

# ANÁLISIS Y DISEÑO DE ALGORITMOS I

## Periodo I – 2023

Jesús Aranda

Universidad del Valle  
Escuela de Ingeniería de Sistemas y Computación

Este documento es una adaptación del material original del profesor Oscar Bedoya



Tablas de direccionamiento directo

Tablas hash

Funciones hash

Método de división

Método de multiplicación

# Tablas Hash

## programa1(int n)

$x \leftarrow 1$

var1←n

var2←0

while ( $x < n$ )

`var2←var2 + var1`

$x \leftarrow x + 1$

print x

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

# Tablas Hash

```
programa1(int n)
```

```
x<1
```

```
var1<n
```

```
var2<0
```

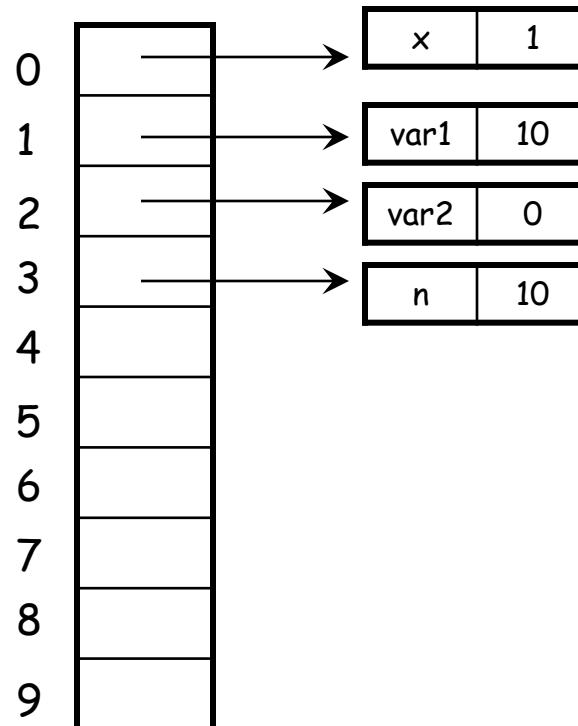
```
while (x<n)
```

```
    var2<var2 + var1
```

```
    x<x+1
```

```
print x
```

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



# Tablas Hash

```
programa1(int n)
```

```
x<1
```

```
var1<n
```

```
var2<0
```

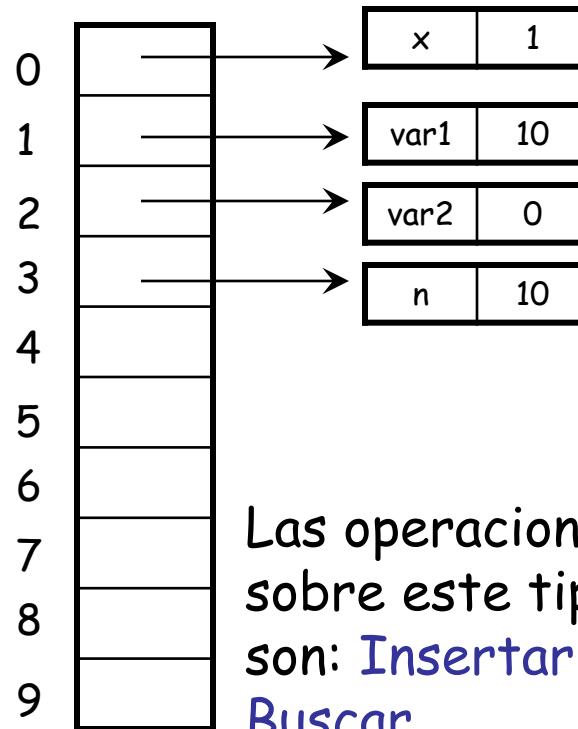
```
while (x<n)
```

```
    var2<var2 + var1
```

```
    x<x+1
```

```
print x
```

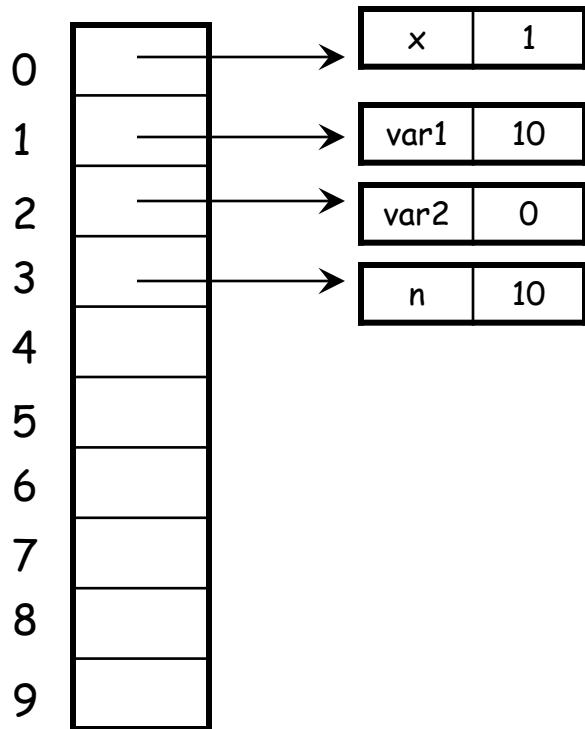
Los compiladores llevan una tabla de símbolos cuya llave son los identificadores de las variables



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

# Tablas Hash

---

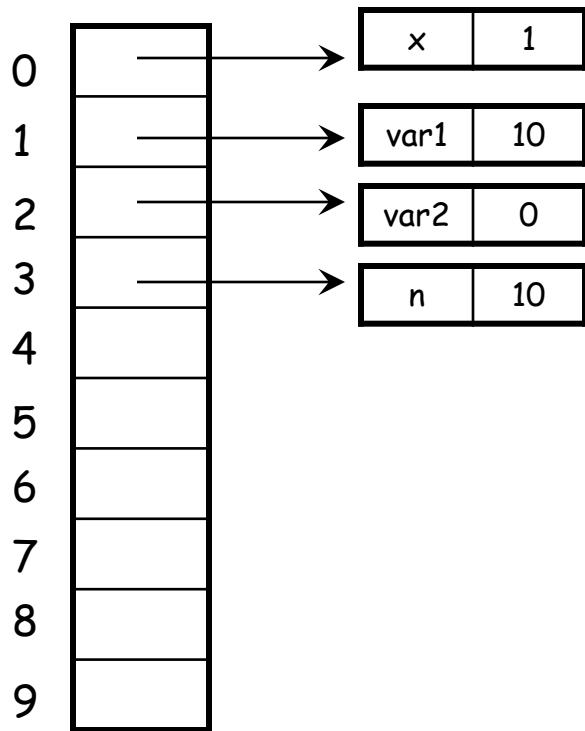


¿Qué tan costoso puede ser insertar un par (llave, valor) en la tabla?

