

BUSQUEDA EXTERNA I: HASHING

- **Filosofía:** evaluar una función, h , (*función hash*) aplicada a claves, k , cuyo valor, $h(k)$, permite la localización del registro buscado en el fichero.
- **Requisitos:**
 - Acceso directo al fichero
 - La utilización de estructuras de datos adicionales (*tablas hash*).
- **Tabla hash:** estructura de datos auxiliar que permite conocer rápidamente la posición del registro de datos en el fichero que contiene la información asociada a cada clave.

Tabla Hash

0	23121	<i>i</i>
1	24576	<i>n</i>
<i>i</i>	23396	<i>n-1</i>
n-1	22563	1
n	21456	0

Fichero

0	21456
1	22563
<i>i</i>	23121
n-1	23396
n	24576

Clave = 23396

$h(23396) = i$

- **Colisión.** Dadas dos claves k_1 y k_2 distintas, si $h(k_1) = h(k_2)$ se produce una *colisión*.

- **Funciones hash.**

Una función hash debería cumplir dos propiedades:

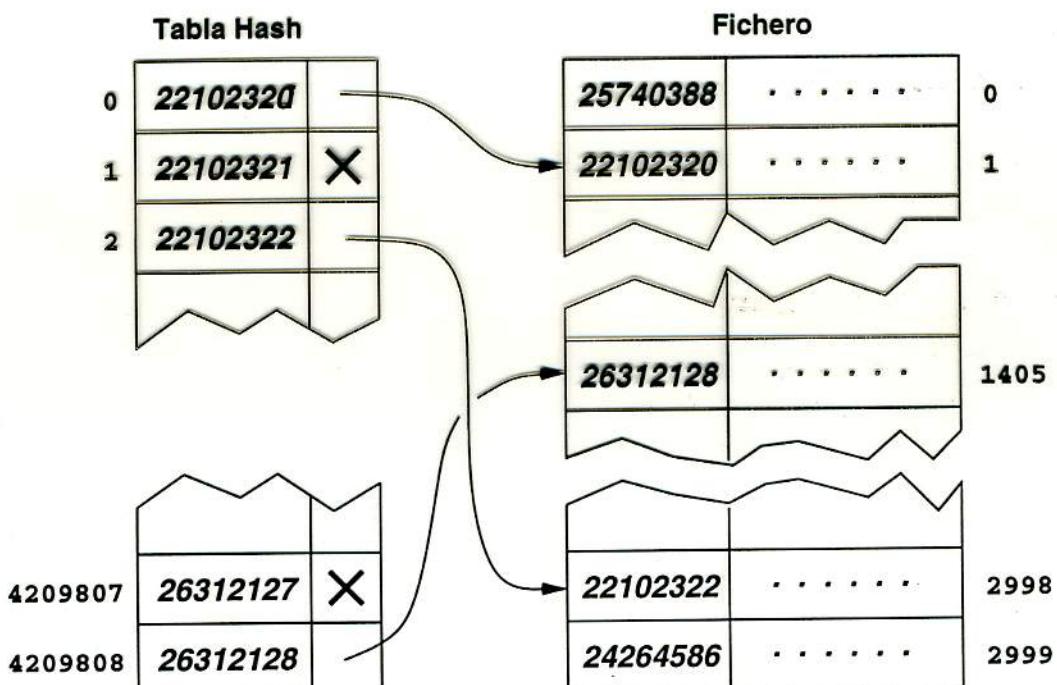
1. Que sea fácil (rápida) de calcular.
2. Que no produzca colisiones.

Sin embargo

- Fichero de alumnos de una Facultad (3000 alumnos).
- El orden en que se encuentran es irrelevante.
- Clave primaria: DNI (22102320 a 26312128).

$$h(k) = k - 22102320$$

La función h es rápida y no produce colisiones.



- La tabla hash contiene $26312128 - 22102320 + 1 = 4209809$ entradas.
- Sólo 3000 (el 0.07 %) direccionan al fichero de datos.

Aunque la función hash es “perfecta” resulta imposible asumir esta solución por problemas de espacio.

Resulta preferible otra solución en la que el tamaño de la tabla sea menor (aunque genere colisiones) en la que se tenga en cuenta el equilibrio entre:

1. El tiempo de resolución de las colisiones
2. El tamaño de la tabla Hash

EJEMPLO (sin aparición de colisiones)

$$m = 11$$

(primo más cercano a 8)

Tabla Hash

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

Fichero

12	Abad Ruiz	0
21	Bernabe Perez	1
68	Camacho Ruiz	2
38	Domingo Lucas	3
62	Fdez Sanchez	4
70	Jvarez Ruiz	5
44	Martín Pérez	6
18	Ruperto Galiaño	7

código apellido

(Tamaño del fichero: 8 registros)

↓
código i → posición del fichero en que está el registro con el código i

Funcionamiento

Registro 0 = (12, Abad Ruiz)

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

Anotamos en la casilla 1 de la tabla hash que el registro con clave 12 se encuentra en la posición 0 del fichero.

