# Fundamentos de análisis y diseño de algoritmos

**Tablas Hash**

- 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
```

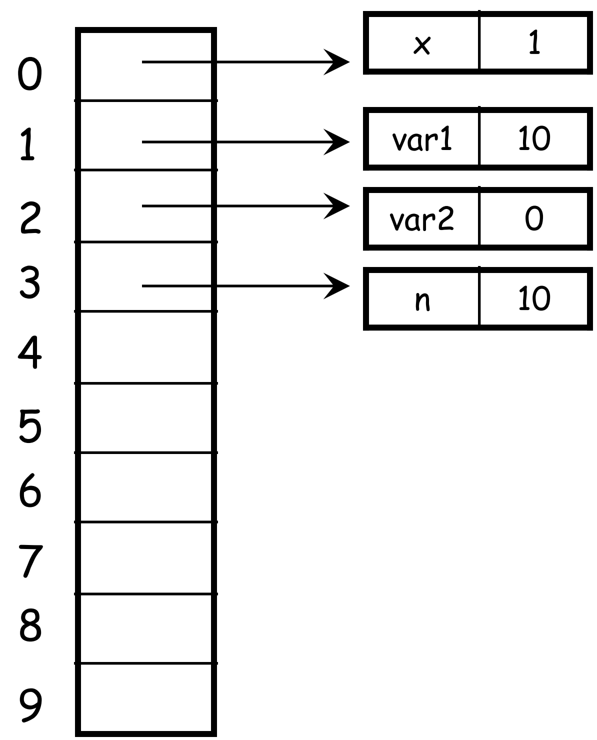
Los compiladores llevan una tabla de símbolos cuya llave son los identificadores de las variables

| | |
|---|---|
| x | 1 |
| var1 | 10 |
| var2 | 0 |
| n | 10 |
| | |
| | |
| | |
| | |
| | |
| | |

# Tablas Hash

```
programa1(int n)
 x←1
 var1←n
 var2←0
 while (x<n)
    var2←var2 + var1
    x←x+1
 print x
```

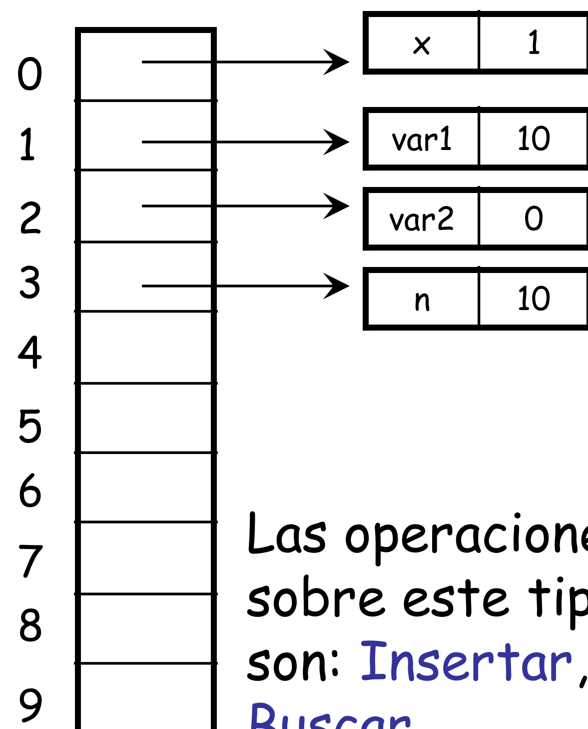
Los compiladores llevan una tabla de símbolos cuya llave son los identificadores de las variables



# Tablas Hash

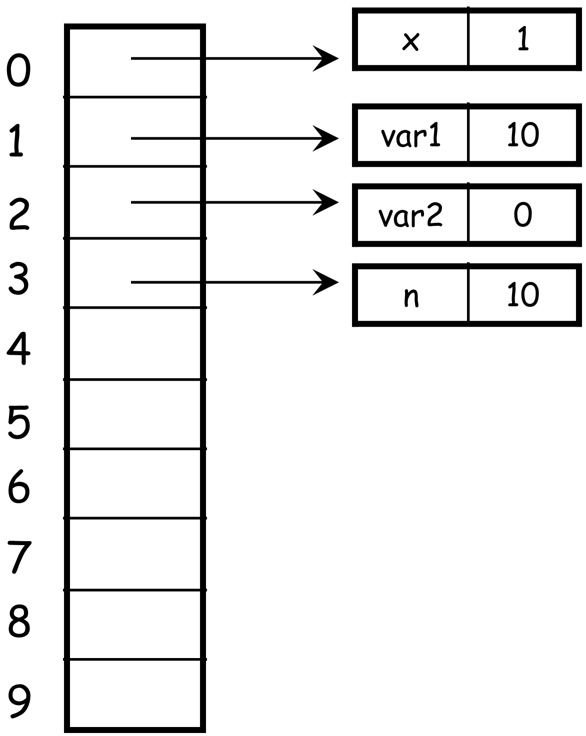
```
programa1(int n)
 x←1
 var1←n
 var2←0
 while (x<n)
    var2←var2 + var1
    x←x+1
 print x
```

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

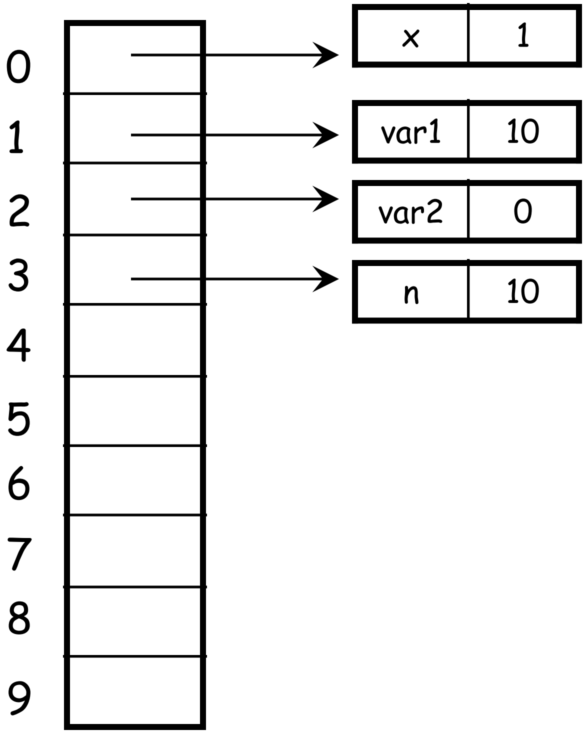



¿Qué tan costoso puede ser insertar un par (llave, valor) en la tabla?

