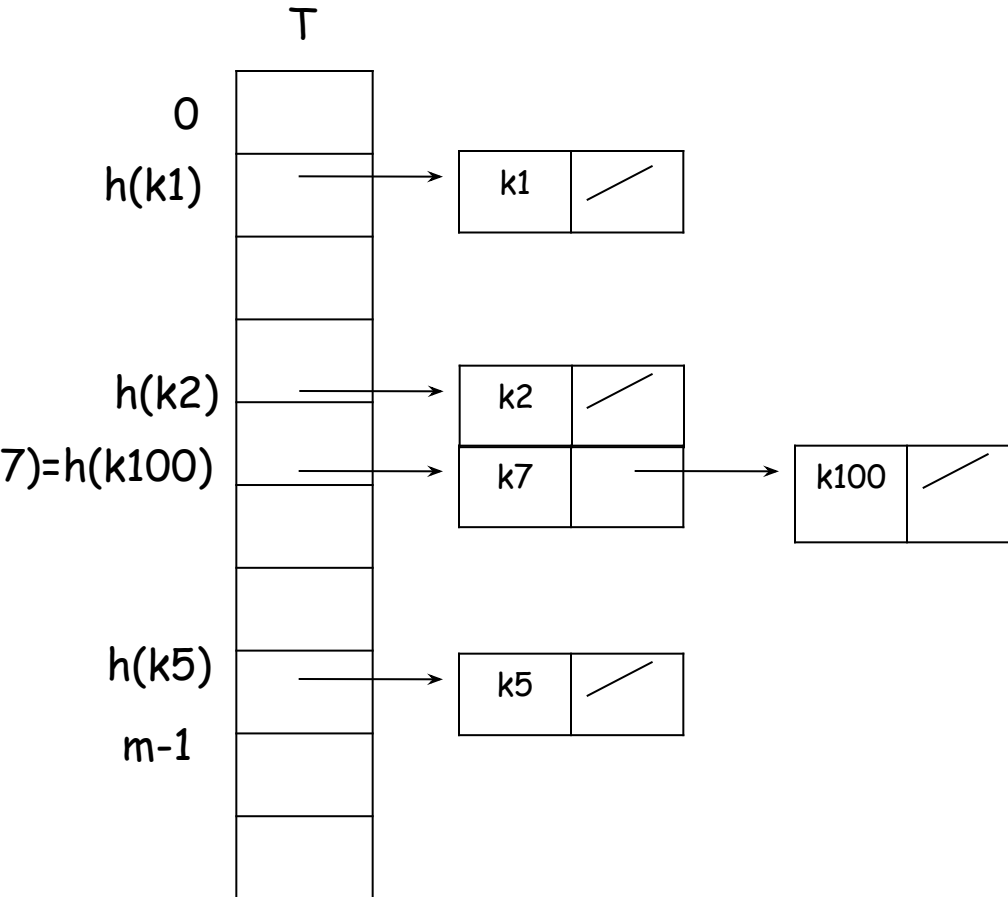
***CHAINED-HASH-INSERT( $T, x$ )***  
*insertar  $x$  en la cabeza de la lista  $T[h(\text{key}(x))]$*