Tabla

0	X
1	12 0
2	X
:	:
:	:

12	Abad Ruiz	0
21	Bernabe Perez	1
!	!	!
!	!	!

Repetimos el proceso con los demás códigos.

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

Tabla Hash

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

Fichero

12	Sbad Ruiz	0
21	Bernabe Perez	1
68	Garrago Ruiz	2
38	Domingo Lucas	3
52	Fdez Sanchez	4
70	Juarez Ruiz	5
44	Martín Pérez	6
18	Rupert Galvano	7

Consultas

① Imaginemos que queremos obtener los datos del registro cuyo código es $K = 52$.

- a) $h(52) = 52 \% 11 = 8$
- b) Accedemos a la casilla 8 de la Tabla Hash
- c) Miramos el campo correspondiente a la posición del registro en el fichero, en este caso 4
- d) Abrimos el fichero y accedemos al registro en la posición 4 recuperando la información almacenada: (52, Fdez Sanchez)

② Datos del registro con código $K = 14$?

- a) $h(14) = 14 \% 11 = 3$
- b) Casilla 3 vacía \rightarrow registro inexistente

¿Qué ocurriría si hay colisiones a la hora de almacenar y recuperar la información?

Ejemplo

Tabla Hash

0	X
1	X
2	X
3	X
4	X

Fichero

0	
1	
2	
3	
4	
5	
6	
7	

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

Inserción de las tres primeras claves:

Tabla Hash

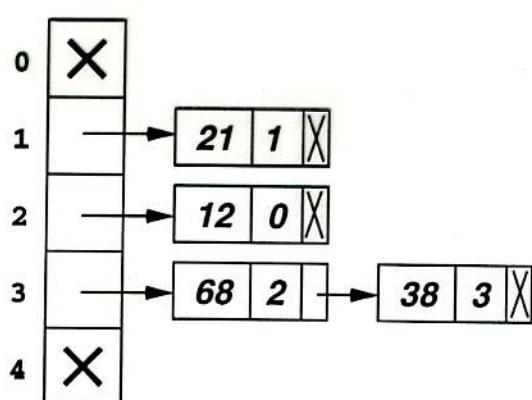
0	X
1	21 1 X
2	12 0 X
3	68 2 X
4	X

Fichero

0	
1	
2	
3	
4	
5	
6	
7	

Inserción de la cuarta clave: *colisión*.

Tabla Hash

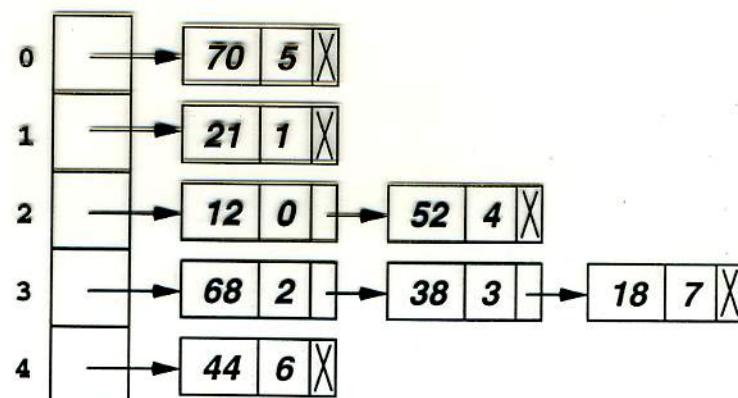


Fichero

0	
1	
2	
3	
4	
5	12 Abad Ruiz
6	21 Berbabe Perez
7	68 Carrasco Martinez
8	38 Domingo Lucas
9	52 Fernandez Sostoa
10	70 Juarez Valladares
11	44 Martin Perez
12	18 Ruperez Galiano

Resultado final:

Tabla Hash



Búsqueda

Se trata de calcular el valor de la función hash para la clave y buscar en una lista enlazada.

- Si la inserción es LIFO o FIFO se debe recorrer (en el peor de los casos) la lista completa.
- Si se inserta de forma ordenada el tiempo de búsqueda se reduce considerablemente (en media).
- Búsqueda de una clave inexistente. Se llega al final de la lista oportuna y no se ha encontrado un nodo que tenga la clave buscada.