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

$O(1)$

# Tablas Hash




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

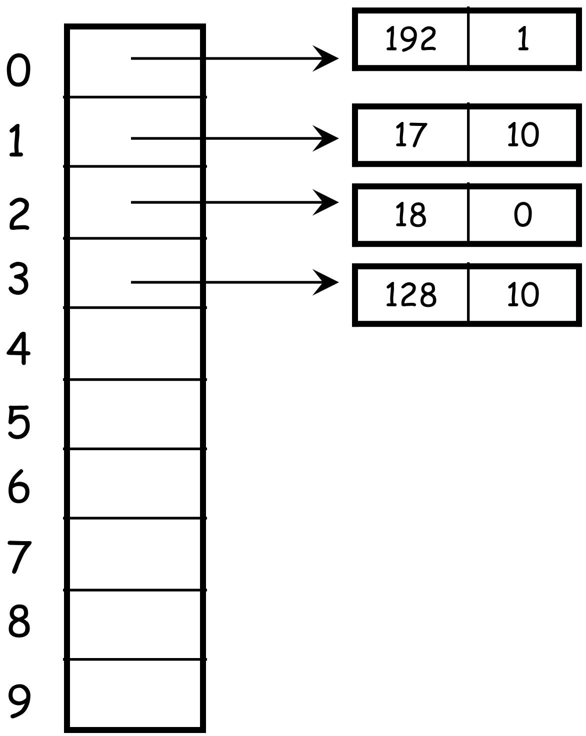
Si la clave permite calcular la posición es

O(1)

Si no

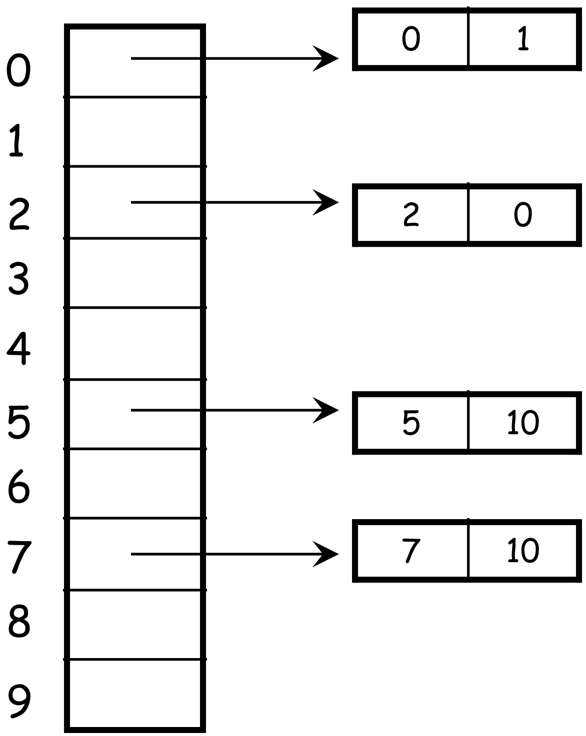
O(n)

# 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



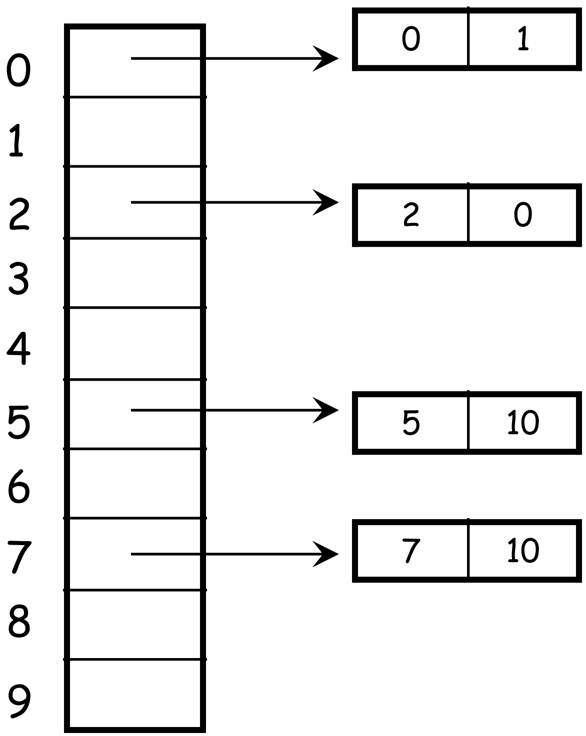

Una estrategia consiste en aprovechar que las llaves son numéricas y almacenar el par (llave, valor) en la posición "llave" de la tabla

Esta estrategia se conoce como
Tabla de direccionamiento directo

¿Cuál es el tiempo de búsqueda ahora?

O(1)

# Tablas Hash




•DIRECT-ADDRESS-INSERT(T,X)

T[key[x]]←x

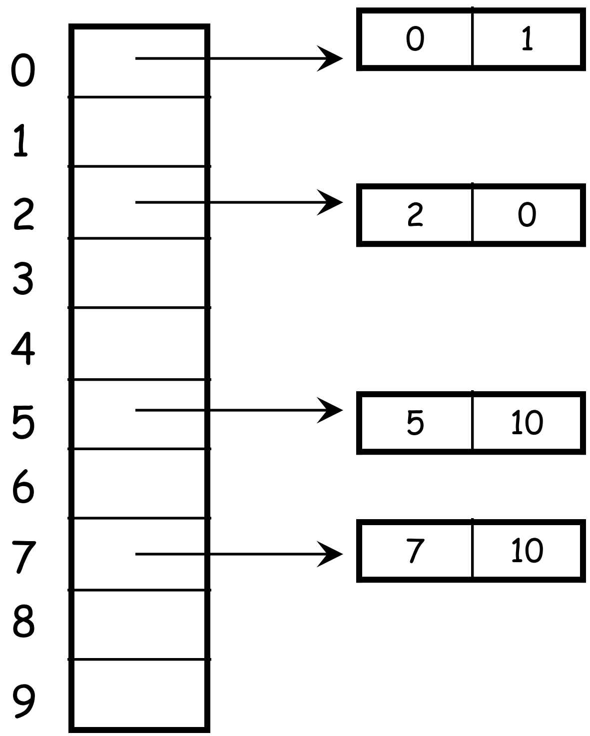
•DIRECT-ADDRESS-SEARCH(T,k)

return T[k]