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

# Tablas Hash

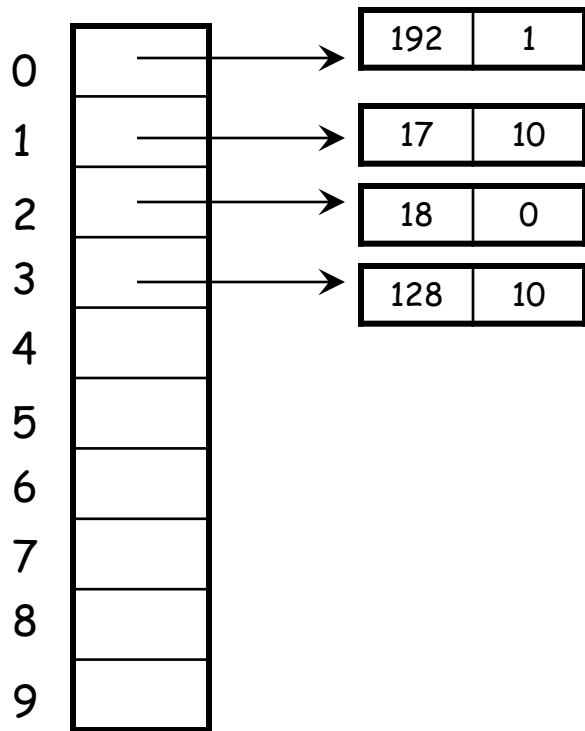
---



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

# Tablas Hash

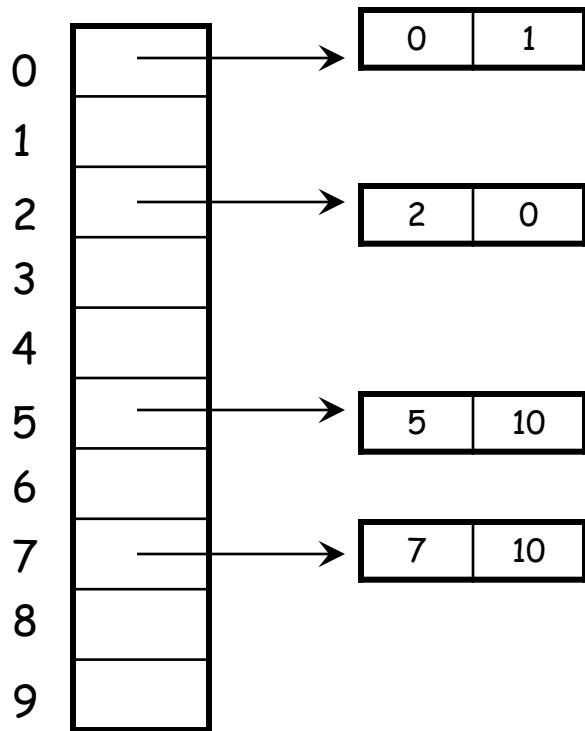
---



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

---



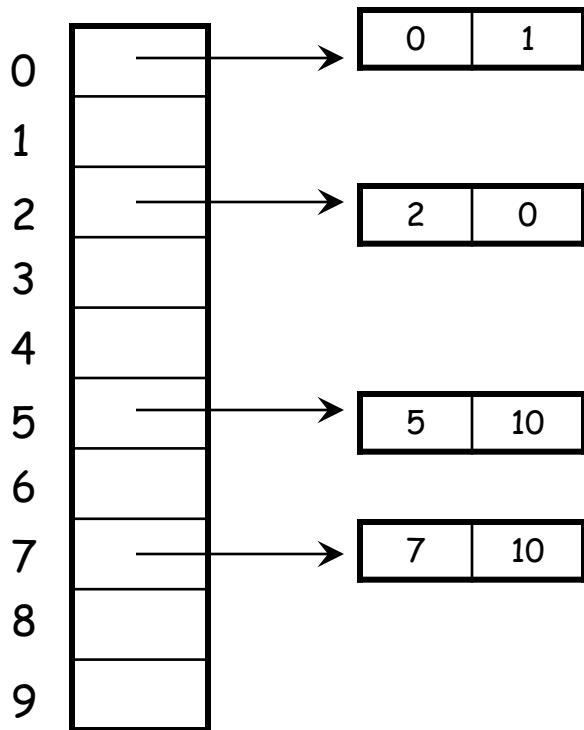
Una estrategia consiste en aprovechar que las llaves son numéricas y almacenar el par (llave, valor) en la posición "llave" de la tabla

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

¿Cuál es el tiempo de búsqueda ahora?

# Tablas Hash

---



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

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

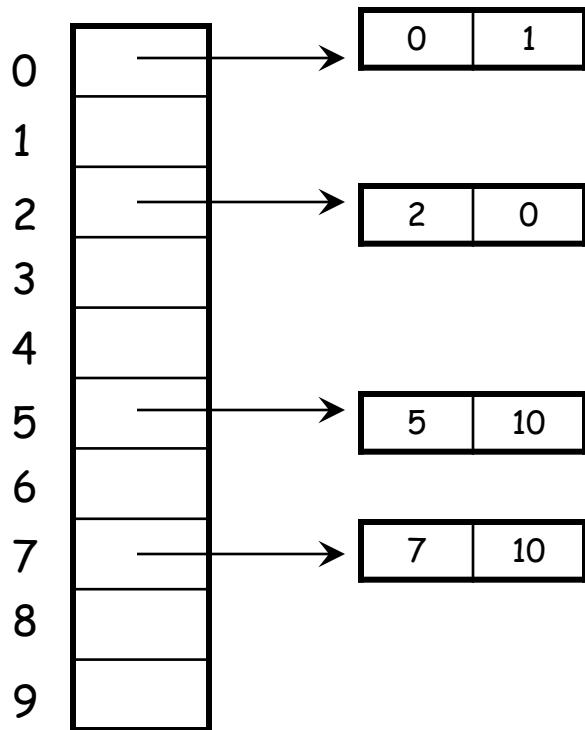
- DIRECT-ADDRESS-SEARCH( $T, k$ )

