

**WUOLAH**



Phantone

[www.wuolah.com/student/Phantone](http://www.wuolah.com/student/Phantone)

 705

**ED-T4-5.pdf**

Apuntes Teoría



2º Estructuras de Datos



Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada



**El más PRO del lugar  
puedes ser Tú.**

¿Quieres eliminar toda la publi  
de tus apuntes?

**Hazte PRO**

4,95€/mes

**W**

**WUOLAH**



**El más PRO del lugar puedes ser Tú.**



**¿Quieres eliminar toda la publi de tus apuntes?**



**¡Fuera Publi!**  
Concéntrate al máximo



**Apuntes a full.**  
Sin publi y sin gastar coins

Para los amantes de la inmediatez, para los que no desperdician ni un solo segundo de su tiempo o para los que dejan todo para el último día.



**Quiero ser PRO**

4,95 / mes

# TEMA 4.5: TABLAS HASH

## Hashing

El Hashing no opera mediante la comparación entre valores clave, sino buscando una función,  $h(K)$ , que nos dé la localización exacta de la clave  $K$  en la estructura de datos en la que estén almacenadas las claves.

¿Son fáciles de buscar estas funciones? NO.

Si buscamos  $\forall i \neq j \Rightarrow h(i) \neq h(j)$

Ejemplo:

Tabla tamaño 40 y 30 claves:

$$\left\{ \begin{array}{l} 40^{30} \approx 1.15 \cdot 10^{48} \text{ posibles funciones} \\ 40! / 10! \approx 2.25 \cdot 10^{41} \text{ no generan duplicados} \end{array} \right.$$

! Solo nos servirían 2 de cada 10 millones!

Los registros de datos a los que corresponden las claves suelen estar almacenados en un fichero de un sistema de almacenamiento externo.

La tabla Hash actúa a modo de índice.

Nuestro objetivo será:

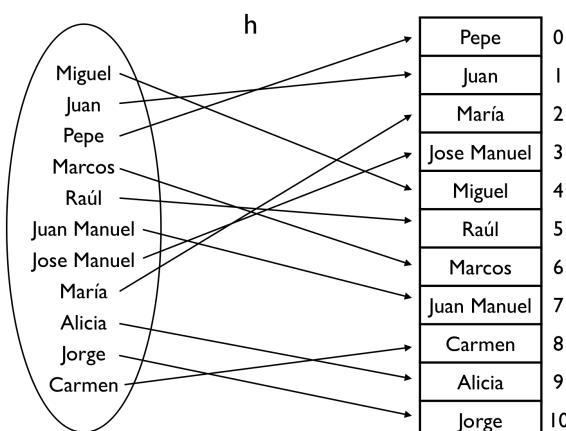
- Encontrar funciones  $h$  que generen el menor número posible de colisiones.
- Diseñar métodos de resolución de colisiones, cuando éstas se produzcan.

## Tablas Hash

Una tabla Hash es un contenedor asociativo (tipo diccionario) que permite un almacenamiento y posterior recuperación eficientes de elementos, denominados valores, a partir de otros objetos, llamados claves.

La forma ideal de realizar la búsqueda de un elemento en un contenedor sería aplicar una función matemática sobre el dato y que ésta devolviera directamente el lugar en el que se encuentra. Esto sería  $O(1)$ .

A esta función se le llama función Hash.



Clave

Tabla Hash		
0	23121	i
1	24576	n
2	...	
3	23396	n-1
4	...	
5	22563	1
6	...	
n-1	23396	...
n	24576	.....

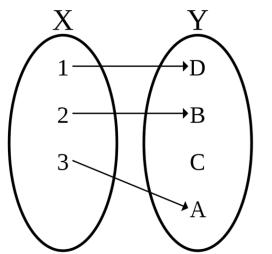
Clave 23396  
 $h(23396) = i$

Posición en el fichero

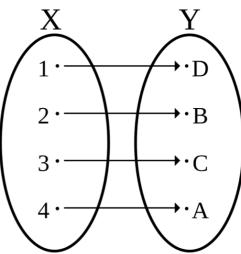
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

La función hash debería ser inyectiva. El problema es que encontrar una función así no es sencillo.

Cuando tenemos una función Hash biyectiva decimos que tenemos una función hash perfecta. El conjunto de datos debe ser fijo y predeterminado.



Función inyectiva

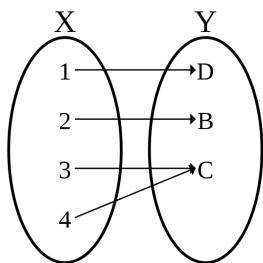


Función biyectiva

Para el resto de casos tendremos funciones sobreyectivas, esto es, para algunas parejas de claves diferentes obtendremos el mismo valor. En este caso se producen colisiones en el valor de la función Hash.