•DIRECT-ADDRESS-DELETE(T,k)

T[key[x]]←nil

# Tablas Hash




- DIRECT-ADDRESS-INSERT(T,X)

  T[key[x]]←x

- DIRECT-ADDRESS-SEARCH(T,k)

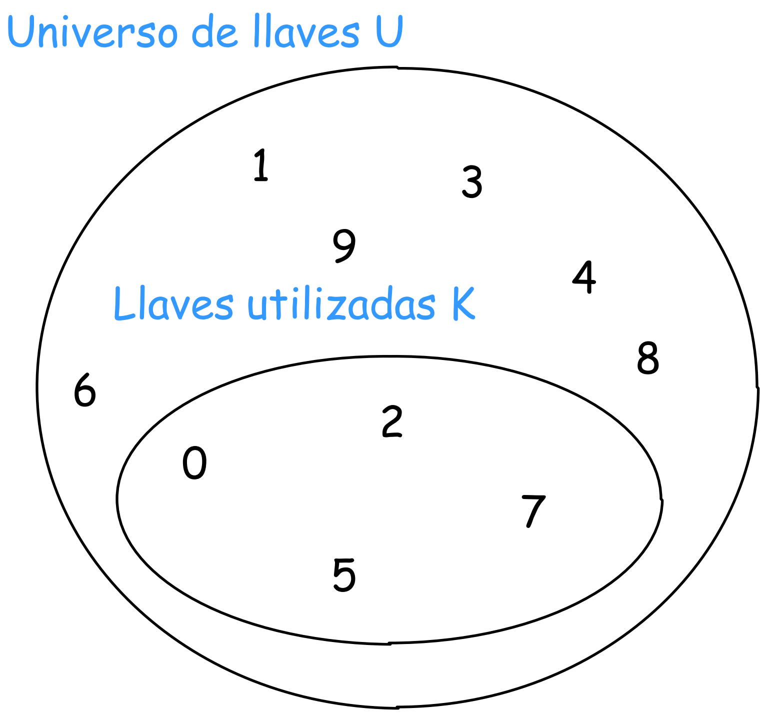
  return T[k]

- DIRECT-ADDRESS-DELETE(T,k)

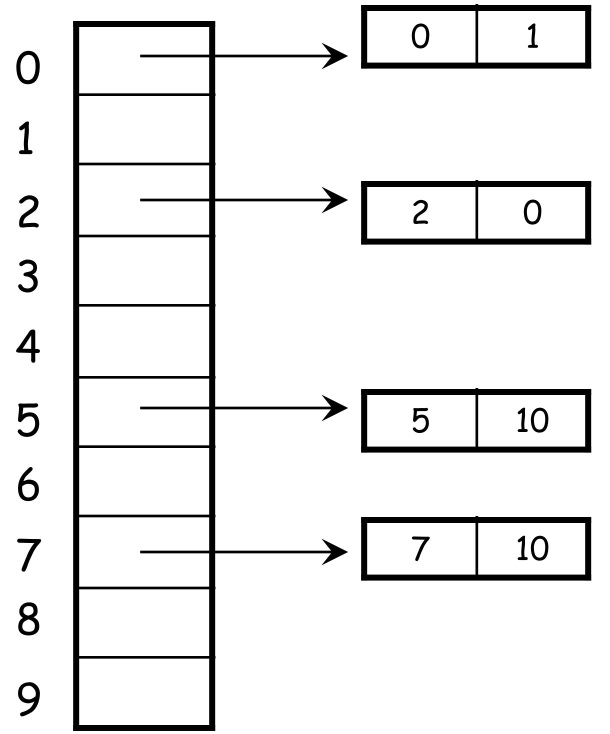
  T[key[x]]←nil

- Todas estas operaciones toman tiempo constante O(1)

# Tablas Hash

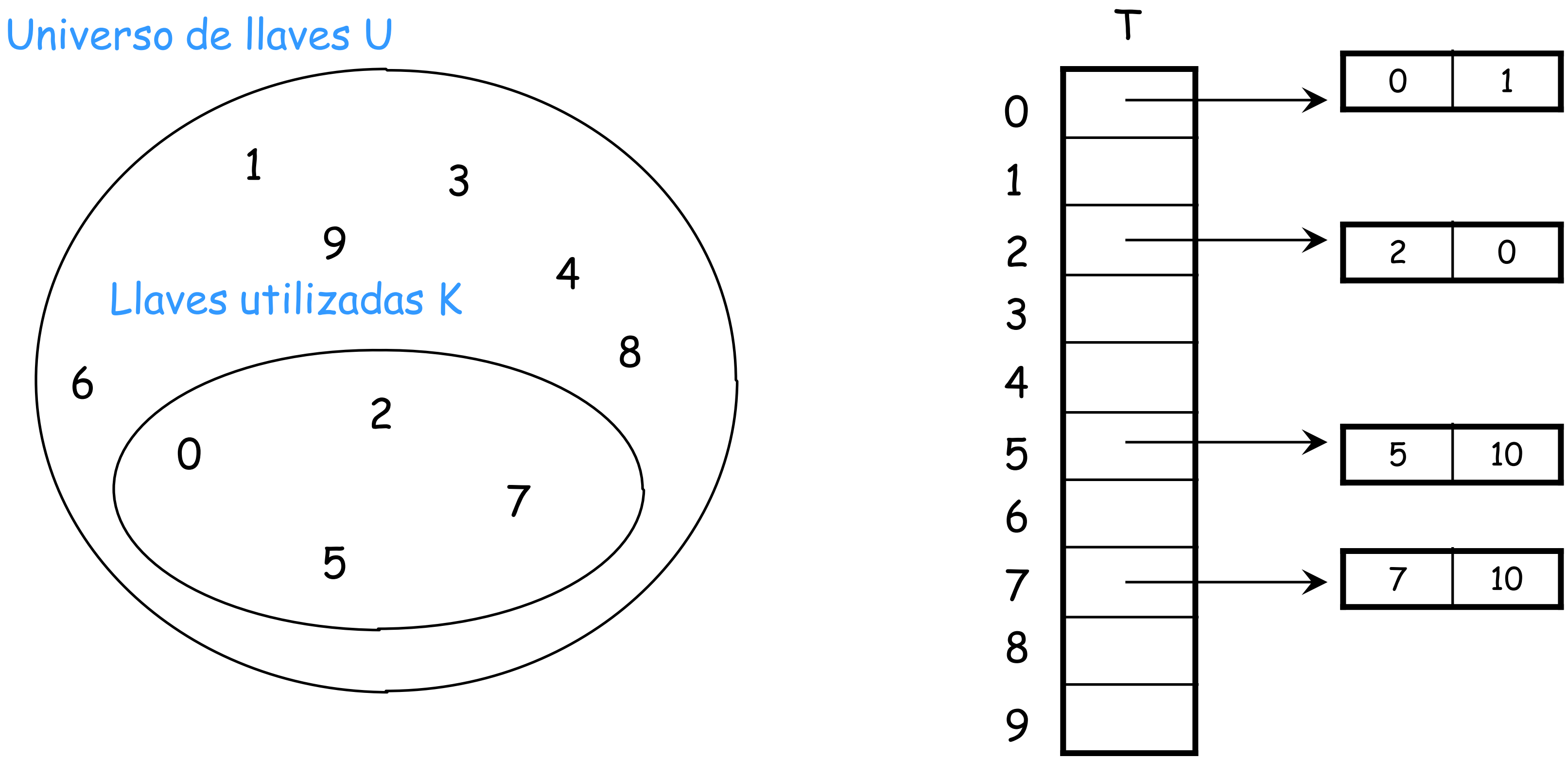


Universo de llaves U
1
3
9
4
Llaves utilizadas K
8
6
2
0
7
5

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

# Tablas Hash




U={0, 1, ..., m-1}, donde |U|=m
La tabla de direccionamiento directo T, se puede ver como un arreglo T[0, ..., 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,...,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

¿Qué pasa si $|K| \ll |U|$?