RESOLUCION DE COLISIONES: HASHING ABIERTO

La resolución de colisiones por hashing abierto consiste en construir para cada índice de la tabla una lista de claves sinónimas. Así:

✓ A través de la casilla i de la Tabla Hash se accede a una lista que contiene las parejas (clave, dirección en el fichero) para los que el valor de la función hash aplicada a la clave sea i .

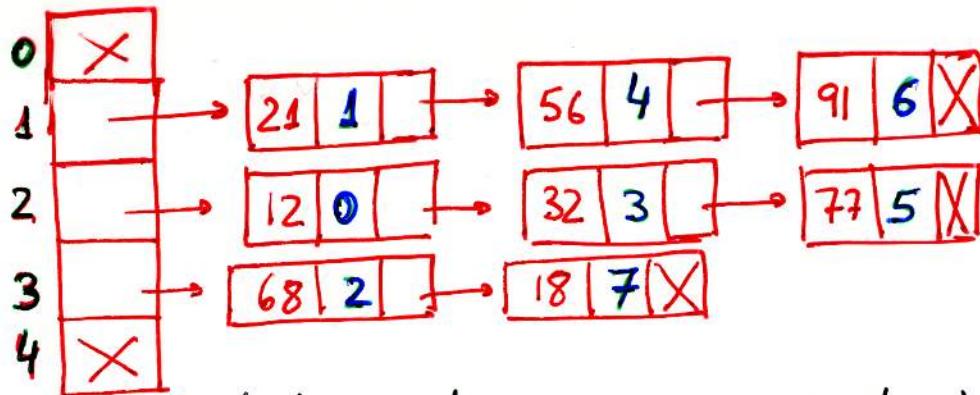
Ejemplo

$$M = 5 \quad h(K) = K \% 5$$

Nºreg.	0	1	2	3	4	5	6	7
K	12	21	68	32	56	77	94	18
$h(K)$	2	1	3	2	4	2	1	3

0	12	Nbad...
1	21	Bernabe...
2	68	Lavado...
3	32	Dormir...
:	:	

Fichero disco



- * El tamaño de la tabla puede ser menor que el nº claves
- * Busqueda
 - cálculo de $h(K)$
 - Busqueda en lista tabla [$h(K)$]
- * → Diferentes posibilidades de mantenimiento de las listas (LIFO, FIFO, ordenada ...) → Eficiencia
- * Inconveniente: Espacio ocupado por los punteros

RESOLUCION DE COLISIONES: HASHING CERRADO

Idea: utilizar una tabla tal que en cada casilla se almacena $\langle \text{clave}, \text{posicion} \rangle$

Problema: Colisión: No puede compartirse una casilla

Solución: Rehashing: Usar una función adicional para (cuando hay colisión) determinar la casilla que le corresponde en la tabla

El uso del rehashing es el mecanismo de resolución de colisiones para el hashing cerrado. 3 estrategias:

- Rehashing lineal
- sondeo aleatorio
- Hashing doble

REHASHING LINEAL

La función de rehashing lineal es:

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

con $h(k)$ valor de la función hash y M tamaño de la tabla.

Estrategia

a) si se evalua $h(k)$ para alguna clave k y hay colisión

b) generar la secuencia de valores $h_2(k), h_3(k), \dots$ mientras se mantenga el estado de colisión

c) cuando para algun t , $h_t(k)$ no produzca colisión, se termina la secuencia de rehashing y la clave se inserta en la posición $h_t(k)$

La función de rehashing lineal predice ponerse como:

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

Ejemplo

Fichero de datos:

$$h(k) = k \% 11$$

Fichero

12	Abad Ruiz	0
21	Bernabe Perez	1
68	Carrasco Martinez	2
32	Duarte Lopez	3
56	Garcia Pi	4
77	Lopez Lopez	5
94	Perez Martin	6
18	Ruiz Perez Galiano	7

a) Inserción de las claves 12, 21 y 68

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

$$h(21) = 21 \% 11 = 10 \quad |$$

$$h(68) = 68 \% 11 = 2$$

0	1	2	3	4	5	6	7	8	9	10
	12	68							21	
	0	2							1	

Tabla Hash
Cerrada

b) Inserción de la clave 32:

$$h(32) = 10 \rightarrow \text{Colisión}$$

Resolución de la Colisión

$$\begin{aligned} h_2(32) &= [h_1(32) + (2-1)] \% 11 = \\ &= (10+1) \% 11 = 0 \end{aligned}$$