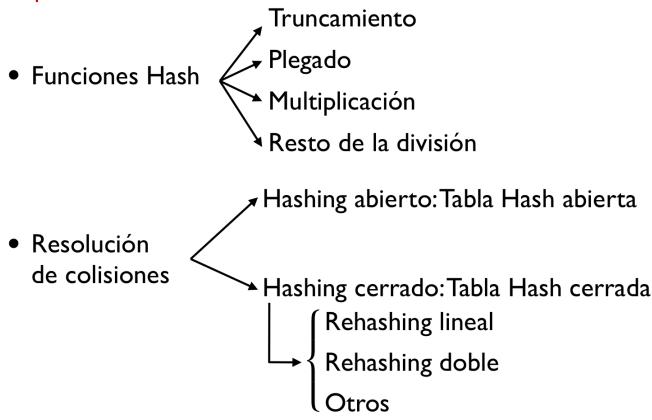
**Colisión:** dadas dos claves distintas,  $K_1$  y  $K_2$ , si  $h(K_1) = h(K_2)$  se produce una colisión.

Dependiendo de cómo resolvamos esas colisiones tendremos hashing abierto o cerrado.



Función sobreyectiva

### Esquema



Ejemplo (sin colisiones):

$M = 11$   
(primo más cercano a 8)

	Código	Pos
0		X
1		X
2		X
3		X
4		X
5		X
6		X
7		X
8		X
9		X
10		X

Tabla Hash

	Código	Apellidos
0	12	Abadía Ruiz
1	21	Bernabé Pérez
2	68	Carrasco Ruiz
3	38	Domingo Lucas
4	52	Fernández Sánchez
5	70	Jiménez Ruiz
6	44	Martín Pérez
7	18	Rodríguez Gómez

Fichero

Tamaño: 8 registros

Funcionamiento:

$$\text{Registro } 0 \equiv (12, \text{Abadía Ruiz})$$

$$h(12) = 12 \% 11 = 1$$

↑  
módulo

Código	Pos
0	X
1	12
⋮	⋮

Código	Apellidos
0	12
1	21
⋮	⋮
⋮	⋮

k	12	21	68	38	52	70	44	18
h(k)	1	10	2	5	8	4	0	7

Código	Pos
0	44
1	12
2	68
3	X
4	70
5	38
6	X
7	18
8	52
9	X
10	21

Tabla Hash

Código	Apellidos
0	12 Abadía Ruiz
1	21 Bernabé Pérez
2	68 Carrasco Ruiz
3	38 Domingo Lucas
4	52 Fernández Sánchez
5	70 Jiménez Ruiz
6	44 Martín Pérez
7	18 Rodríguez Gómez

Fichero

Ejemplo: consultas.

Si queremos obtener los datos del registro con código  $k=52$ . (en el ejemplo anterior).

- 1.)  $52 \% 11 = 8$
- 2.) Accedemos a la casilla 8 de la tabla Hash.
- 3.) Consultamos la posición del registro : 4.
- 4.) Accedemos a la posición en el fichero, recuperando la información : (52, Fernández Sánchez).

Datos del registro con código  $k=14$ .

- 1.)  $14 \% 11 = 3$
- 2.) Casilla 3 vacía  $\Rightarrow$  registro inexistente.

### Funciones Hash

$$h: C \rightarrow Z$$

- $C$ : el dominio, corresponde al conjunto de posibles claves.
- $Z$ : el rango, es el conjunto de enteros positivos y corresponde al conjunto de índices sobre la tabla hash.

La función hash debe definirse de forma que:

- Sea rápida de calcular.
- Tome todos y cada uno de los posibles valores
- Distribuya de forma más aleatoria posible las claves.
- Minimice el número de colisiones.

1. Truncamiento: consiste en eliminar algunos dígitos de la clave

$$h(123456789) = h(123\cancel{456}789) = 123$$

$$h(121567890) = h(121\cancel{567}890) = 121$$

WUOLAH

# Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



- Inconveniente: la tabla flash deberá tener un tamaño potencia de 10.
- Alternativa: truncamiento a nivel interno (a nivel de bits). La tabla debe tener un tamaño de potencia de 2.

2. **Plegado**: consiste en dividir una clave numérica en dos o más partes y sumarlas.

$$h(\underline{123456}) = 123 + 456 = 579$$

Puede modificarse para que robe algún sumando.

$$h(\underline{123456}) = 123 + 654 = 777$$

Puede combinarse con el truncamiento.

$$h(\underline{456882}) = 456 + 882 = 1338 \rightarrow \underline{1338} = 338$$

Puede involucrar más de 2 sumandos.

$$h(123456789) = h(\underline{123456789}) = 123 + 456 + 789 = 1368$$

- Inconveniente: el tamaño de la tabla hash debe ser potencia de 10.

3. **Multiplicación**: similar al plegado, pero en lugar de sumas, involucra productos. Puede haber plegado antes o después del producto.

Ejemplo:

Tabla de tamaño 1000 y claves de 9 dígitos.

$$h(\underline{123456789}) = 123 \cdot 789 = \underline{97047} = 7047$$

Variantes:

- Cuadrado del centro.
- Centro del cuadrado.