# Tablas Hash

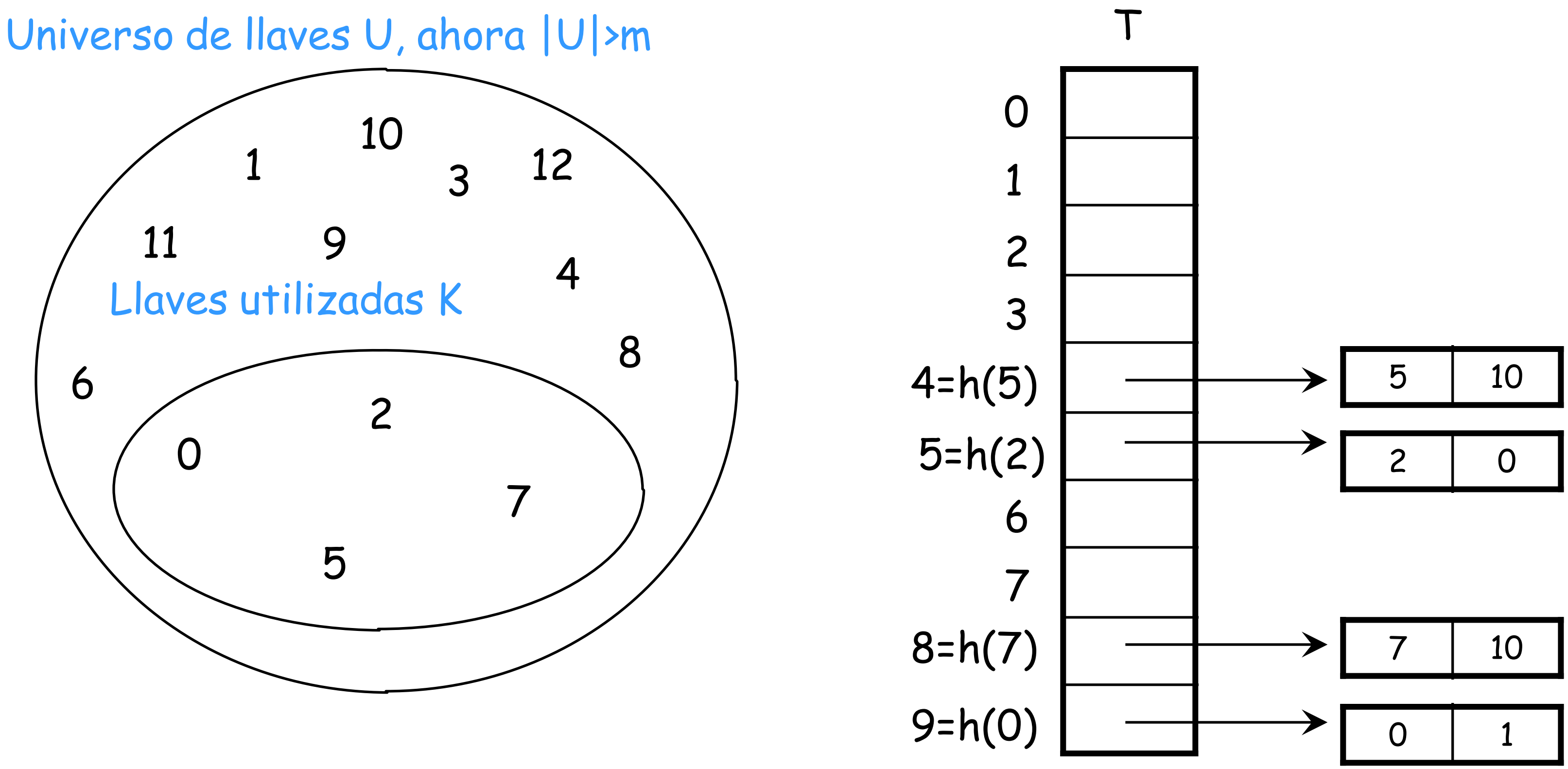
Considere el caso en el que tuviese que almacenar 1000 datos utilizando una tabla de direccionamiento directo

¿Qué pasa si $|K| \ll |U|$?

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

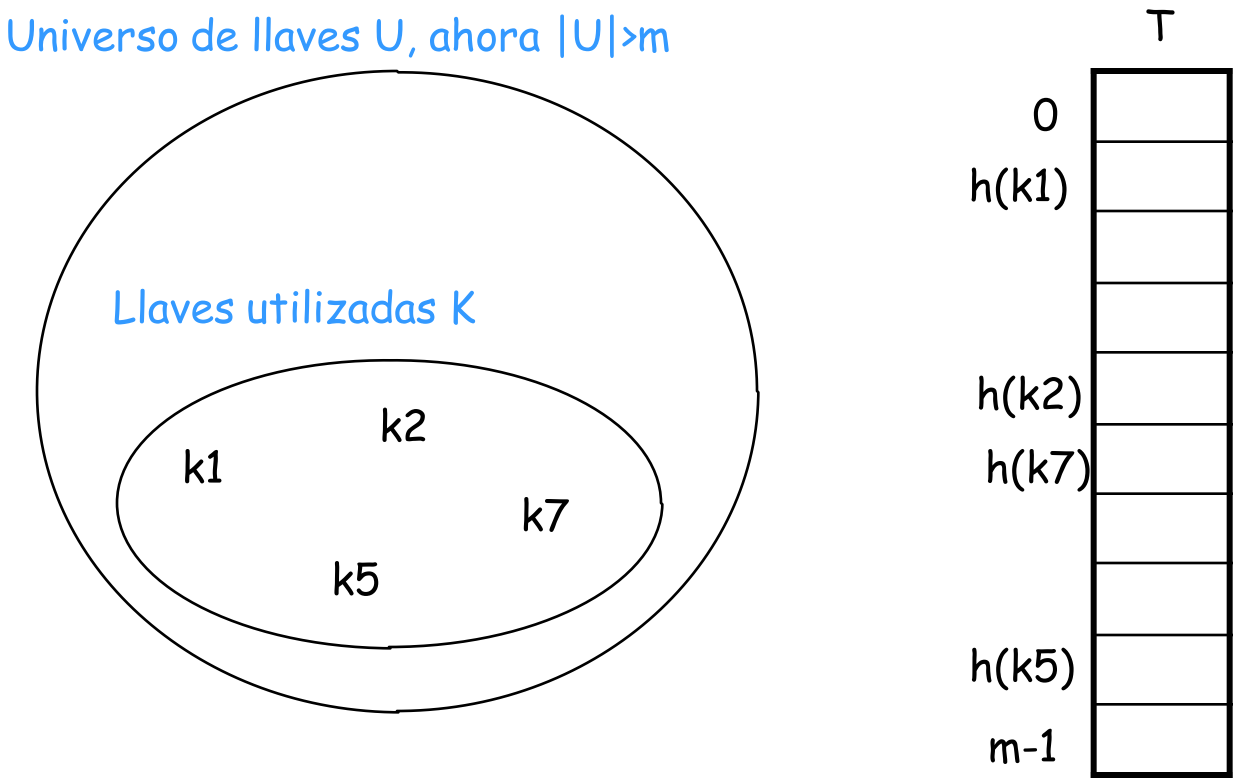
Las tablas hash ofrecen una mecanismo para asignar la posición de almacenamiento para las llaves, de tal forma que los requerimientos de memoria pueden ser de $O(|K|)$

# Tablas Hash



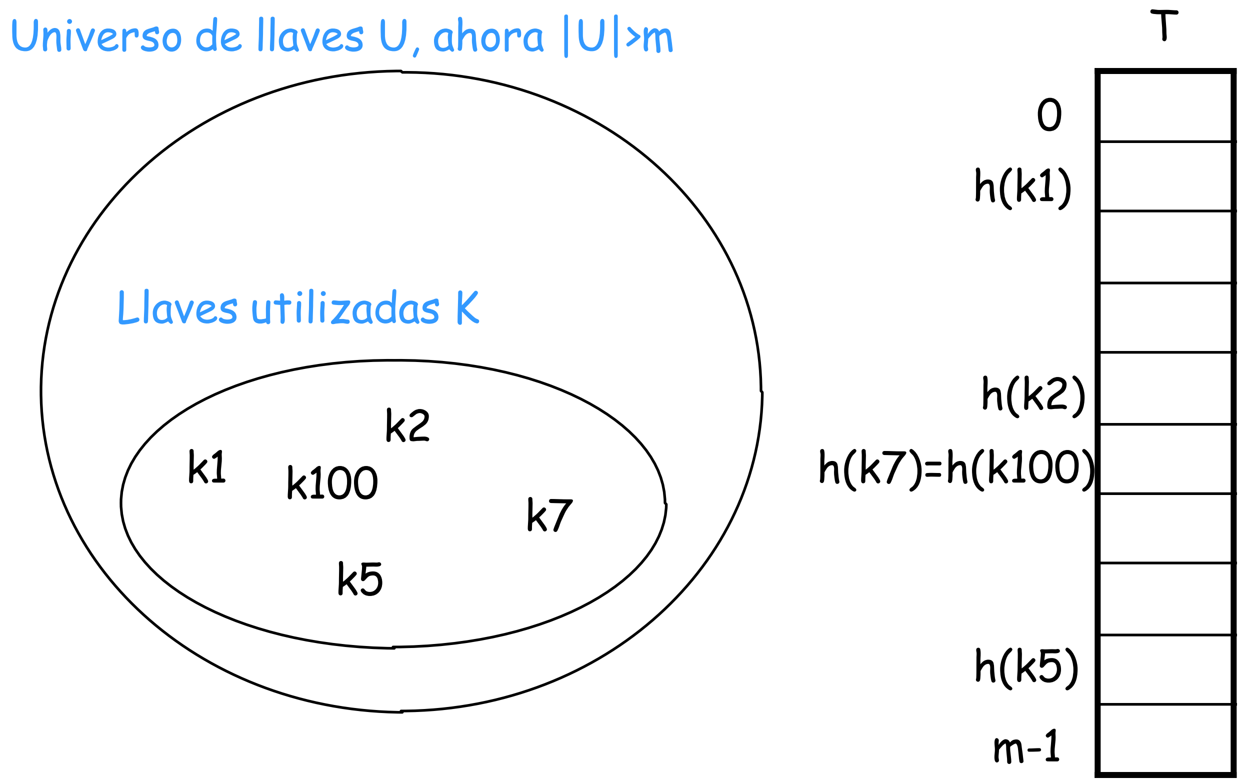

Las tablas hash utilizan una función h: U→{0, 1, ..., m-1}

# Tablas Hash




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

# Tablas Hash



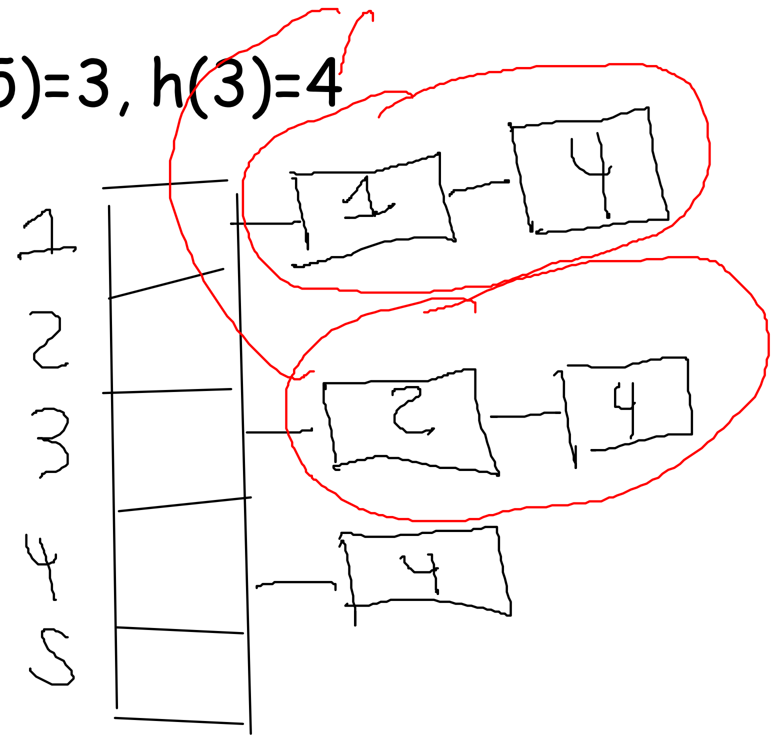

Como |U|>m, pueden existir dos llaves en el mismo slot, esto se llama una **colisión**