Como la casilla \varnothing esta libre lo insertamos allí

c) Inserción de la clave 56

$$h(56) = 56 \% 14 = 1 \rightarrow \underline{\text{colisión}}$$

$$h_2(56) = (1+1) \% 14 = 2 \rightarrow \text{colisión}$$

$$h_3(56) = (1+2) \% 14 = \boxed{3}$$

d) Inserción de la clave 77

$$h(77) = 77 \% 14 = 0 \rightarrow \underline{\text{colisión}}$$

$$h_2(77) = 1, h_3(77) = 2, h_4(77) = 3 \rightarrow \text{colisiones}$$

$$h_5(77) = \boxed{4}$$

e) Inserción de la clave 91

$$h(91) = 91 \% 14 = 3 \rightarrow \underline{\text{colisión}}$$

$$h_2(91) = (3+1) \% 14 = 4 \rightarrow \text{colisión}$$

$$h_3(91) = \boxed{5}$$

f) Inserción de la clave 18

$$h(18) = 7$$

Estado final de la tabla indicando el número intentos

0	1	2	3	4	5	6	7	8	9	10
32	12	68	56	77	91		18	-		21
3	0	2	4	5	6		7	-		1
2	1	1	3	5	3		1	-		1

→ N: total intentos: 17

Ejemplo

Queremos crear una tabla hash cerrada con $M = 13$ posiciones.
El fichero de datos contiene 12 registros.

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

Inserción de las cuatro primeras claves.

Posición	0	1	2	3	4	5	6	7	8	9	10	11	12
Clave			119		43			85			141		
Registro			0		2			1			3		

Inserción de las restantes claves.

- Inserción de la clave 72.

$$h(72) = 7 \quad \text{Colisión}$$

$$h_2(72) = (7 + (2 - 1)) \% 13 = (7 + 1) \% 13 = 8 \% 13 = 8$$

- Inserción de la clave 91.

$h(91) = 0$. Inserción directa, no hay colisión.

- Inserción de la clave 109.

$h(109) = 5$. Inserción directa, no hay colisión.

- Inserción de la clave 147.

$$h(147) = 4 \quad \text{Colisión}$$

$$h_2(147) = (4 + (2 - 1)) \% 13 = (4 + 1) \% 13 = 5 \% 13 = 5 \quad \text{Colisión}$$

$$h_3(147) = (4 + (3 - 1)) \% 13 = (4 + 2) \% 13 = 6 \% 13 = 6$$

- Inserción de la clave 38.

$$h(38) = 12. \text{ Inserción directa, no hay colisión.}$$

- Inserción de la clave 137.

$$h(137) = 7 \quad \text{Colisión}$$

$$h_2(137) = (7 + (2 - 1)) \% 13 = (7 + 1) \% 13 = 8 \% 13 = 8 \quad \text{Colisión}$$

$$h_3(137) = (7 + (3 - 1)) \% 13 = (7 + 2) \% 13 = 9 \% 13 = 9$$

- Inserción de la clave 148.

$$h(148) = 5 \quad \text{Colisión}$$

$$h_2(148) = (5 + (2 - 1)) \% 13 = (5 + 1) \% 13 = 6 \% 13 = 6 \quad \text{Colisión}$$

$$h_3(148) = (5 + (3 - 1)) \% 13 = (5 + 2) \% 13 = 7 \% 13 = 7 \quad \text{Colisión}$$

$$h_4(148) = (5 + (4 - 1)) \% 13 = (5 + 3) \% 13 = 8 \% 13 = 8 \quad \text{Colisión}$$

$$h_5(148) = (5 + (4 - 1)) \% 13 = (5 + 3) \% 13 = 9 \% 13 = 9 \quad \text{Colisión}$$

$$h_6(148) = (5 + (4 - 1)) \% 13 = (5 + 3) \% 13 = 10 \% 13 = 10$$

- Inserción de la clave 101.

$$h(101) = 10 \quad \text{Colisión}$$

$$h_2(101) = (10 + (2 - 1)) \% 13 = (10 + 1) \% 13 = 11 \% 13 = 11 \quad \text{Col.}$$

$$h_3(101) = (10 + (3 - 1)) \% 13 = (10 + 2) \% 13 = 12 \% 13 = 12 \quad \text{Col.}$$

$$h_4(101) = (10 + (4 - 1)) \% 13 = (10 + 3) \% 13 = 13 \% 13 = 0 \quad \text{Colisión}$$

$$h_5(101) = (10 + (4 - 1)) \% 13 = (10 + 3) \% 13 = 14 \% 13 = 1$$

Estado final de la tabla hash.