- **Cuadrado del centro**: seleccionar cierto número de cifras del centro de la clave y calcular su cuadrado [+ truncamiento].

$$h(\underline{123456789}) = 7936$$
$$\overline{\rightarrow 456^2} = \underline{207936} = 7936$$

- **Centro del cuadrado**: calcular el cuadrado de la clave y seleccionar un cierto número de cifras del centro.

$$h(1234) = 1234^2 = \underline{1522756} = 2275$$

4. **Resto de la división**: consiste en tomar el resto de la división de la clave entre el tamaño de la tabla (M).  $h(K) = K \bmod M$  /  $h(K) = K \% M$

- Método muy simple que no requiere truncamiento.

Ejemplo:  $h(K) = K \% M$

Claves: 12, 21, 68, 38, 52, 70, 44, 18 → Rango 0...10 (11 casillas)

$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$   
1 10 2 5 8 4 0 7



Consideraciones sobre el método del resto:

- El tamaño de la tabla hash debe ser, al menos, igual al número de claves posibles.
- La mejor elección es no tomar  $M$  simplemente par o impar, sino primo  $\rightarrow M$  un número primo mayor que el número de claves.

## Tratamiento de colisiones

**Motivación:** en la práctica totalidad de los casos, las funciones hash provocan colisiones.

**Objetivo:** encontrar un mecanismo para la clave que provoca la colisión de forma que más tarde, en una operación de consulta, la búsqueda sea eficiente.

Alternativas para resolver colisiones: dependen de la estructura de datos escogida.

En última instancia, depende si conocemos de antemano el número de elementos a ubicar en la tabla hash.

## Hashing abierto

Consiste en construir para cada índice de la tabla una lista de claves sündigmas.

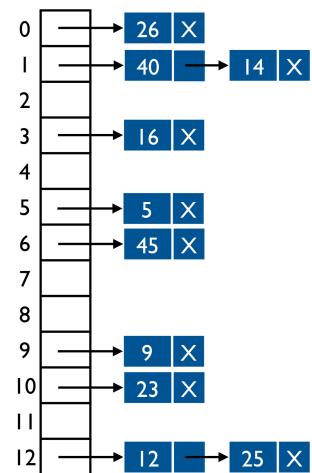
↳ Cada una de estas listas puede implementarse como una lista dinámica.

El tamaño de la tabla Hash se fija a priori y suele implementarse como un vector estático de punteros a estas listas.

- Ventaja: la tabla puede tener un tamaño inferior al número de claves, ya que "crece" con memoria dinámica.
- Desventaja: el espacio adicional requerido por los punteros necesarios para mantener las listas y la eficiencia de las operaciones sobre las listas.

Búsqueda: calculamos el valor hash de la clave y buscamos en la lista enlazada correspondiente.

- Si la inserción es LIFO o FIFO, se debe recorrer la lista completa.
- Si se inserta de forma ordenada, se reduce, en media, el tiempo de búsqueda.
- Búsqueda de clave inexistente: si se llega al final de la lista correspondiente y no se encuentra un nodo con la clave buscada.



- Las colisiones se resuelven insertándolas en una lista.
- La ED resultante es un vector de listas.
- Factor de carga: número medio de claves por lista.
- Objetivo: que el factor de carga esté próximo a 1.

Ejemplo:

23, 45, 16, 26, 40, 14, 5, 12, 9, 25

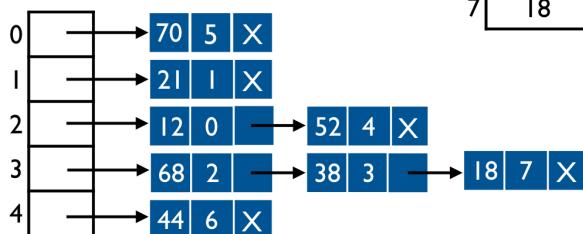
con  $h(x) = x \% 13$

**WUOLAH**

Ejemplo:

k	12	21	68	38	52	70	44	18
h(k)	2	1	3	3	2	0	4	3

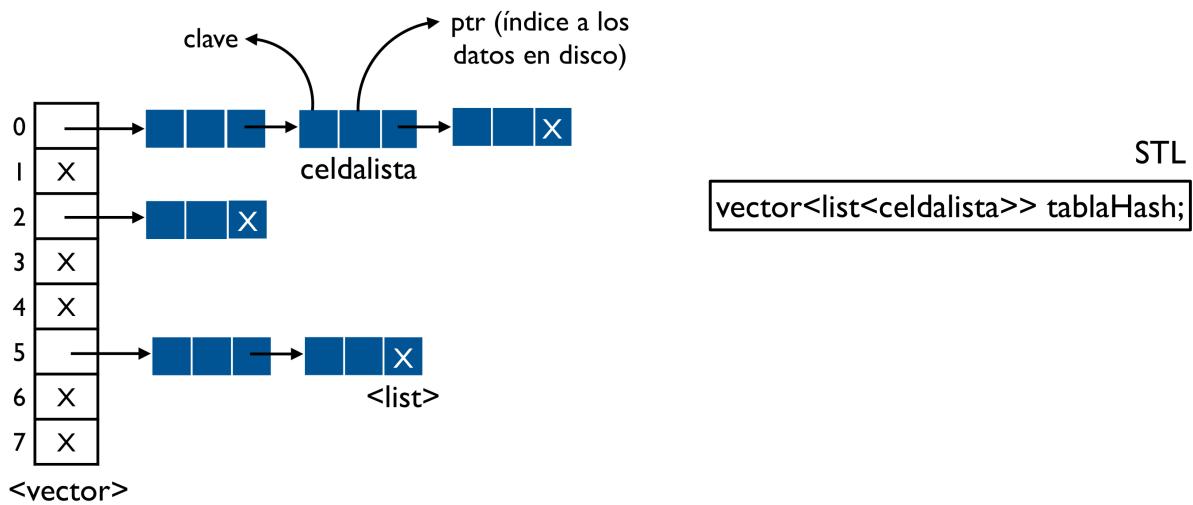
$$h(k) = k \% M, \text{ con } M = 5$$



	Código	Apellidos
0	12	Abadía Ruiz
1	21	Bernabé Pérez
2	68	Carrasco Ruiz
3	38	Domingo Lucas
4	52	Fernández Sánchez
5	70	Jiménez Ruiz
6	44	Martín Pérez
7	18	Rodríguez Gómez

Fichero

## Clase Tabla Hash Abierta



## Hashing cerrado

Usamos un vector para alojar la tabla Hash.

Rehashing: cuando se produzca colisión, la resolvemos asignándole otro valor hash a la clave hasta encontrar un hueco.

Estrategias:

- Rehashing lineal
- Sondeo aleatorio.
- Hashing doble

Las busquedas se hacen siguiendo la misma secuencia de la función hash usada para la inserción.

! Cuidado con los borrados ! La casilla puede formar parte de una secuencia de búsqueda.

↳ La casilla debe marcarse como borrada, un estado diferente al de libre u ocupada.

## Diferencia entre casilla libre y borrada:

- Inserción: borrada y libre son equivalentes ( disponemos de un hueco).
- Búsqueda: borrada y ocupada son equivalentes ( seguimos el proceso de búsqueda).

**Redimensionamiento:** consiste en volver a construir la tabla hash con un nuevo tamaño y volver a hacer hashin de todas las claves de la tabla antigua. ( función hash cambia al cambiar M).

Debe realizarse cuando la tabla hash se desborda (se llena) o cuando su eficiencia decaiga demasiado debido a inserciones y borrados.

### 1. Rehashing lineal

$$h_i(K) = [h(K) + (i-1)] \% M, i = 2, 3, \dots$$

Estrategia:

- Se evalúa  $h(K)$  para una clave  $K$  y hay colisión.
- Generamos la secuencia de valores  $h_2(K), h_3(K), \dots$  mientras se mantenga el estado de colisión.
- Cuando para un  $t$ ,  $h_t(K)$  no se produzca colisión, se termina la secuencia de rehashing y ubicamos la clave en  $h_t(K)$ .

Podemos reescribir la función de rehashing lineal como:

$$\begin{cases} h_0(K) = h(K) \\ h_i(K) = [h_{i-1}(K) + 1] \% M, i = 2, 3, \dots \end{cases}$$

Ejemplo: 23, 45, 16, 26, 40, 14, 5, 12, 9, 25 con  $h(x) = x \% 13$

	Clave	Posición	Status
0	26	pos	O
1	40	pos	O
2	14	pos	O
3	16	pos	O
4	25	pos	O
5	5	pos	O
6	45	pos	O
7			L
8			L
9	9	pos	O
10	23	pos	O
11			L
12	12	pos	O