# Tablas Hash

Tabla hash (suponga que key(x)=x y m=5)

colisión

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



HASH-INSERT(T,1)

HASH-INSERT(T,2)

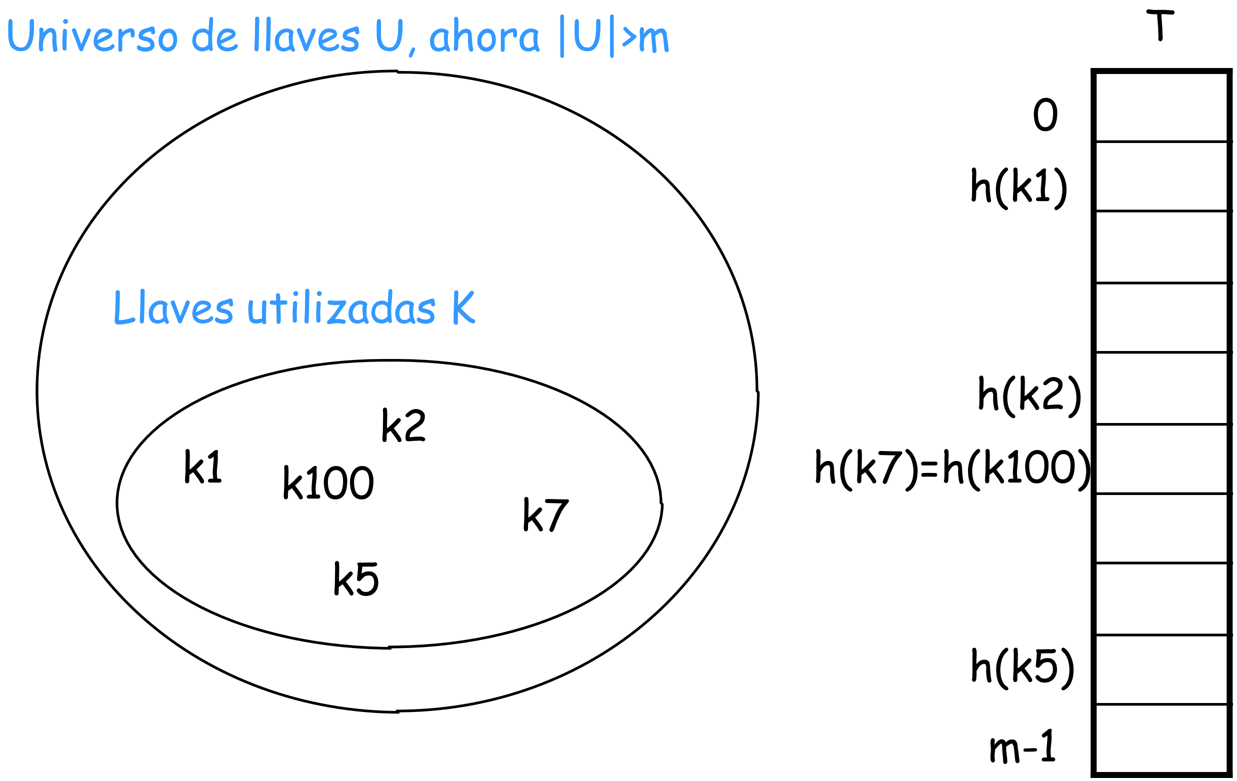
HASH-INSERT(T,3)

HASH-INSERT(T,4)

HASH-INSERT(T,5)

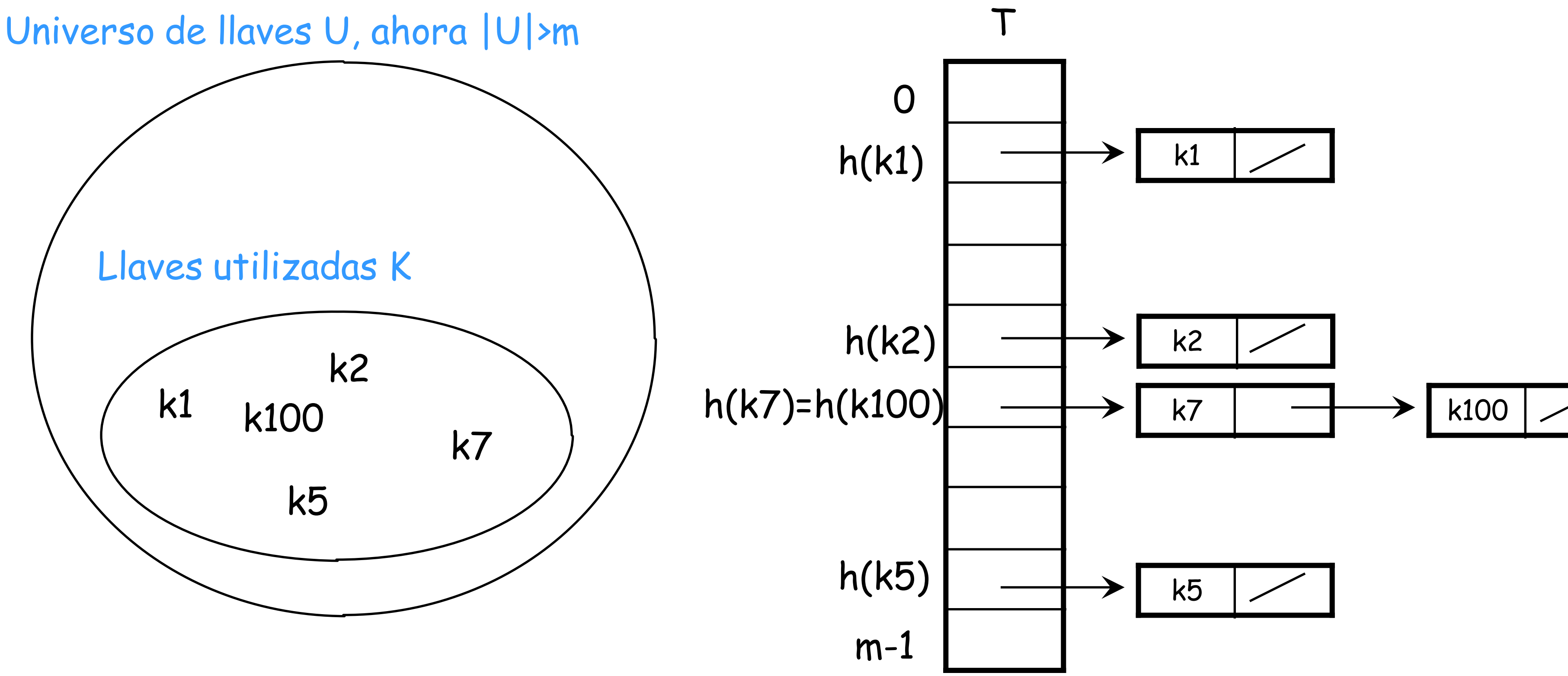
Muestre la tabla hash

# Tablas Hash




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

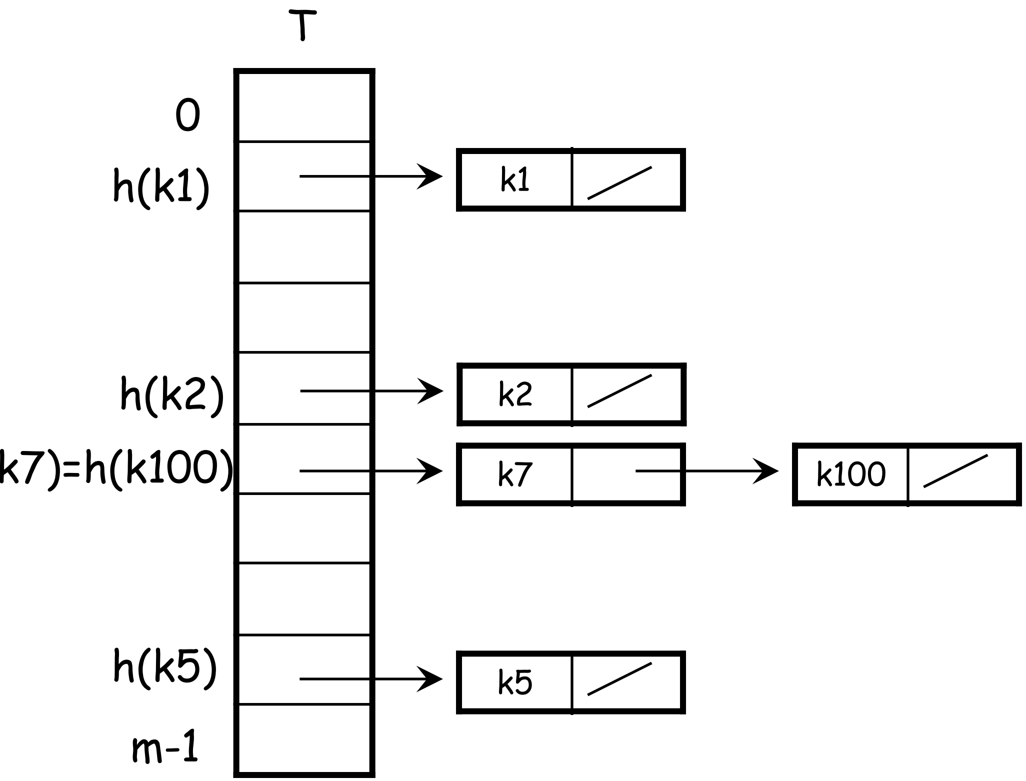
# Tablas Hash




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

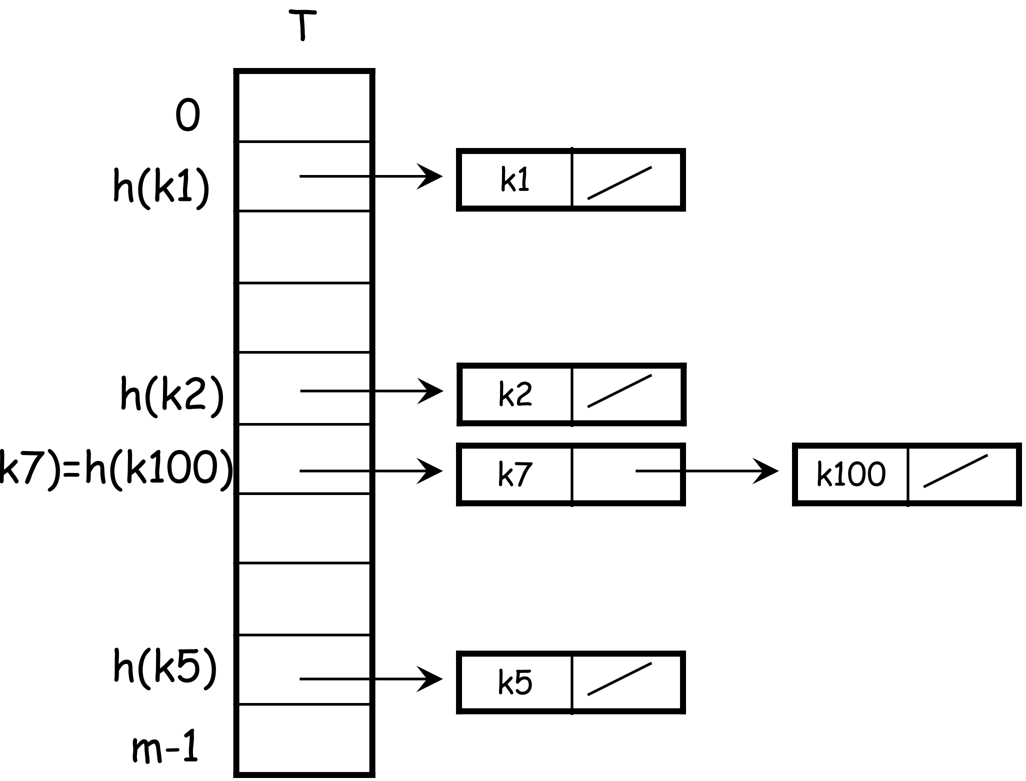



***CHAINED-HASH-INSERT(T,x)***
***insertar x en la cabeza de la lista T[h(key(x))]***

***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))]***

# Tablas Hash




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