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

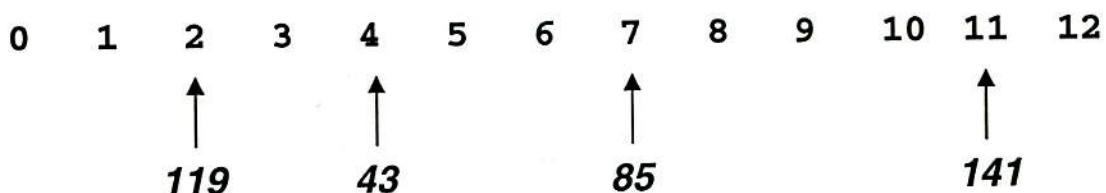
Rendimiento

k	119	85	43	141	72	91	109	147	38	137	148	101	Total
n	1	1	1	1	2	1	1	3	1	3	6	5	26

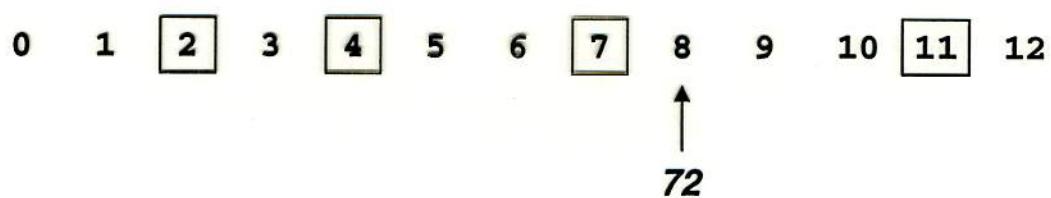
Problema: Agrupaciones primarias.

- Una **agrupación primaria** es una sucesión de casillas ocupadas en una tabla hash a distancia 1.
- Las agrupaciones primarias conllevan largas series de búsqueda que degradan la eficiencia de la inserción o búsqueda.

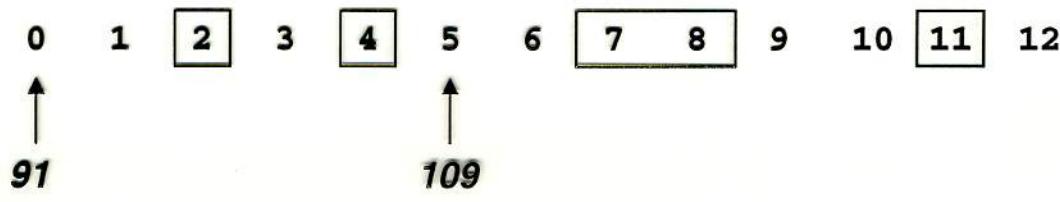
Inserción de las cuatro primeras claves.



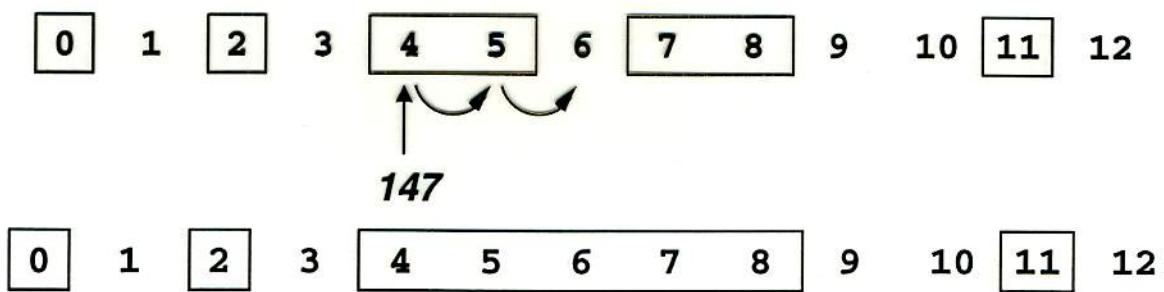
Inserción de la clave 72.



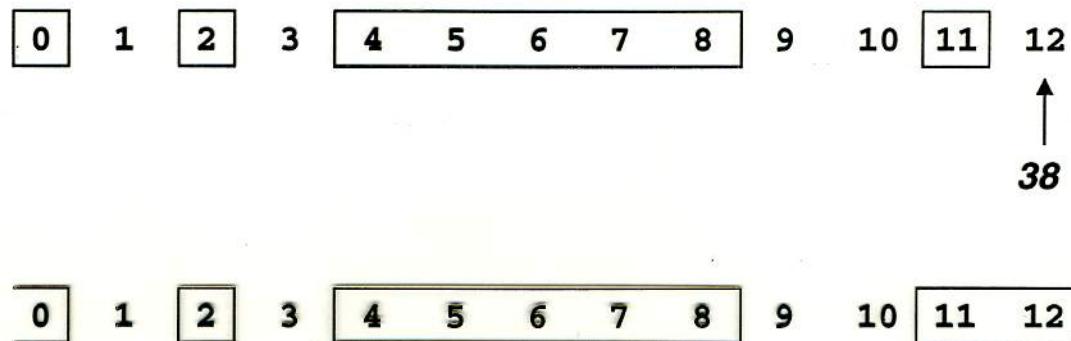
Inserción de las claves 91 y 109.