O: Casilla ocupada  
L: Casilla libre  
B: Casilla borrada

k	23	45	16	26	40	14	5	12	9	25
h(k)	10	6	3	0	1	1	5	12	9	12

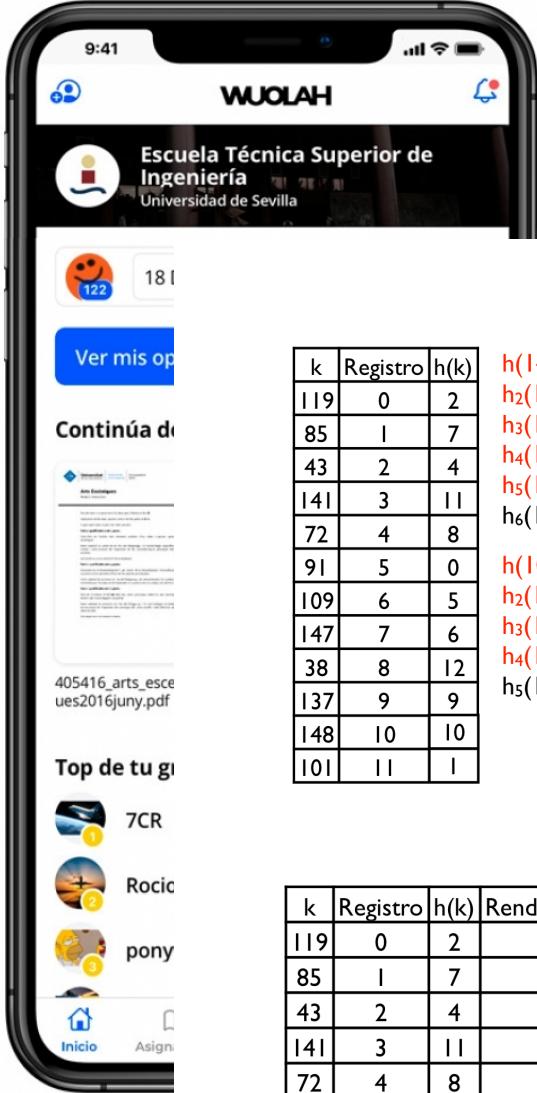


k	Registro	h(k)
119	0	2
85	1	7
43	2	4
141	3	11
72	4	8
91	5	0
109	6	5
147	7	6
38	8	12
137	9	9
148	10	
101	11	

$$\begin{aligned} h(72) &= 7 \\ h_2(72) &= (7+(2-1)) \% 13 = 8 \\ h(147) &= 4 \\ h_2(147) &= (4+(2-1)) \% 13 = 5 \\ h_3(147) &= (4+(3-1)) \% 13 = 6 \\ h(137) &= 7 \\ h_2(137) &= (7+(2-1)) \% 13 = 8 \\ h_3(147) &= (7+(3-1)) \% 13 = 9 \end{aligned}$$

	Clave	Posición	Status
0	91	5	O
1			L
2	119	0	O
3			L
4	43	2	O
5	109	6	O
6	147	7	O
7	85	1	O
8	72	4	O
9	137	9	O
10			L
11	141	3	O
12	38	8	O

WUOLAH



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

k	Registro	h(k)
119	0	2
85	1	7
43	2	4
141	3	11
72	4	8
91	5	0
109	6	5
147	7	6
38	8	12
137	9	9
148	10	10
101	11	1

$$\begin{aligned}
 h(148) &= 5 \\
 h_2(148) &= (5+(2-1))\%13 = 6 \\
 h_3(148) &= (5+(3-1))\%13 = 7 \\
 h_4(148) &= (5+(4-1))\%13 = 8 \\
 h_5(148) &= (5+(5-1))\%13 = 9 \\
 h_6(148) &= (5+(6-1))\%13 = 10
 \end{aligned}$$

$$\begin{aligned}
 h(101) &= 10 \\
 h_2(101) &= (10+(2-1))\%13 = 11 \\
 h_3(101) &= (10+(3-1))\%13 = 12 \\
 h_4(101) &= (10+(4-1))\%13 = 0 \\
 h_5(101) &= (10+(5-1))\%13 = 1
 \end{aligned}$$

	Clave	Posición	Status
0	91	5	O
1	101	11	O
2	119	0	O
3			L
4	43	2	O
5	109	6	O
6	147	7	O
7	85	1	O
8	72	4	O
9	137	9	O
10	148	10	O
11	141	3	O
12	38	8	O

k	Registro	h(k)	Rendimiento
119	0	2	1
85	1	7	1
43	2	4	1
141	3	11	1
72	4	8	2
91	5	0	1
109	6	5	1
147	7	6	3
38	8	12	1
137	9	9	3
148	10	10	6
101	11	1	5
			26

	Clave	Posición	Status
0	91	5	O
1	101	11	O
2	119	0	O
3			L
4	43	2	O
5	109	6	O
6	147	7	O
7	85	1	O
8	72	4	O
9	137	9	O
10	148	11	O
11	141	3	O
12	38	8	O

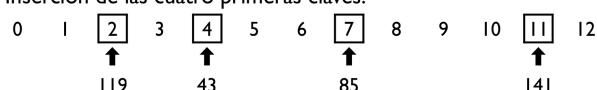
El rehashing lineal tiende a crear agrupaciones primarias.

Una agrupación primaria es una sucesión de casillas ocupadas en una tabla Hash a distancia 1 (contiguas).

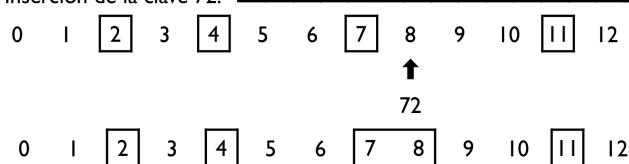
Las agrupaciones primarias conllevan largas series de búsqueda que degradan la eficiencia de las inserciones y los borrados.