$$\text{return } T[k]$$

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

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

# Tablas Hash



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

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

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

$$\text{return } T[k]$$

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

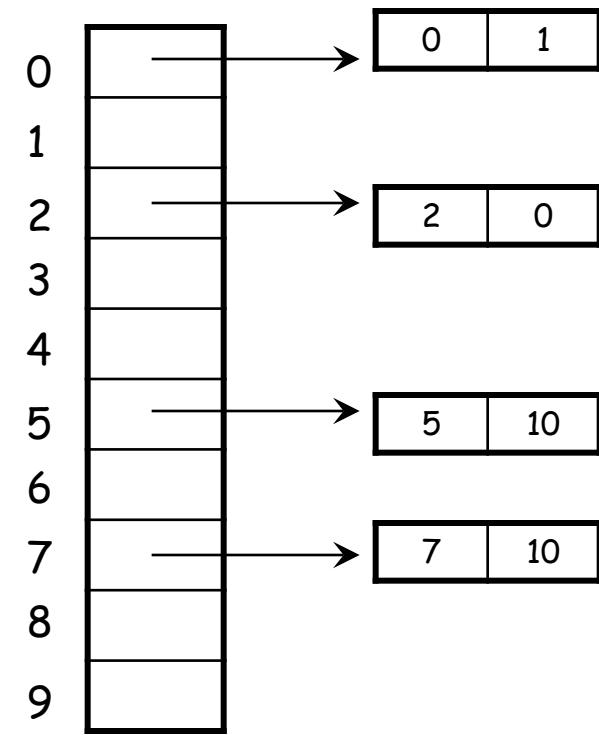
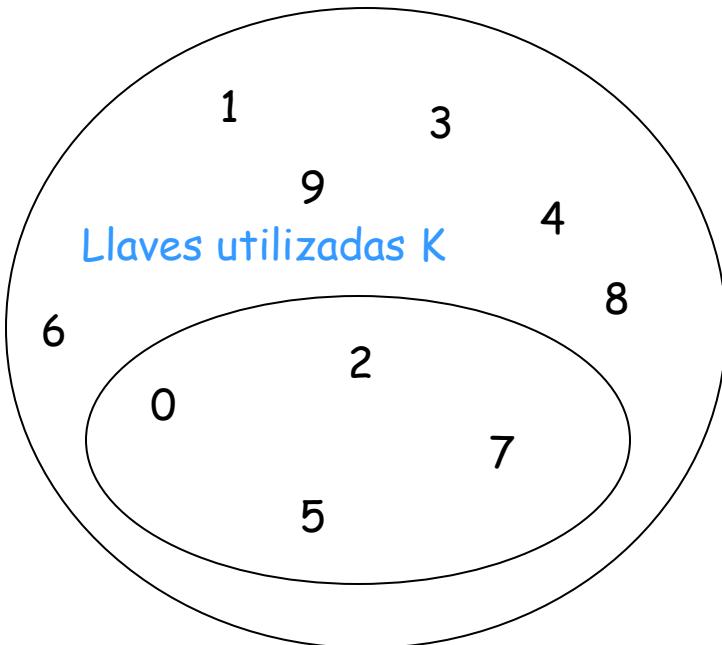
$$T[\text{key}[x]] \leftarrow \text{nil}$$

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

# Tablas Hash

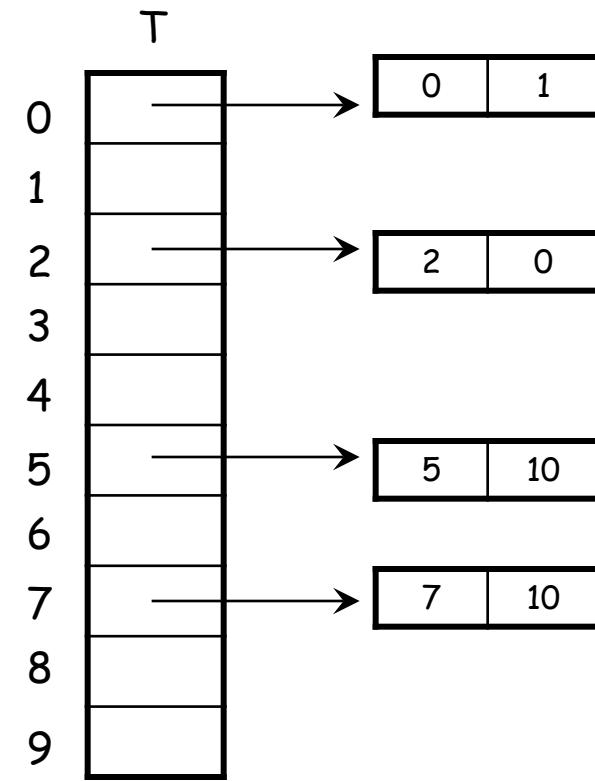
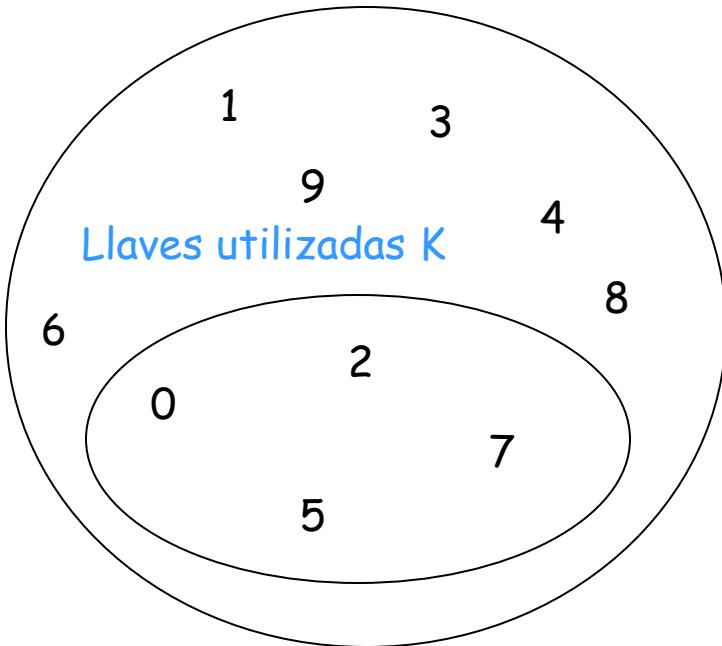
---

Universo de llaves U



# 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  $U$

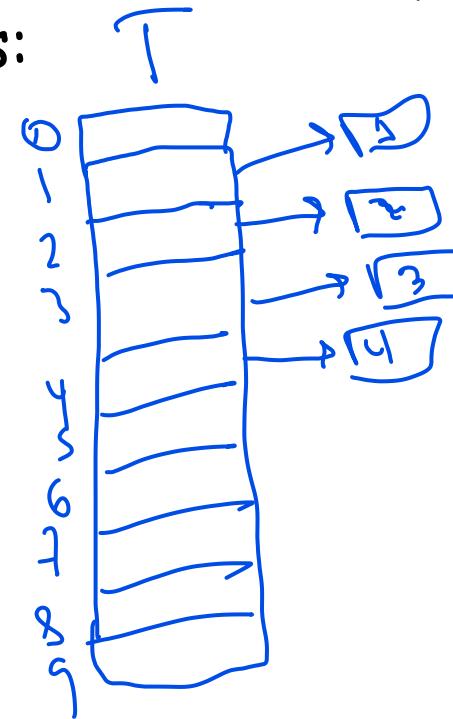
# Tablas Hash

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:

$$|U|=10$$

- 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

¿Qué pasa si  $|K| \ll |U|$ ?