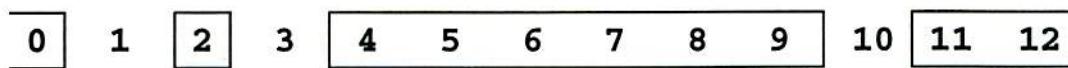
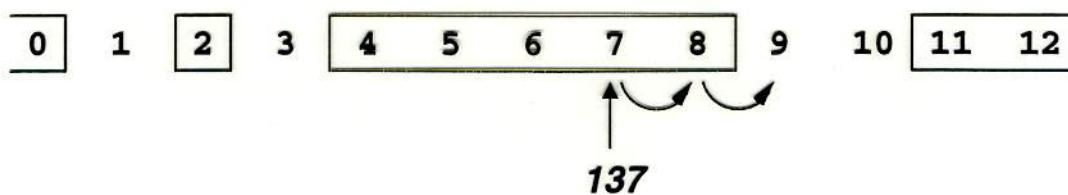
Inserción de la clave 147.



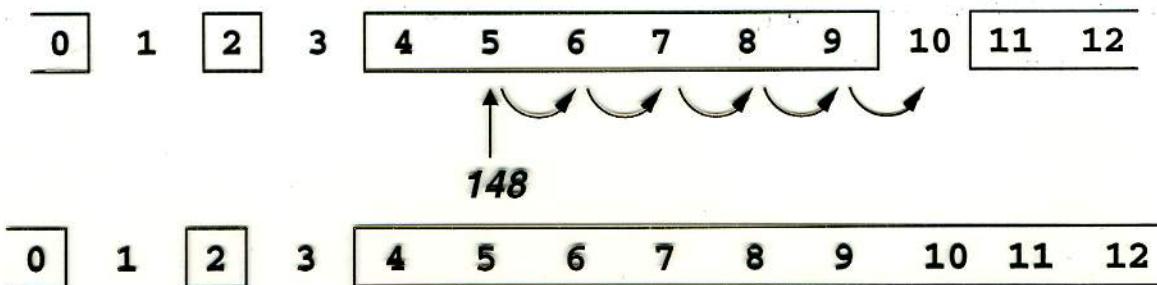
Inserción de la clave 38.



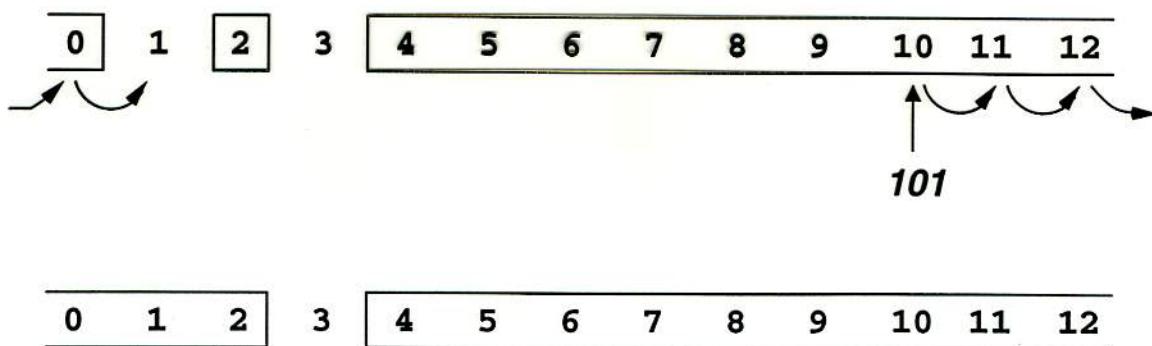
Inserción de la clave 137.



Inserción de la clave 148.



Inserción de la clave 101.



Conclusiones.

- Las agrupaciones primarias de longitud apreciable provocan un bajo rendimiento en las inserciones y consultas.
- Soluciones:
 1. Mantener estructuras de datos auxiliares que mantengan el estado (inicio y fin) de las agrupaciones primarias para acceder directamente a los “huecos”
 2. Cambiar la estrategia de rehashing, de forma que la función de rehashing distribuya de forma más aleatoria las casillas vacías (*sondeo aleatorio*).