Ejemplo:

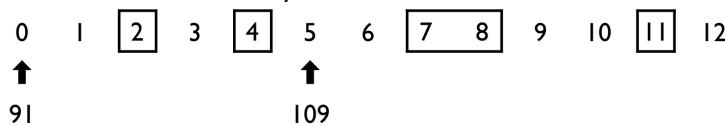
Inserción de las cuatro primeras claves:



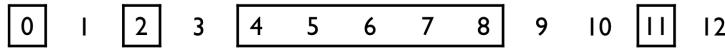
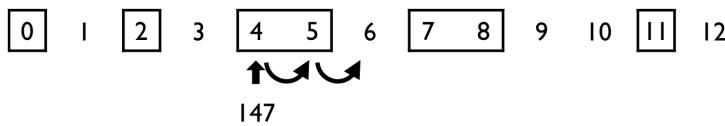
Inserción de la clave 72:



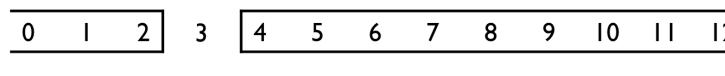
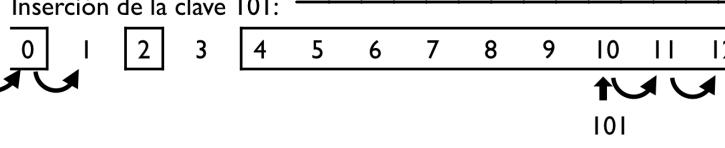
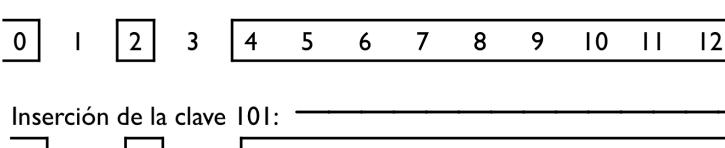
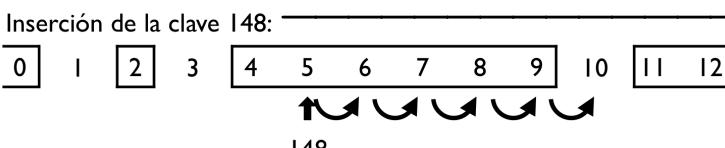
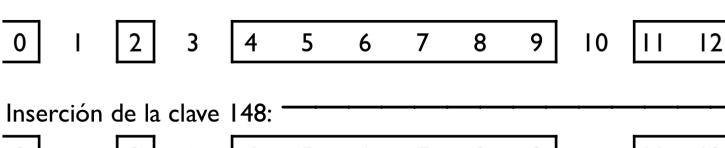
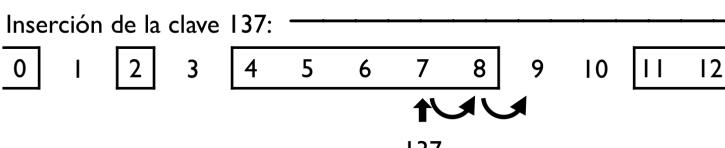
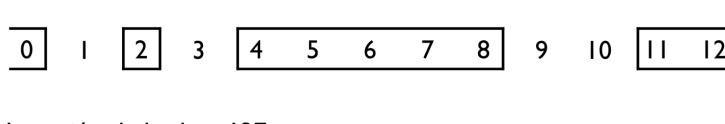
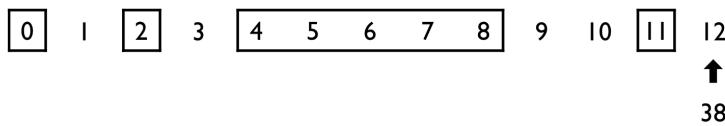
Inserción de las claves 91 y 109:



Inserción de la clave 147:



Inserción de la clave 38:



Soluciones ante la aparición de agrupaciones primarias:

- Mantener estructuras de datos auxiliares que mantengan información (inicio y fin) de las agrupaciones primarias, de forma que se pueda acceder directamente a los "huecos".
- Buscar otros métodos que distribuyan las casillas vacías de forma más aleatoria (al fin y al cabo, la idea del hashing es la distribución "aleatoria" de las claves).

## Algoritmo de búsqueda

1. Calcular  $h(K)$ .
2. Si ( no borrada( $h(K)$ )  $\&$  clave( $h(K)$ ) ==  $K$ )  
    posición = registro( $h(K)$ )

Si no,

    repetir

$$h_i(K) = \text{rehashing}(h_{i-1}(K))$$

    hasta que ( no borrada( $h(K)$ )  $\&$  (clave( $h(K)$ ) ==  $K$  || vacía( $h_i(K)$ )))

Si ( clave( $h_i(K)$ ) ==  $K$ )

    posición = registro( $h_i(K)$ )

Si no,

    posición = -1

3. Devolver (posición)

## 2. Sondeo aleatorio

$$h_i(K) = (h(K) + (i-1) * C) \% M, i=2,3,\dots$$

$$h_i(K) = [h_{i-1}(K) + C] \% M, i=2,3,\dots$$

donde:

- $h(K)$  es el valor de la función hash
- $M$  es el tamaño de la tabla.
- $C > 1$  es primo relativo con  $M$ .