# Tablas Hash

---

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

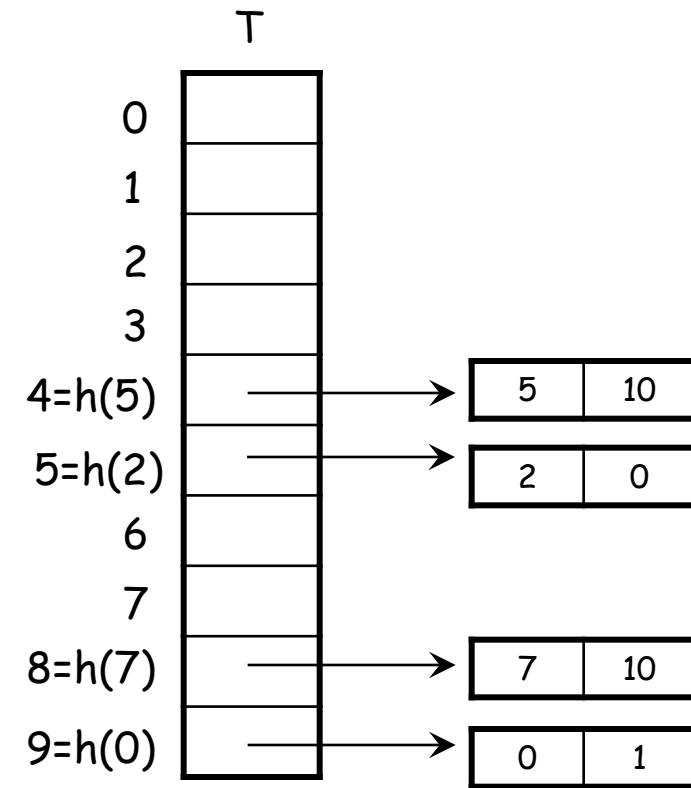
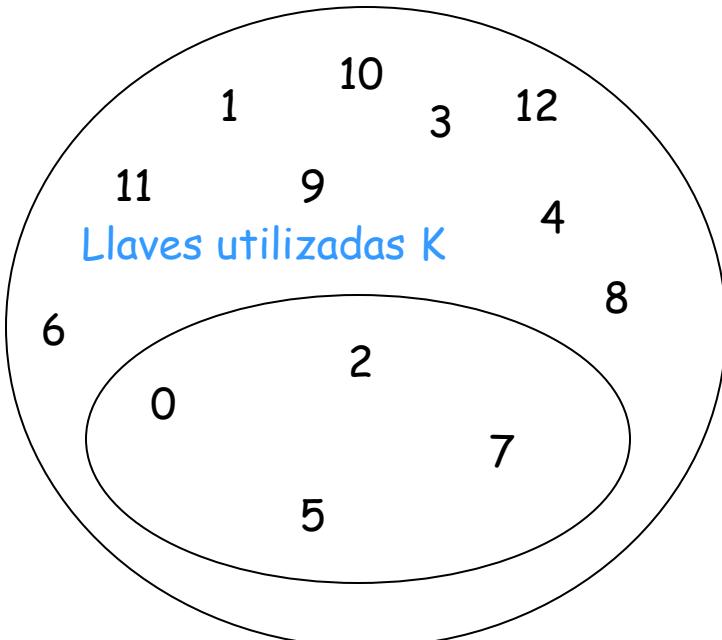
¿Qué pasa si  $|K| \ll |U|$ ?

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

Las **tablas hash** ofrecen una 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

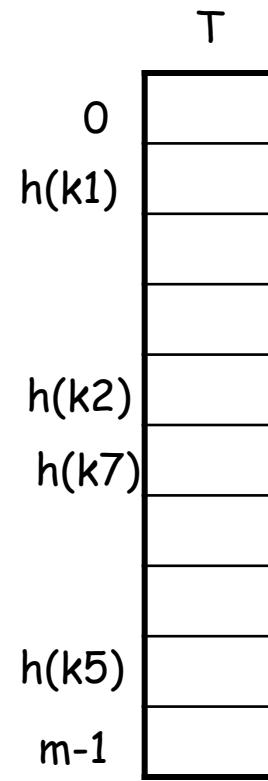
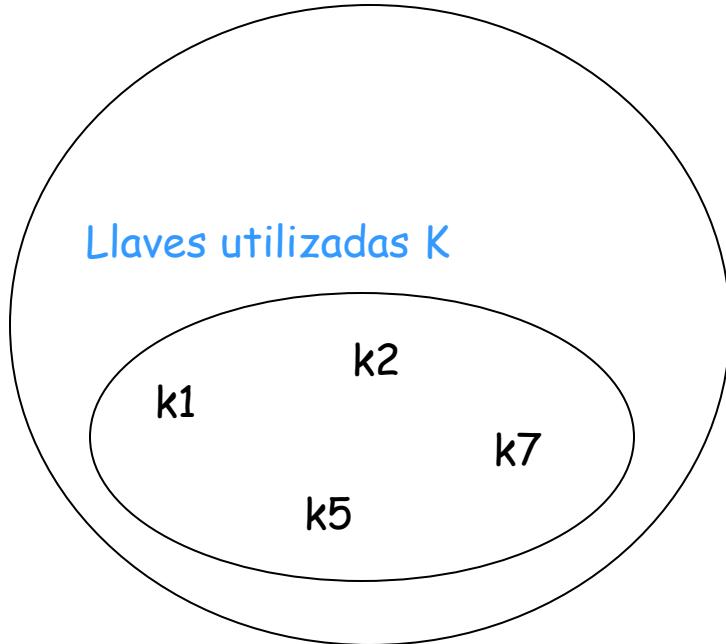
Universo de llaves U, ahora  $|U|>m$



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

# Tablas Hash

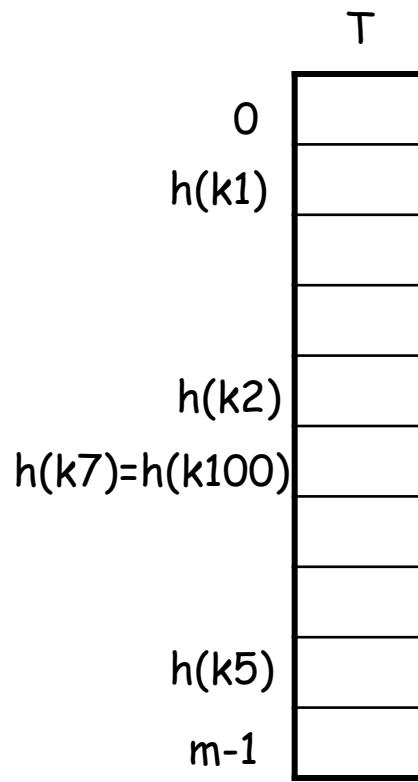
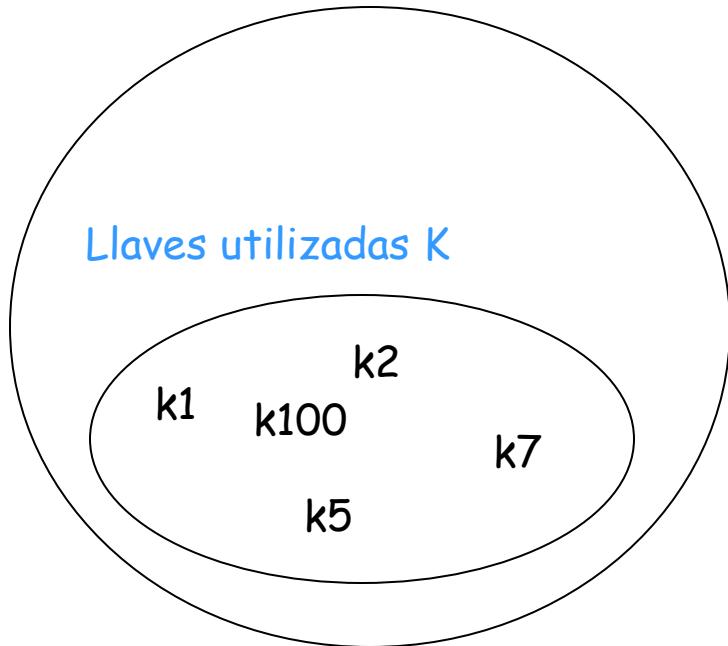
Universo de llaves U, ahora  $|U| > m$