Problema

Aparición de agrupaciones primarias: sucesión de casillas ocupadas en la tabla a distancia 1 \Rightarrow da lugar a largas series de búsqueda

Solución

Buscar métodos de rehashing que distribuyan de forma más aleatoria las casillas vacías

Borrados y rehashing

- * No se puede borrar un elemento sin más
- * Mirar cada casilla como inmersa en uno de tres posibles estados:
 - ocupada
 - vacía
 - borrada
- * Cara a la búsqueda: borrada = ocupada
- * Cara a la inserción: borrada = vacía

Redimensionamiento de la Tabla Hash

Cuando una tabla hash se desborda o baja en eficiencia \rightarrow redimensionarla \Rightarrow

- . volver a construir la tabla (>)
- . volver a hacer hashing/rehashing con todas las claves activas en la tabla antigua

ALGORITMO PARA LA BUSQUEDA DE UNA CLAVE

a) Calcular $h(c)$

b) Si (no_borrada(h(c))) \neq (clave(h(c)) = c)

 posicion = registro(h(c))

Si no,

 Repetir

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

 hasta que ((No_borrada(h_i(c))) \neq

 ((clave(h_i(c)) == c) || (vacia(h_i(c)))))

Si (clave(h_i(c)) == c)

 posicion = registro(h_i(c))

Si no posicion = -1

c) devolver (posicion)

Hashing cerrado Sondeo aleatorio

$$h_i = (h(k) + (i-1) \cdot C) \% M \quad i = 2, 3, \dots$$

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

$$h_i(k) = [h_{i-1}(k) + c] \% M$$

$$i = 2, 3, \dots$$

Ejemplo

Queremos crear una tabla hash cerrada con $M = 13$ posiciones.
 El fichero de datos contiene 12 registros.

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

Inserción de las cuatro primeras claves.

Posición	0	1	2	3	4	5	6	7	8	9	10	11	12
Clave			119		43			85			141		
Registro			0		2			1			3		

Inserción de las restantes claves.

- Inserción de la clave 72.

$$h(72) = 7 \quad \text{Colisión}$$

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

- Inserción de la clave 91.

$h(91) = 0$. Inserción directa, no hay colisión.

- Inserción de la clave 109.

$h(109) = 5$. Inserción directa, no hay colisión.

- Inserción de la clave 147.

$h(147) = 4$ Colisión

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

- Inserción de la clave 38.

$h(38) = 12$ Colisión

$$h_2(38) = (12 + (2 - 1)5)\%13 = (12 + 1 \times 5)\%13 = (12 + 5)\%13 = 17\%13 = 4$$

Colisión

$$h_3(38) = (12 + (3 - 1)5)\%13 = (12 + 2 \times 5)\%13 = (12 + 10)\%13 = 22\%13 = 9$$

Colisión

$$h_4(38) = (12 + (4 - 1)5)\%13 = (12 + 3 \times 5)\%13 = (12 + 15)\%13 = 27\%13 = 1$$

- Inserción de la clave 137.

$h(137) = 7$ Colisión

$$h_2(137) = (7 + (2 - 1)5)\%13 = (7 + 1 \times 5)\%13 = (7 + 5)\%13 = 12\%13 = 12$$

Colisión

$$h_3(137) = (7 + (3 - 1)5)\%13 = (7 + 2 \times 5)\%13 = (7 + 10)\%13 = 17\%13 = 4$$

Colisión

$$h_4(137) = (7 + (4 - 1)5)\%13 = (7 + 3 \times 5)\%13 = (7 + 15)\%13 = 22\%13 = 9$$

Colisión

$$h_5(137) = (7 + (5 - 1)5)\%13 = (7 + 4 \times 5)\%13 = (7 + 20)\%13 = 27\%13 = 1$$

Colisión

$$h_6(137) = (7 + (6 - 1)5)\%13 = (7 + 5 \times 5)\%13 = (7 + 25)\%13 = 32\%13 = 6$$

- Inserción de la clave 148.

$h(148) = 5$ Colisión

$$h_2(148) = (5 + (2 - 1)5)\%13 = (5 + 1 \times 5)\%13 = 10\%13 = 10$$

- Inserción de la clave 101.