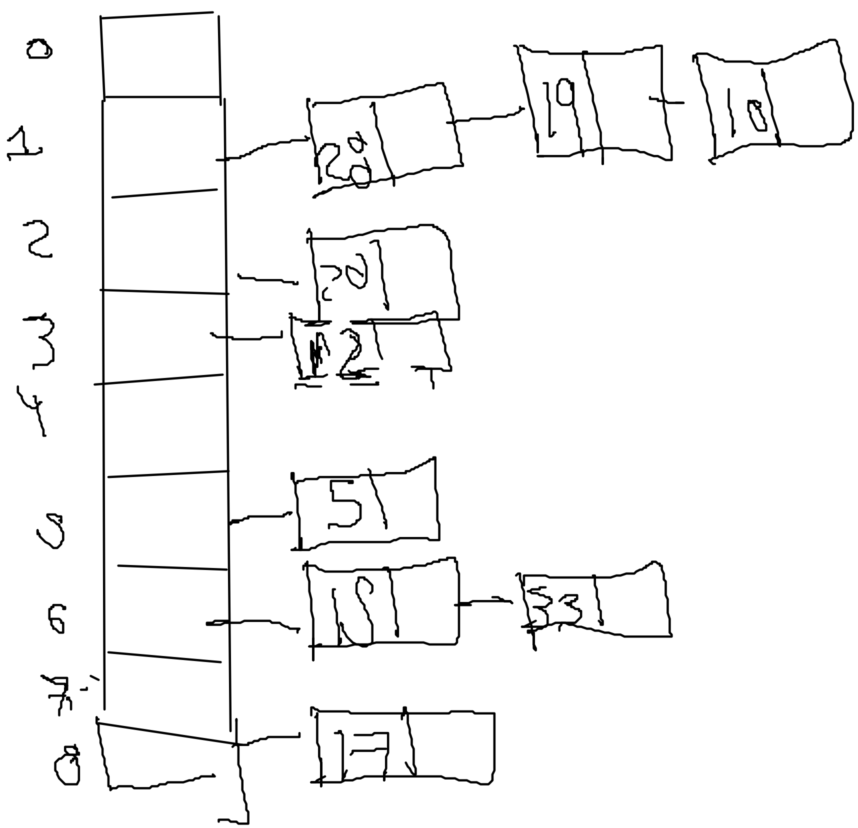
***CHAINED-HASH-SEARCH(T,k)***
***buscar por un elemento con llave k en la lista T[h(key(k))]***

***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 mod 9**



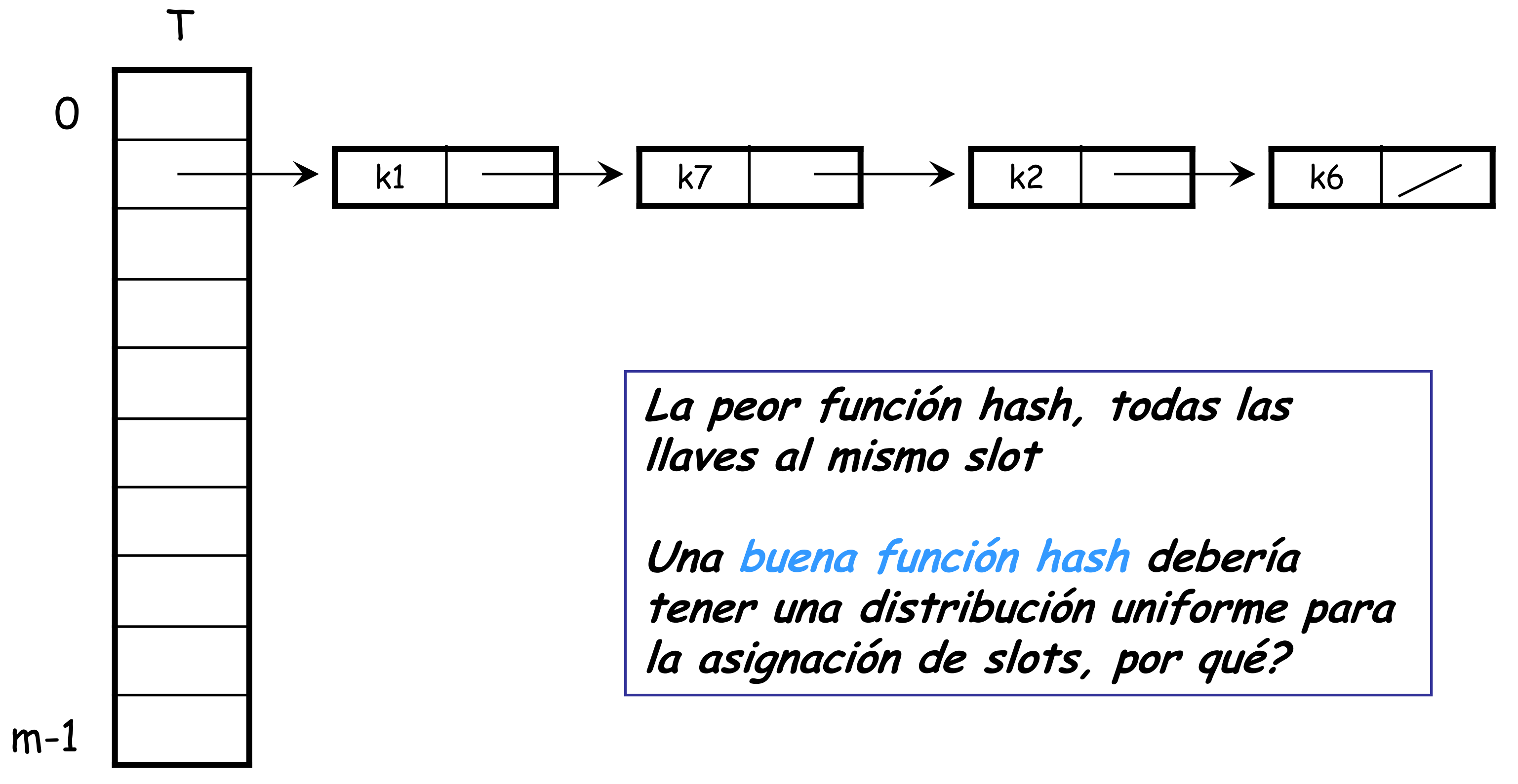

5 mod 9 = 5

28 mod 9 = 1

# Tablas Hash

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

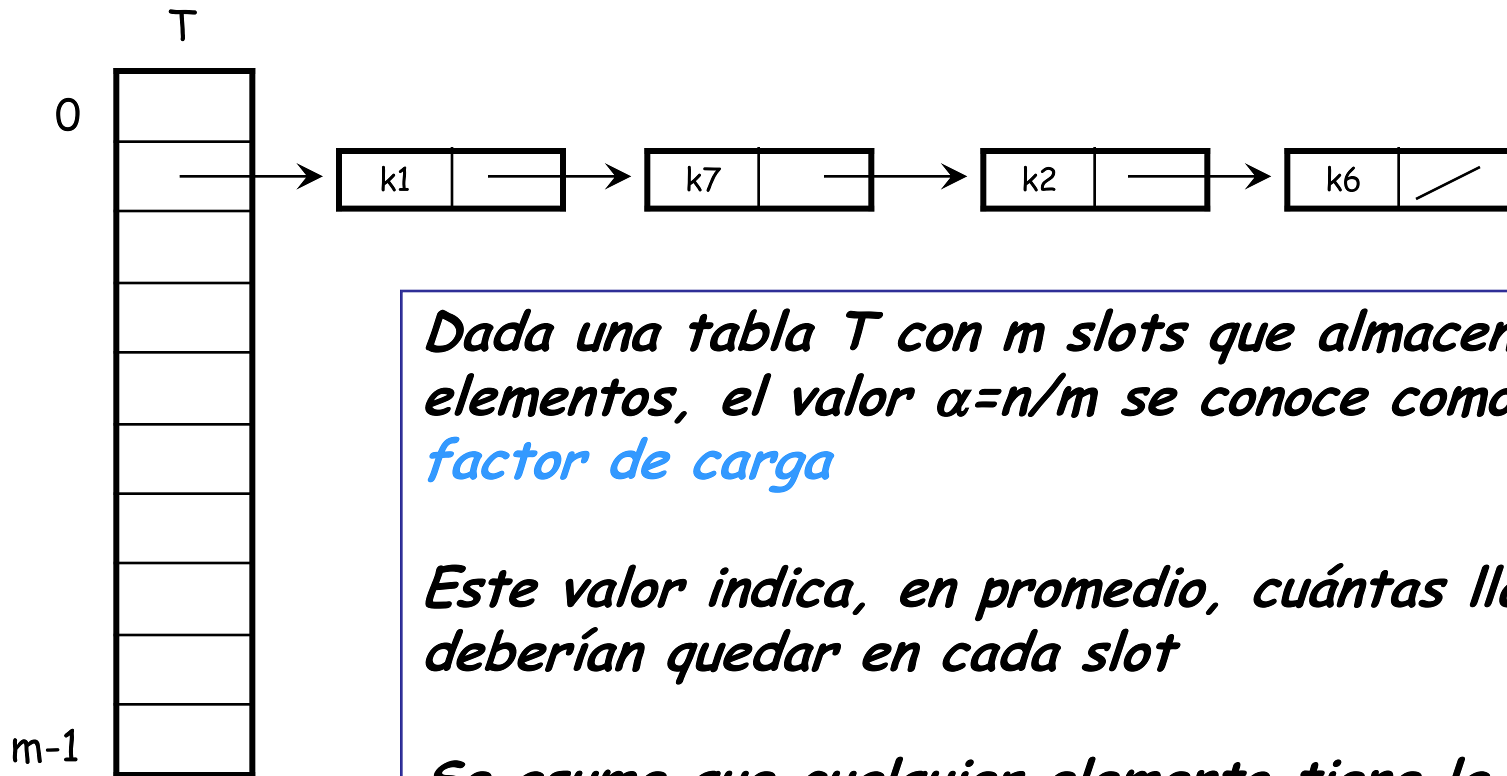
# Tablas Hash




*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

llave lista encadenada $\alpha = n/m$

## 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:

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