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

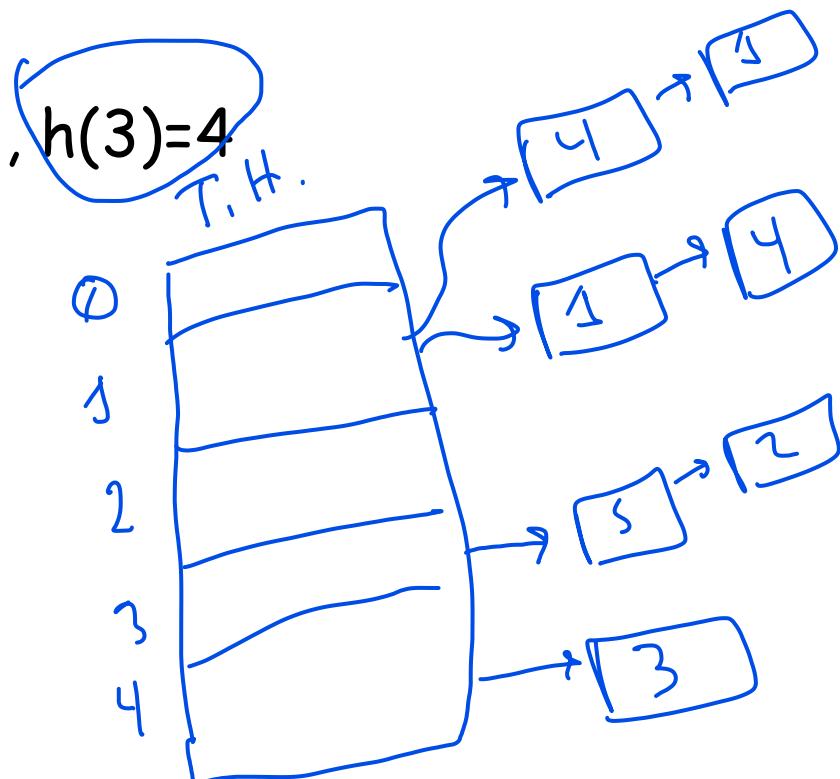
• HASH-INSERT( $T, 1$ )

HASH-INSERT( $T, 2$ )

HASH-INSERT( $T, 3$ )

HASH-INSERT( $T, 4$ )

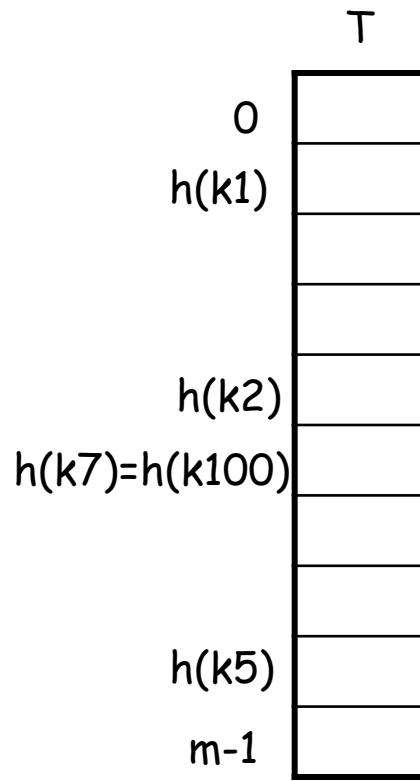
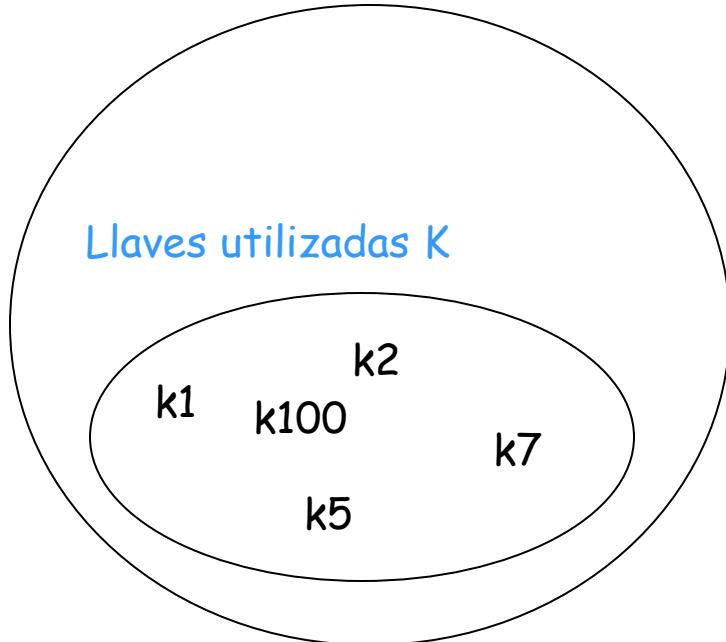
HASH-INSERT( $T, 5$ )