***CHAINED-HASH-SEARCH( $T, k$ )***  
*buscar por un elemento con llave  $k$  en la lista  $T[h(\text{key}(k))]$*

***CHAINED-HASH-DELETE( $T, k$ )***  
*borrar  $x$  de la lista  $T[h(\text{key}(k))]$*



# Tablas Hash



***CHAINED-HASH-INSERT( $T, x$ )***  
*insertar  $x$  en la cabeza de la lista  $T[h(key(x))]$*

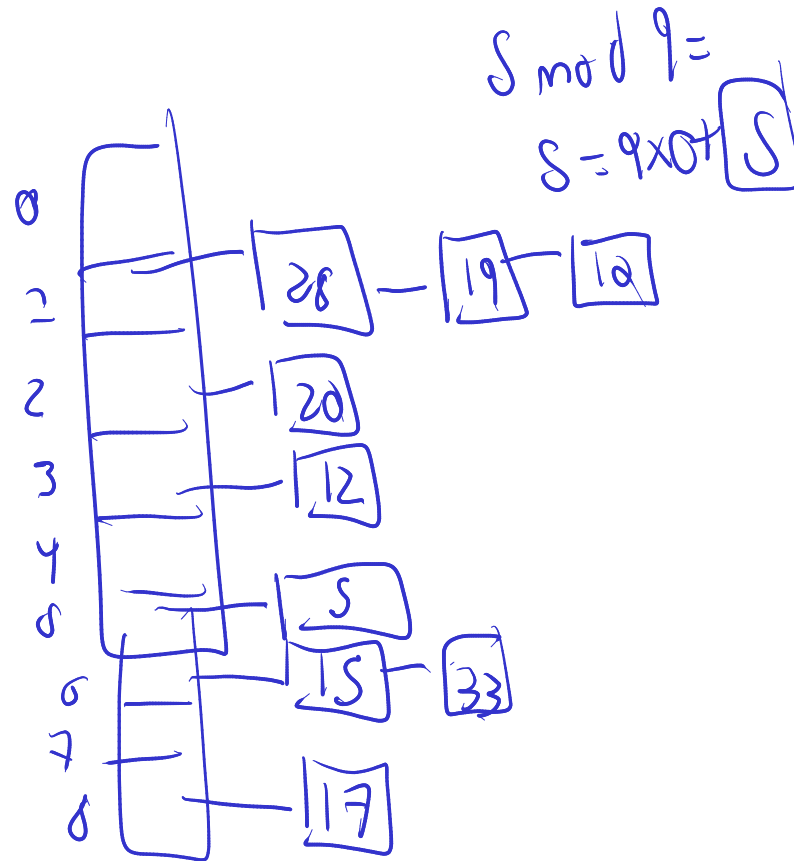
***CHAINED-HASH-SEARCH( $T, k$ )***  
*buscar por un elemento con llave  $k$  en la lista  $T[h(key(k))]$*

***CHAINED-HASH-DELETE( $T, k$ )***  
*borrar  $x$  de la lista  $T[h(key(k))]$*

***Analice las complejidades de las operaciones***

# Tablas Hash

Muestra la tabla T después de insertar las llaves 5, 28, 19, 15, 20, 33, 12, 17, 10 en una tabla hash con 9 slots siendo la función hash  $h(k) = k \bmod 9$



# Tablas Hash

¿Si se mantuvieran ordenados los elementos de cada lista encadenada, cómo cambian los tiempo para insertar, borrar, y buscar?