# Tablas Hash

Es común tener en un programa nombres de identificadores que son similares, var1, var2, por ejemplo. Una buena función hash deberías 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 $0 \leq k < 1$,

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

# Tablas Hash

## Una función hash

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

# Tablas Hash

Complete la tabla utilizando la fucion:

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

5

para almacenar las llaves

k1=0.4 ⟵ 2

k2=1.2 ⟵ 6

k3=1.8 ⟵ 9

k4=0.9 ⟵ 5

| | T |
|---|---|
| 0 | |
| 1 | |
| 2 | 1 |
| 3 | |
| 4 | |

# Tablas Hash

## Método división

Utiliza la función hash:

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

# Tablas Hash

Complete la tabla utilizando la función:

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

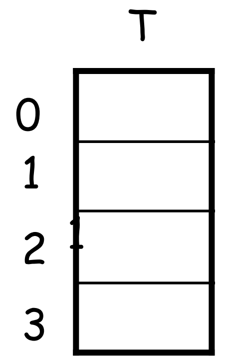
para almacenar las llaves

$k1=4$

$k2=2$

$k3=8$

$k4=9$



# Tablas Hash

Complete la tabla utilizando la función:

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

para almacenar las llaves

k1=4

k2=2

k3=8

k4=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.

| | T |
|---|---|
| 0 | K1,k3 |
| 1 | k4 |
| 2 | K2 |
| 3 | |

# 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

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

$h(k)=\lfloor 10000(618,03 \bmod 1)\rfloor$

$h(k)=\lfloor 10000 \times 0,03 \rfloor = 300$

$h(k)=\lfloor 10000(123400 \times 0.61803 \bmod 1)\rfloor = 9020$

$h(k)=\lfloor 10000(40321 \times 0.61803 \bmod 1)\rfloor = 5976$

$h(k)=\lfloor 10000(10002 \times 0.61803 \bmod 1)\rfloor = 5360$