Muestre la tabla hash

# Tablas Hash

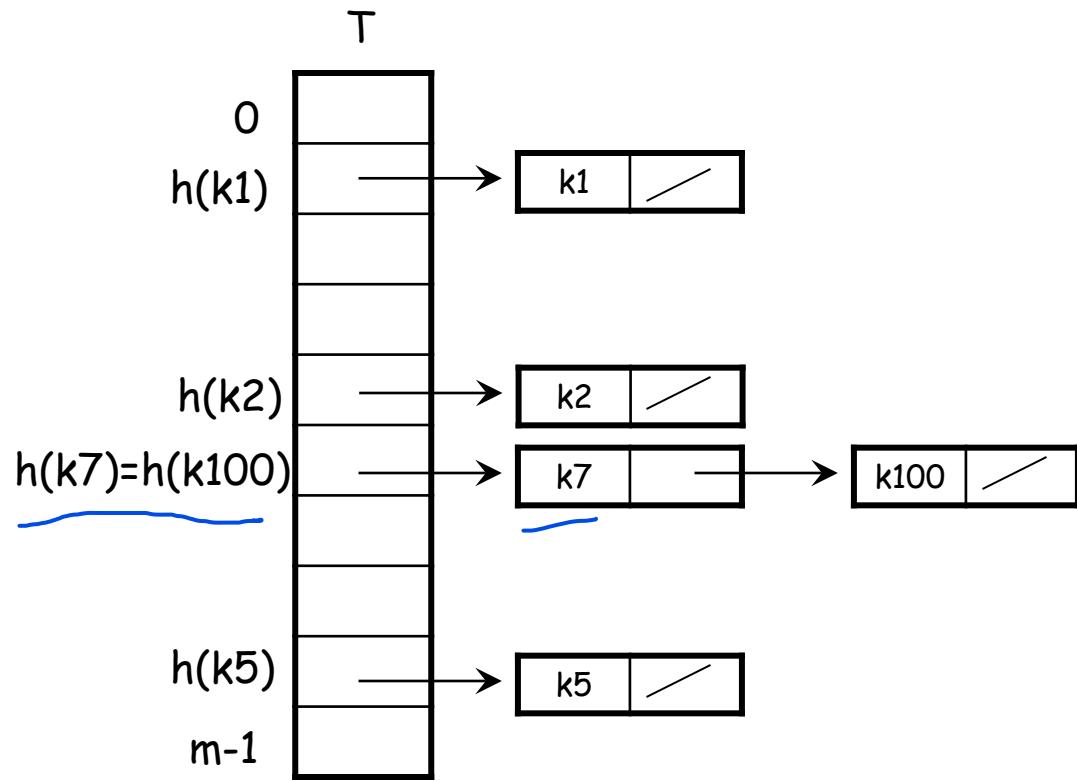
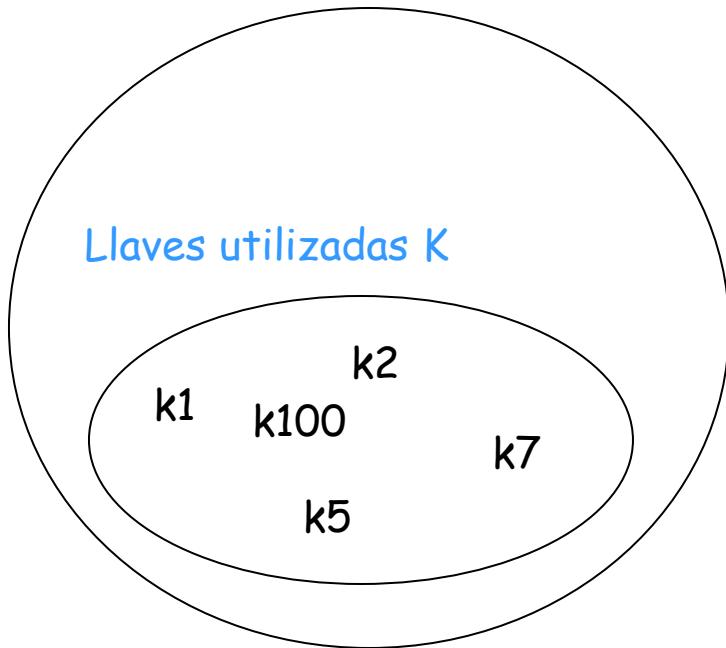
Universo de llaves U, ahora  $|U| > m$