⑧

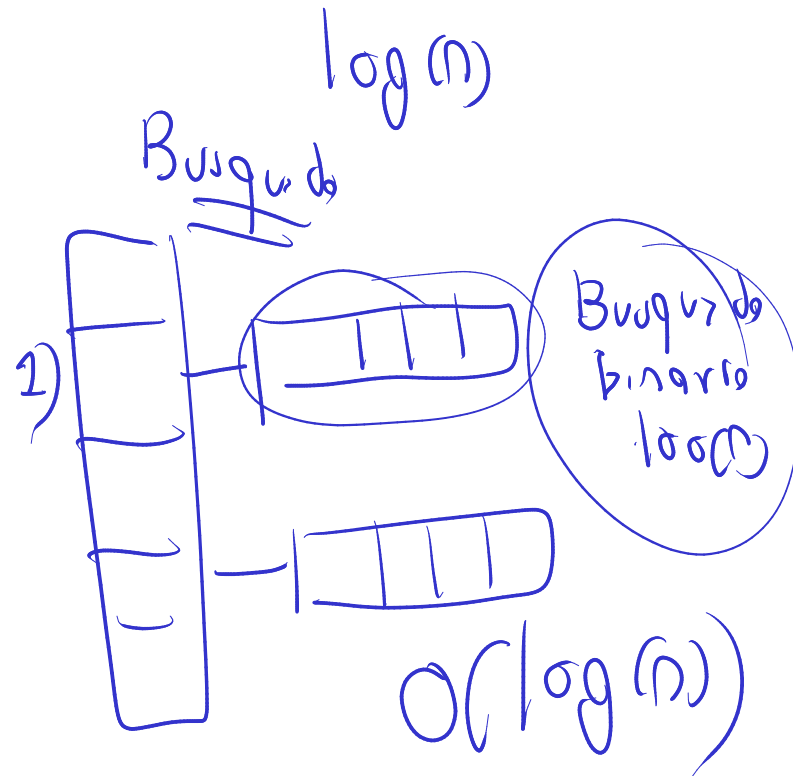
1 2 3 4 5 6  
↑  
→

10	7
10	9
10	1

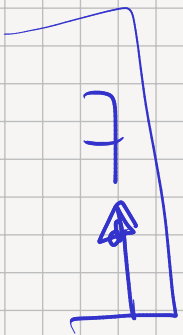
5 6  
↑  
5

7  
7 9

1 7 9  
↑




1 2 4 8 7 9 10 11




③


1 2 4 5



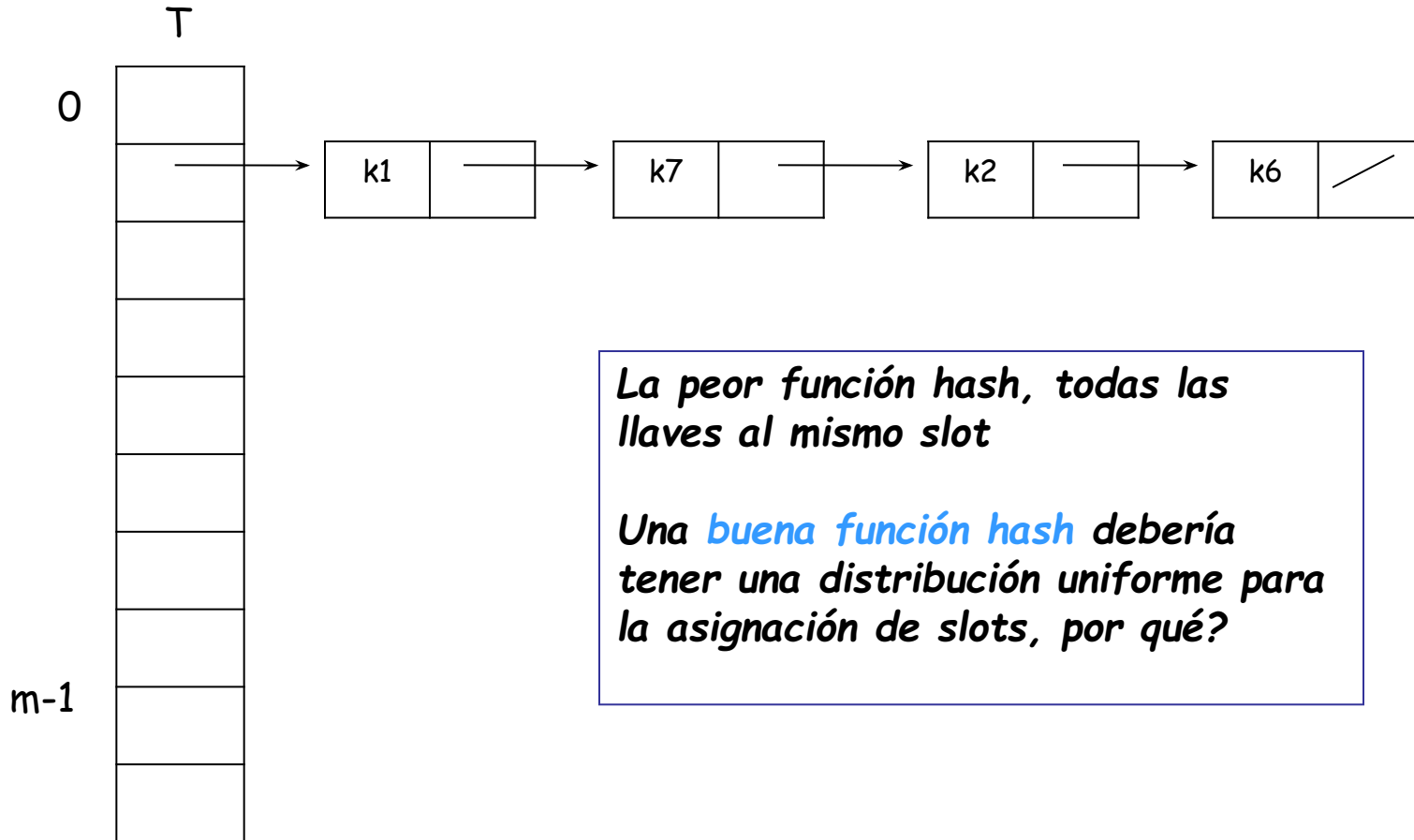
1 2



1 2 3 4



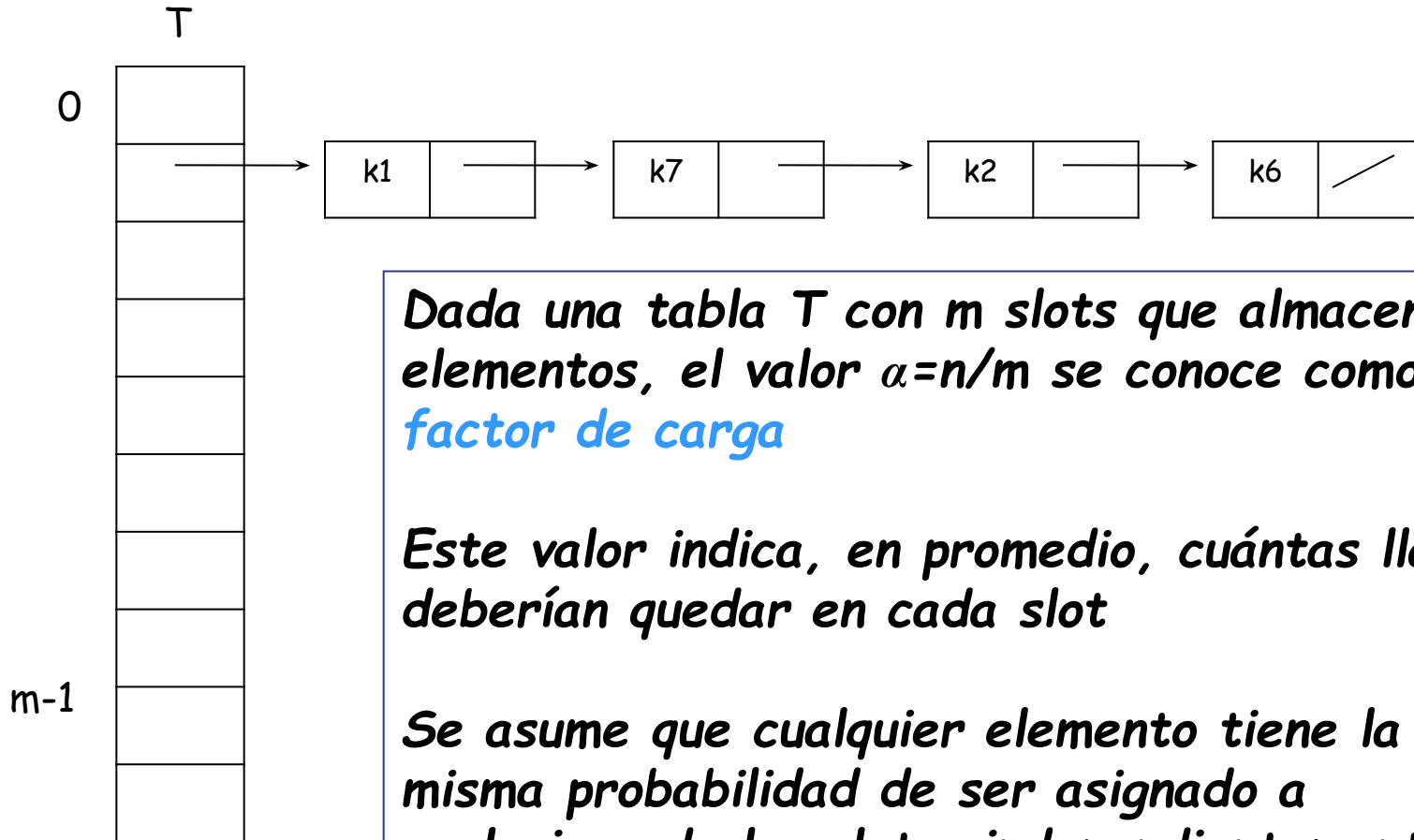
# Tablas Hash



*La peor función hash, todas las llaves al mismo slot*

*Una buena función hash debería tener una distribución uniforme para la asignación de slots, por qué?*

# Tablas Hash



*Dada una tabla  $T$  con  $m$  slots que almacena  $n$  elementos, el valor  $\alpha = n/m$  se conoce como **factor de carga***

*Este valor indica, en promedio, cuántas llaves deberían quedar en cada slot*

*Se asume que cualquier elemento tiene la misma probabilidad de ser asignado a cualquiera de los slots, independientemente de donde se hayan asignado otros elementos.*

***Suposición de hashing uniforme***

# Tablas Hash

---