Ejemplo: 119, 85, 43, 141, 72, 91, 109, 147, 38, 137, 148, 101 con  $h(x) = x \% 13$

k	Registro	$h(k)$
119	0	2
85	1	7
43	2	4
141	3	11
72	4	12
91	5	0
109	6	5
147	7	9
38	8	1
137	9	
148	10	
101	11	

$$h(72) = 7$$

$$h_2(72) = (7 + (2-1) * 5) \% 13 = 12$$

$$h(147) = 4$$

$$h_2(147) = (4 + (2-1) * 5) \% 13 = 9$$

$$h(38) = 12$$

$$h_2(38) = (12 + (2-1) * 5) \% 13 = 4$$

$$h_3(38) = (12 + (3-1) * 5) \% 13 = 9$$

$$h_4(38) = (12 + (4-1) * 5) \% 13 = 1$$

O: Casilla ocupada  
L: Casilla libre  
B: Casilla borrada

	Clave	Posición	Status
0	91	5	O
1	38	8	O
2	119	0	O
3			L
4	43	2	O
5	109	6	O
6			L
7	85	1	O
8			L
9	147	7	O
10			L
11	141	3	O
12	72	4	O

k	Registro	h(k)
119	0	2
85	1	7
43	2	4
141	3	11
72	4	12
91	5	0
109	6	5
147	7	9
38	8	1
137	9	6
148	10	10
101	11	

$$\begin{aligned}
 h(137) &= 7 \\
 h_2(137) &= (7+(2-1) * 5) \% 13 = 12 \\
 h_3(137) &= (7+(3-1) * 5) \% 13 = 4 \\
 h_4(137) &= (7+(4-1) * 5) \% 13 = 9 \\
 h_5(137) &= (7+(5-1) * 5) \% 13 = 1 \\
 h_6(137) &= (7+(6-1) * 5) \% 13 = 6 \\
 h(148) &= 5 \\
 h_2(148) &= (5+(2-1) * 5) \% 13 = 10
 \end{aligned}$$

O: Casilla ocupada  
L: Casilla libre  
B: Casilla borrada

	Clave	Posición	Status
0	91	5	O
1	38	8	O
2	119	0	O
3			L
4	43	2	O
5	109	6	O
6	137	9	O
7	85	1	O
8			L
9	147	7	O
10	137	9	O
11	141	3	O
12	72	4	O

k	Registro	h(k)
119	0	2
85	1	7
43	2	4
141	3	11
72	4	12
91	5	0
109	6	5
147	7	9
38	8	1
137	9	6
148	10	10
101	11	3

$$\begin{aligned}
 h(101) &= 10 \\
 h_2(101) &= (10+(2-1) * 5) \% 13 = 2 \\
 h_3(101) &= (10+(3-1) * 5) \% 13 = 7 \\
 h_4(101) &= (10+(4-1) * 5) \% 13 = 12 \\
 h_5(101) &= (10+(5-1) * 5) \% 13 = 4 \\
 h_5(101) &= (10+(5-1) * 5) \% 13 = 9 \\
 h_5(101) &= (10+(5-1) * 5) \% 13 = 1 \\
 h_5(101) &= (10+(5-1) * 5) \% 13 = 6 \\
 h_5(101) &= (10+(5-1) * 5) \% 13 = 11 \\
 h_6(101) &= (10+(6-1) * 5) \% 13 = 3
 \end{aligned}$$

O: Casilla ocupada  
L: Casilla libre  
B: Casilla borrada

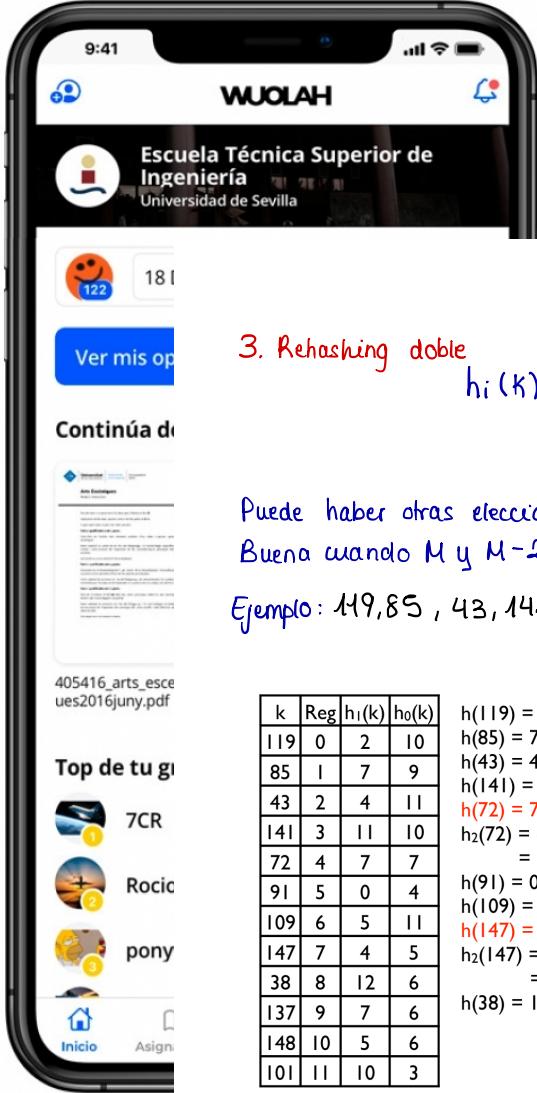
	Clave	Posición	Status
0	91	5	O
1	38	8	O
2	119	0	O
3	101	11	O
4	43	2	O
5	109	6	O
6	137	9	O
7	85	1	O
8			L
9	147	7	O
10	137	9	O
11	141	3	O
12	72	4	O

k	Registro	h(k)	Rendimiento
119	0	2	1
85	1	7	1
43	2	4	1
141	3	11	1
72	4	12	2
91	5	0	1
109	6	5	1
147	7	9	2
38	8	1	4
137	9	6	6
148	10	10	2
101	11	3	10