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

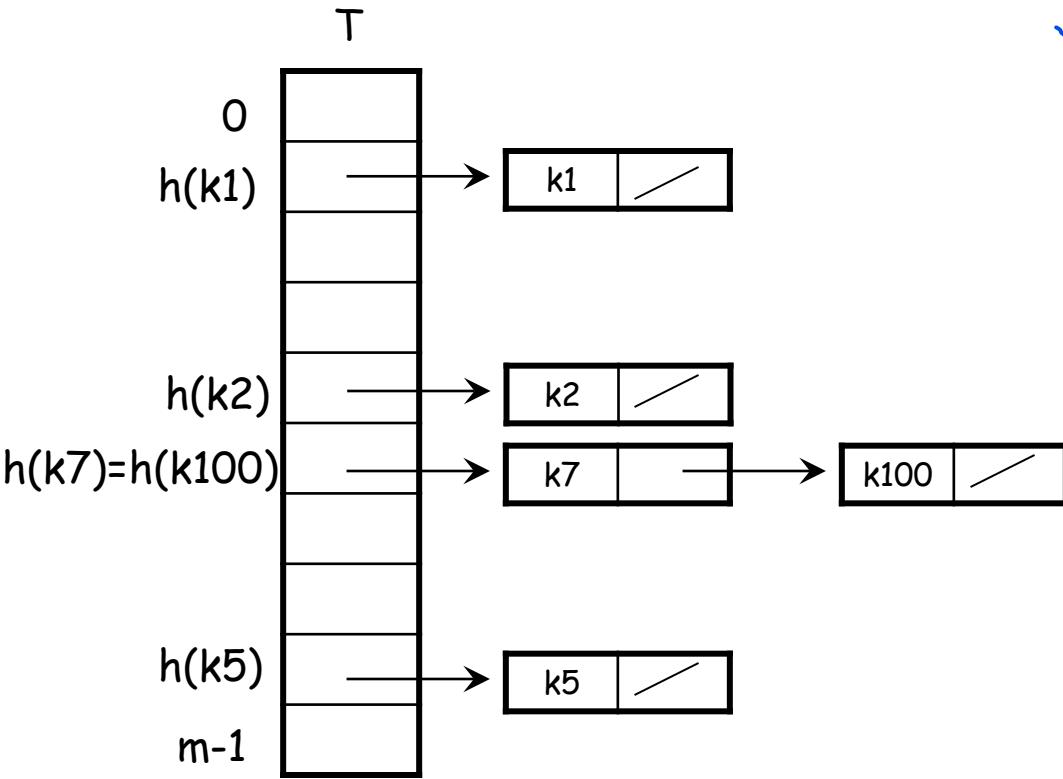
# Tablas Hash

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



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

# Tablas Hash

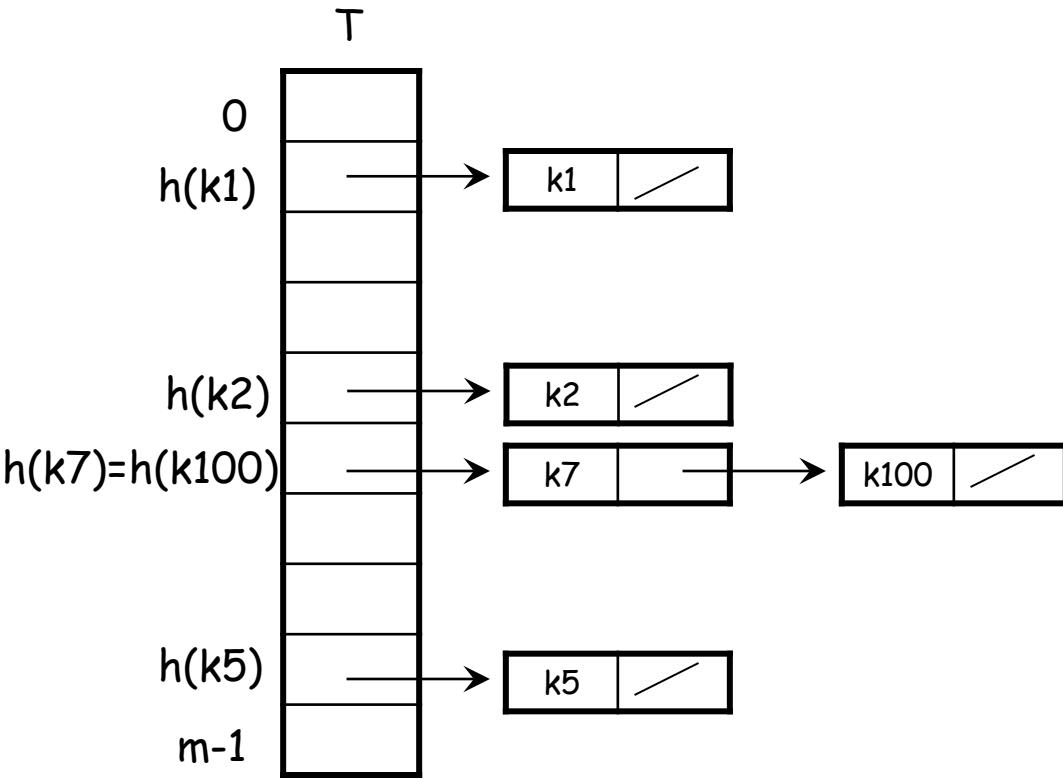


**CHAINED-HASH-INSERT( $T, x$ )**  
insertar  $x$  en la cabeza de la lista  $T[h(\text{key}(x))]$

**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  
 $T[h(\text{key}(x))]$

**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  $T[h(\text{key}(k))]$

Analice las complejidades de las operaciones

# Tablas Hash

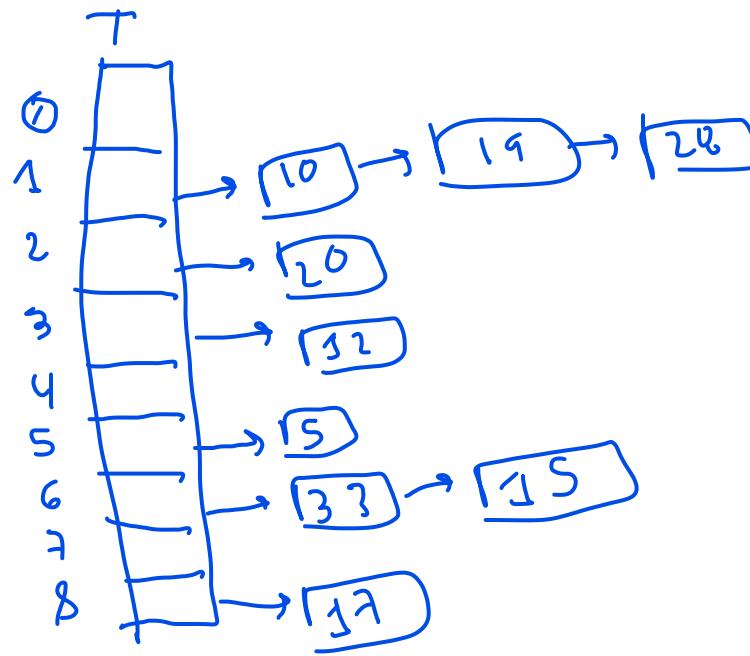
Muestra la tabla T después de insertar las llaves 5, 28, 19, 15, 20, 33, 12, 17, 10 en una tabla hash con 9 slots siendo la función