## Teorema 1

En una tabla hash en la cual las colisiones son resueltas con encadenamiento, una búsqueda sin éxito toma en promedio  $\Theta(1+\alpha)$ , bajo la suposición de hasing uniforme

## Teorema 2

En una tabla hash en la cual las colisiones son resueltas con encadenamiento, una búsqueda exitosa toma en promedio  $\Theta(1+\alpha)$ , bajo la suposición de hasing uniforme

# Tablas Hash

---

Una buena función hash:

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m} \quad , \text{ para } j= 0, 1, \dots, m-1$$



# Tablas Hash

---

Es común tener en un programa nombres de identificadores que son similares, var1, var2, por ejemplo. Una buena función hash debería asignarlos a slots diferentes, así se muestra que existe independencia entre cada par de llaves

# Tablas Hash

---

## Llaves de tipo string

Cuando una llave es un string, se utiliza una transformación del código ASCII, en el cual se consideran los caracteres de 0 a 127

Tipo ASCII

$$pt = 112 * 128^1 + 116 * 128^0 = 14452$$

# Tablas Hash

---

## Funciones hash

Cómo evitar la colisiones o que por lo menos ocurran de tal forma que cualquier colisión sea igual de probable?

# Tablas Hash

---

## Una función hash

Para el caso en que las llaves sean números reales distribuidos en el rango  $0 \leq k < 1$ ,

$h(k) = \lfloor km \rfloor$ , donde  $T[0, 1, \dots, m-1]$



# Tablas Hash

---

Una función hash

$h(k) = \lfloor km \rfloor$ , donde  $T[0,1,\dots,m-1]$

# Tablas Hash

Complete la tabla utilizando la función:

$$h(k) = \lfloor km \rfloor,$$

para almacenar las llaves

$$k_1 = 0.4$$

$$k_2 = 1.2$$

$$k_3 = 1.8 \times$$

$$k_4 = 0.9$$

T	
0	
1	$\rightarrow k_1$
2	
3	$\rightarrow k_4$
4	$\rightarrow k_2$

$$k_1 = \lfloor 0.4 \times 4 \rfloor = 1$$

$$k_2 = \lfloor 1.2 \times 4 \rfloor = 4$$

$$k_3 = \lfloor 1.8 \times 4 \rfloor = 7 \times$$

$$k_4 = \lfloor 0.9 \times 4 \rfloor = 3$$

# Tablas Hash

---

## Método división

Utiliza la función hash:

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

# Tablas Hash

Complete la tabla utilizando la función:

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

para almacenar las llaves

$$m = 4$$

$$k_1 = 4$$

$$4 \bmod 4 = 0$$

$$k_2 = 2$$

$$2 \bmod 4 = 2$$

$$k_3 = 8$$

$$8 \bmod 4 = 0$$

$$k_4 = 9$$

$$9 \bmod 4 = 1$$

T	
0	$\leftarrow k_1 \leftarrow k_3$
1	$\leftarrow k_4$
2	$\leftarrow k_2$
3	



# Tablas Hash

---

Complete la tabla utilizando la función:

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

para almacenar las llaves

$$k_1 = 4$$

$$k_2 = 2$$

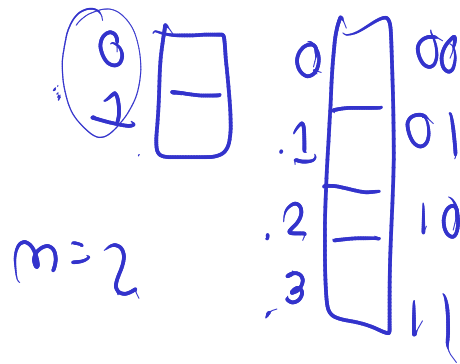
$$k_3 = 8$$

$$k_4 = 9$$

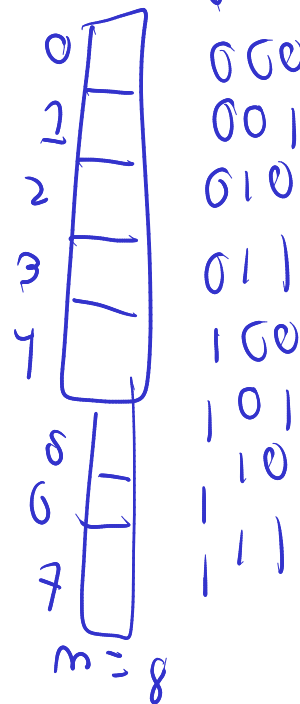
T	
0	K1,k3
1	k4
2	K2
3	

# Tablas Hash

A nivel de bits, si  $m$  es potencia de 2, se cumple que el valor  $h(k)$  dependerá los bits de más bajo orden de  $k$ . haciendo que  $h(k)$  no dependa de todos los valores de  $k$ .

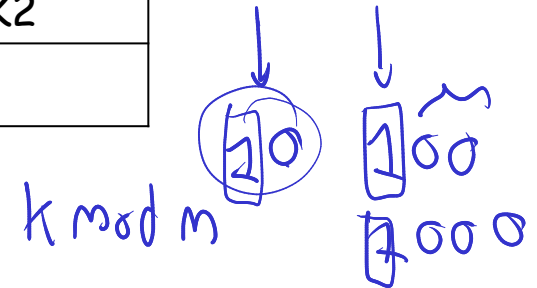


00  
01  
10  
11  
 $m=4$



T

0	K1, k3
1	k4
2	K2
3	



# Tablas Hash

---

## Método multiplicación

Utiliza la función hash:

Tamaño tabla

$h(k) = [m * (KA \bmod 1)]$ , donde  $A$  es cualquier número real entre 0 y 1

Hash

El valor de  $A$  no es crítico

# Tablas Hash

## Método multiplicación

Sea  $m=10000$ ,  $A=0.61803$ , muestre los valores  $h(k)$  que se asignarían para  $K=1000$ ,  $123400$ ,  $40321$  y  $10002$

$$h(k) = \lfloor 10000 * (0.61803 * k \bmod 1) \rfloor$$

$$\begin{aligned} & (0.61803 \times 1000 \bmod 1) \\ & \lfloor 10000 * (618,03 \bmod 1) \rfloor \\ & 10000 \times 0,03 = 300 \end{aligned}$$

$$k = 123400$$

$$9020$$

$$k = 40321$$

$$8876$$

$$k = 10002$$

$$5360$$

# Referencias

---

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press. Chapter 10

# Gracias

---

Próximo tema:

Estructuras de datos: Árboles binarios de búsqueda