**Problema:**  
Agrupaciones secundarias de orden C

33

	Clave	Posición	Status
0	91	5	O
1	38	8	O
2	119	0	O
3	101	11	O
4	43	2	O
5	109	6	O
6	137	9	O
7	85	1	O
8			L
9	147	7	O
10	137	9	O
11	141	3	O
12	72	4	O



**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

### 3. Rehashing doble

$$h_i(K) = (h_{i-1}(K) + h_0(K)) \% M, \quad i = 2, 3, \dots$$

$$h_0(K) = 1 + (K \% (M - 2))$$

$$h_1(K) = h(K)$$

Puede haber otras elecciones de  $h_0(K)$ , siempre que no sea constante y distinta de 0. Buena cuando  $M$  y  $M-2$  son primos relativos.

Ejemplo: 119, 85, 43, 141, 72, 91, 109, 147, 38, 137, 148, 101 con  $h(x) = x \% 13$

k	Reg	$h_1(k)$	$h_0(k)$
119	0	2	10
85	1	7	9
43	2	4	11
141	3	11	10
72	4	7	7
91	5	0	4
109	6	5	11
147	7	4	5
38	8	12	6
137	9	7	6
148	10	5	6
101	11	10	3

$$\begin{aligned}
 h(119) &= 2 \\
 h(85) &= 7 \\
 h(43) &= 4 \\
 h(141) &= 11 \\
 \mathbf{h(72) = 7} \\
 h_2(72) &= (h_1(72) + h_0(72)) \% 13 = \\
 &= (7+7)\%13 = 1 \\
 h(91) &= 0 \\
 h(109) &= 5 \\
 \mathbf{h(147) = 4} \\
 h_2(147) &= (h_1(147) + h_0(147)) \% 13 = \\
 &= (4+5)\%13 = 9 \\
 h(38) &= 12
 \end{aligned}$$

O: Casilla ocupada  
L: Casilla libre  
B: Casilla borrada

	Clave	Posición	Status
0	91	5	O
1	72	4	O
2	119	0	O
3			L
4	43	2	O
5	109	6	O
6			L
7	85	1	O
8			L
9	147	7	O
10			L
11	141	3	O
12	38	8	O

k	Reg	$h_1(k)$	$h_0(k)$
119	0	2	10
85	1	7	9
43	2	4	11
141	3	11	10
72	4	7	7
91	5	0	4
109	6	5	11
147	7	4	5
38	8	12	6
137	9	7	6
148	10	5	6
101	11	10	3

$$\begin{aligned}
 \mathbf{h(137) = 7} \\
 h_2(137) &= (h_1(137) + h_0(137)) \% 13 = \\
 &= (7+6)\%13 = 0 \\
 h_3(137) &= (h_2(137) + h_0(137)) \% 13 = \\
 &= (0+6)\%13 = 6 \\
 \mathbf{h(148) = 5} \\
 h_2(148) &= (h_1(148) + h_0(148)) \% 13 = \\
 &= (5+6)\%13 = 11 \\
 h_3(148) &= (h_2(148) + h_0(148)) \% 13 = \\
 &= (11+6)\%13 = 4 \\
 h_4(148) &= (h_3(148) + h_0(148)) \% 13 = \\
 &= (4+6)\%13 = 10 \\
 \mathbf{h(101) = 10} \\
 h_2(101) &= (h_1(101) + h_0(101)) \% 13 = \\
 &= (10+3)\%13 = 0 \\
 h_3(101) &= (h_2(101) + h_0(101)) \% 13 = \\
 &= (0+3)\%13 = 3
 \end{aligned}$$

	Clave	Posición	Status
0	91	5	O
1	72	4	O
2	119	0	O
3	101	11	O
4	43	2	O
5	109	6	O
6	137	9	O
7	85	1	O
8			L
9	147	7	O
10	148	10	O
11	141	3	O
12	38	8	O

k	Reg	$h_1(k)$	$h_0(k)$	Rendimiento
119	0	2	10	1
85	1	7	9	1
43	2	4	11	1
141	3	11	10	1
72	4	7	7	2
91	5	0	4	1
109	6	5	11	1
147	7	4	5	2
38	8	12	6	1
137	9	7	6	3
148	10	5	6	4
101	11	10	3	3

	Clave	Posición	Status
0	91	5	O
1	72	4	O
2	119	0	O
3	101	11	O
4	43	2	O
5	109	6	O
6	137	9	O
7	85	1	O
8			L
9	147	7	O
10	148	10	O
11	141	3	O
12	38	8	O