$$\underline{\text{hash } h(k) = k \bmod 9}$$

$$h(12) = 3$$

$$h(17) = 8$$

$$h(10) = 1$$



$$h(5) = 5 \bmod 9 \\ = 5$$

$$h(28) = 28 \bmod 9 \\ = 1$$

$$h(19) = 1$$

$$h(15) = 6$$

$$h(20) = 2$$

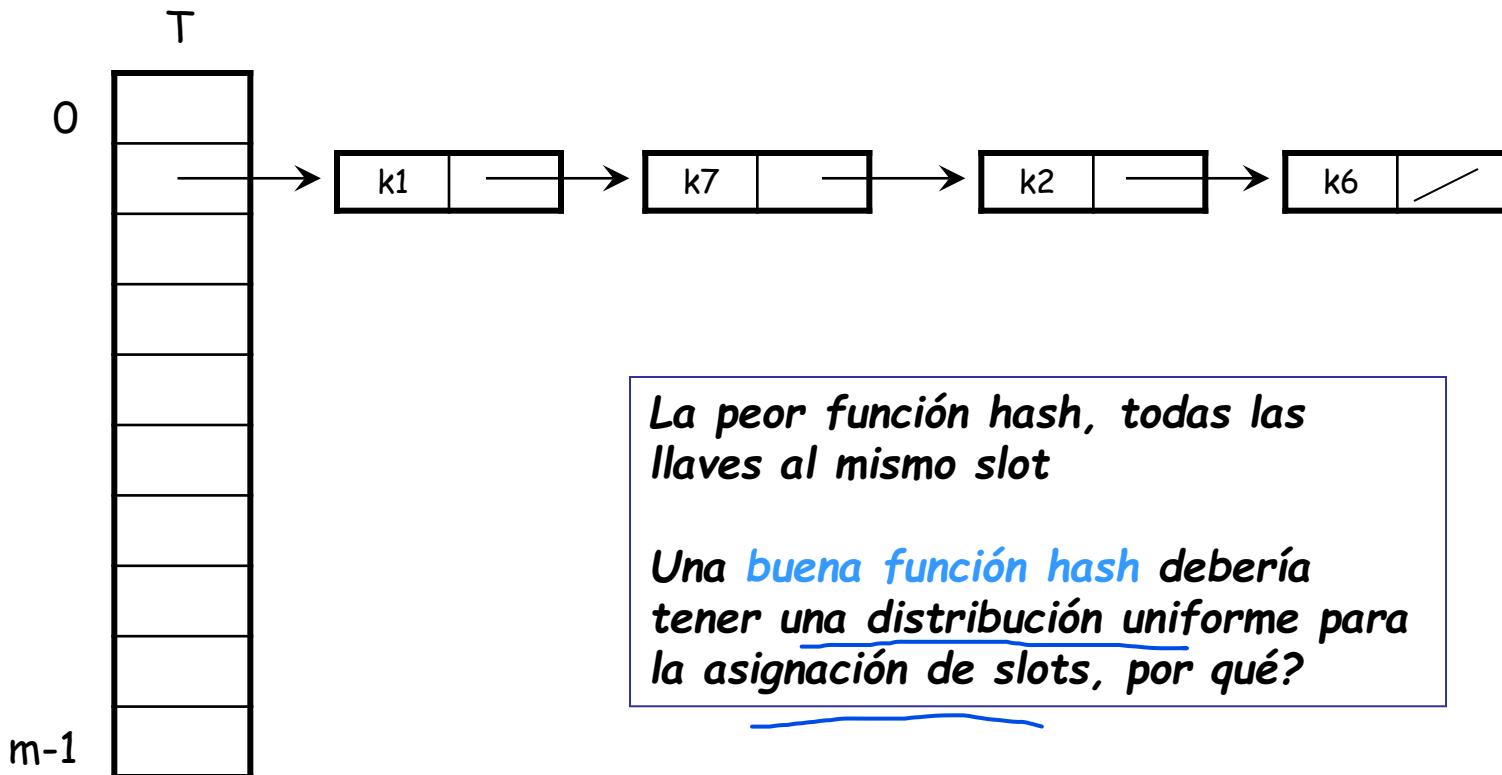
$$h(33) = 6$$

# Tablas Hash

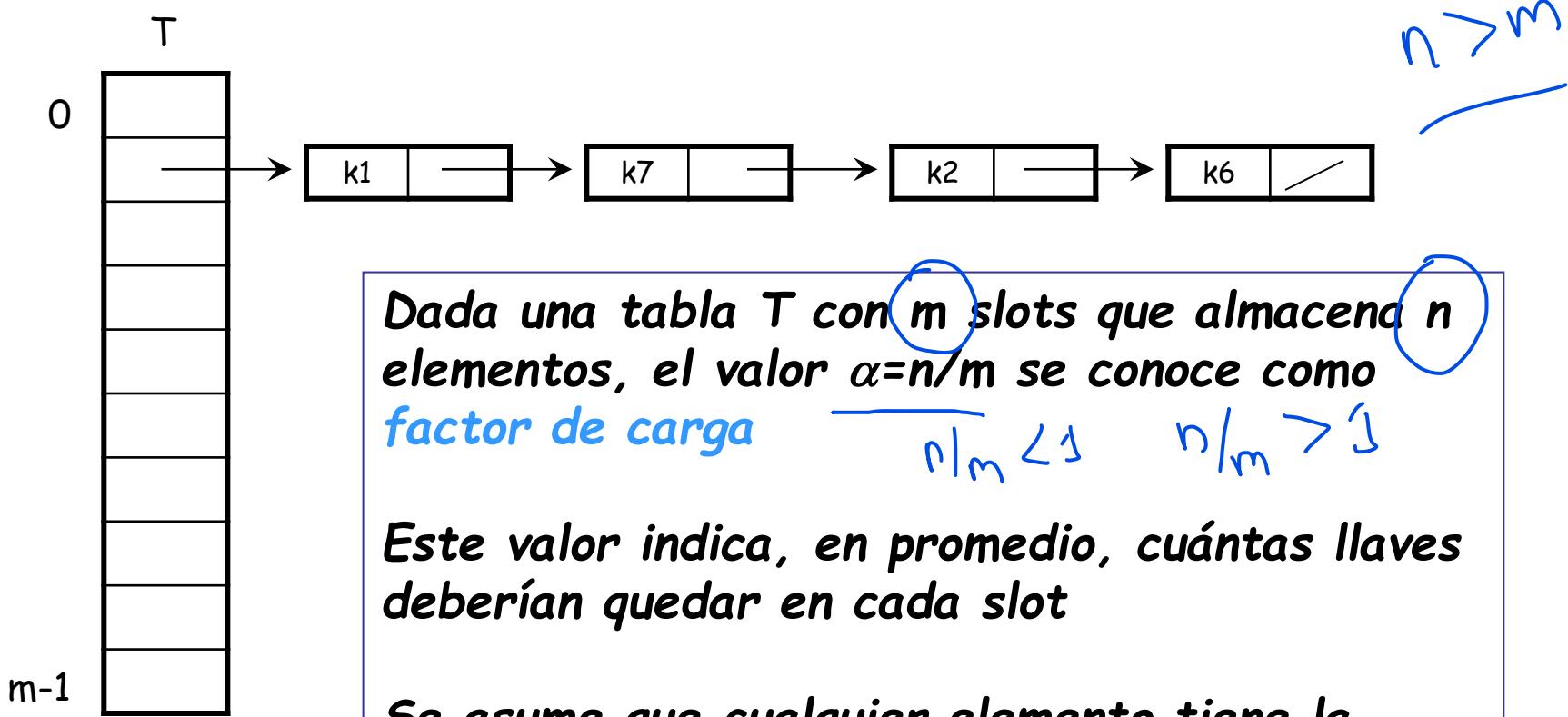
---

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

# Tablas Hash



# Tablas Hash



# Tablas Hash

---

## Teorema 1

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

## Teorema 2

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

# Tablas Hash

Una buena función hash:

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m}$$

, para  $j = 0, 1, \dots, m-1$

$$j = 0, 1, 2, 3$$

$$P(h(k)=0) = 1/4$$

$$P(h(k)=1) = 1/4$$

$$P(h(k)=2) = 1/4$$

$$P(h(k)=3) = 1/4$$

# 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

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

# 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  $0 \leq k < 1$ ,

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

# Tablas Hash

---

Una función hash

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

# Tablas Hash

---

Complete la tabla utilizando la fucion:

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

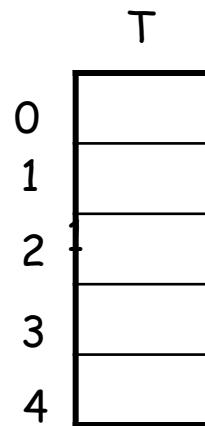
para almacenar las llaves

$$k_1=0.4$$

$$k_2=0.2$$

$$k_3=0.8$$

$$k_4=0.9$$



# Tablas Hash

---

Método división

Utiliza la función hash:

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

# Tablas Hash

---

Complete la tabla utilizando la función:

$$h(k) = k \bmod m, (m = 4)$$

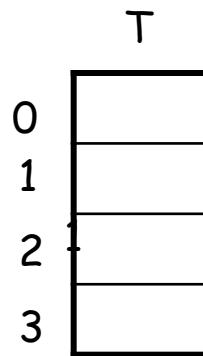
para almacenar las llaves

$$k_1 = 4$$

$$k_2 = 2$$

$$k_3 = 8$$

$$k_4 = 9$$



# Tablas Hash

---

Complete la tabla utilizando la función:

$$h(k) = k \bmod m, \quad (m = 4)$$

para almacenar las llaves

$$k_1=4$$

$$k_2=2$$

$$k_3=8$$

$$k_4=9$$

T
0
1
2
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  $k$ .

# Tablas Hash

---

Método multiplicación

Utiliza la función hash:

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

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  $10002$

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