$$h(101) = 10 \quad \text{Colisión}$$

$$h_2(101) = (10 + (2-1)5)\%13 = (10 + 1 \times 5)\%13 = (10 + 5)\%13 = 15\%13 = 2 \quad \text{Colisión}$$

$$h_3(101) = (10 + (3-1)5)\%13 = (10 + 2 \times 5)\%13 = (10 + 10)\%13 = 20\%13 = 7 \quad \text{Colisión}$$

$$h_4(101) = (10 + (4-1)5)\%13 = (10 + 3 \times 5)\%13 = (10 + 15)\%13 = 25\%13 = 12 \quad \text{Colisión}$$

$$h_5(101) = (10 + (5-1)5)\%13 = (10 + 4 \times 5)\%13 = (10 + 20)\%13 = 30\%13 = 4 \quad \text{Colisión}$$

$$h_6(101) = (10 + (6-1)5)\%13 = (10 + 5 \times 5)\%13 = (10 + 25)\%13 = 35\%13 = 9 \quad \text{Colisión}$$

$$h_7(101) = (10 + (7-1)5)\%13 = (10 + 6 \times 5)\%13 = (10 + 30)\%13 = 40\%13 = 1 \quad \text{Colisión}$$

$$h_8(101) = (10 + (8-1)5)\%13 = (10 + 7 \times 5)\%13 = (10 + 35)\%13 = 45\%13 = 6 \quad \text{Colisión}$$

$$h_9(101) = (10 + (9-1)5)\%13 = (10 + 8 \times 5)\%13 = (10 + 40)\%13 = 50\%13 = 11 \quad \text{Colisión}$$

$$h_{10}(101) = (10 + (10 - 1)5)\%13 = (10 + 9 \times 5)\%13 = (10 + 45)\%13 = 55\%13 = 3$$

Estado final de la tabla hash.

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

Rendimiento

k	119	85	43	141	72	91	109	147	38	137	148	101	Total
n	1	1	1	1	2	1	1	2	4	6	2	10	33

Problema: Agrupaciones secundarias (de orden C).

Hashing cerrado Rehashing doble

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

Alternativa:

$$\begin{aligned} h_0(k) &= 1 + (k \% (M - 2)) \\ h_1(k) &= h(k) \end{aligned}$$

$$\begin{aligned} h_0(k) &= R - (k \% R) \\ R < M \text{ primo} \end{aligned}$$

- Otras elecciones de $h_0(k)$, siempre que no sea cte. y sea distinta de cero.
- Buena cuando M y $M - 2$ son primos relativos.

Ejemplo

Queremos crear una tabla hash cerrada con $M = 13$ posiciones.
El fichero de datos contiene 12 registros.

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

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

Inserción de las cuatro primeras claves.

Posición	0	1	2	3	4	5	6	7	8	9	10	11	12
Clave			119		43			85			141		
Registro			0		2			1			3		

Inserción de las restantes claves.

$$h_i(K) = [h_{i-1}(K) + h_0(K)] \% M$$

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

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

$$h(72) = 7 \quad \text{Colisión}$$

$$h_2(72) = (h_1(72) + h_0(72)) \% 13 = (7 + 7) \% 13 = 14 \% 13 = 1$$

- Inserción de la clave 91.

$$h(91) = 0. \text{ Inserción directa, no hay colisión.}$$

- Inserción de la clave 109.

$$h(109) = 5. \text{ Inserción directa, no hay colisión.}$$

- Inserción de la clave 147.

$$h(147) = 4 \quad \text{Colisión}$$

$$h_2(147) = (h_1(147) + h_0(147)) \% 13 = (4 + 5) \% 13 = 9 \% 13 = 9$$

- Inserción de la clave 38.

$$h(38) = 12. \text{ Inserción directa, no hay colisión.}$$

- Inserción de la clave 137.

$$h(137) = 7 \quad \text{Colisión}$$

$$h_2(137) = (h_1(137) + h_0(137)) \% 13 = (7 + 6) \% 13 = 13 \% 13 = 0 \quad \text{Colisión}$$

$$h_3(137) = (h_2(137) + h_0(137)) \% 13 = (0 + 6) \% 13 = 6 \% 13 = 6$$

- Inserción de la clave 148.

$$h(148) = 5 \quad \text{Colisión}$$

$$h_2(148) = (h_1(148) + h_0(148)) \% 13 = (5 + 6) \% 13 = 11 \% 13 = 11 \quad \text{Colisión}$$

$$h_3(148) = (h_2(148) + h_0(148)) \% 13 = (11 + 6) \% 13 = 17 \% 13 = 4 \quad \text{Colisión}$$

$$h_4(148) = (h_3(148) + h_0(148)) \% 13 = (4 + 6) \% 13 = 10 \% 13 = 10$$

- Inserción de la clave 101.

$$h(101) = 10 \quad \text{Colisión}$$

$$h_2(101) = (h_1(101) + h_0(101)) \% 13 = (10 + 3) \% 13 = 13 \% 13 = 0 \quad \text{Colisión}$$

$$h_3(101) = (h_2(101) + h_0(101)) \% 13 = (0 + 3) \% 13 = 3 \% 13 = 3$$

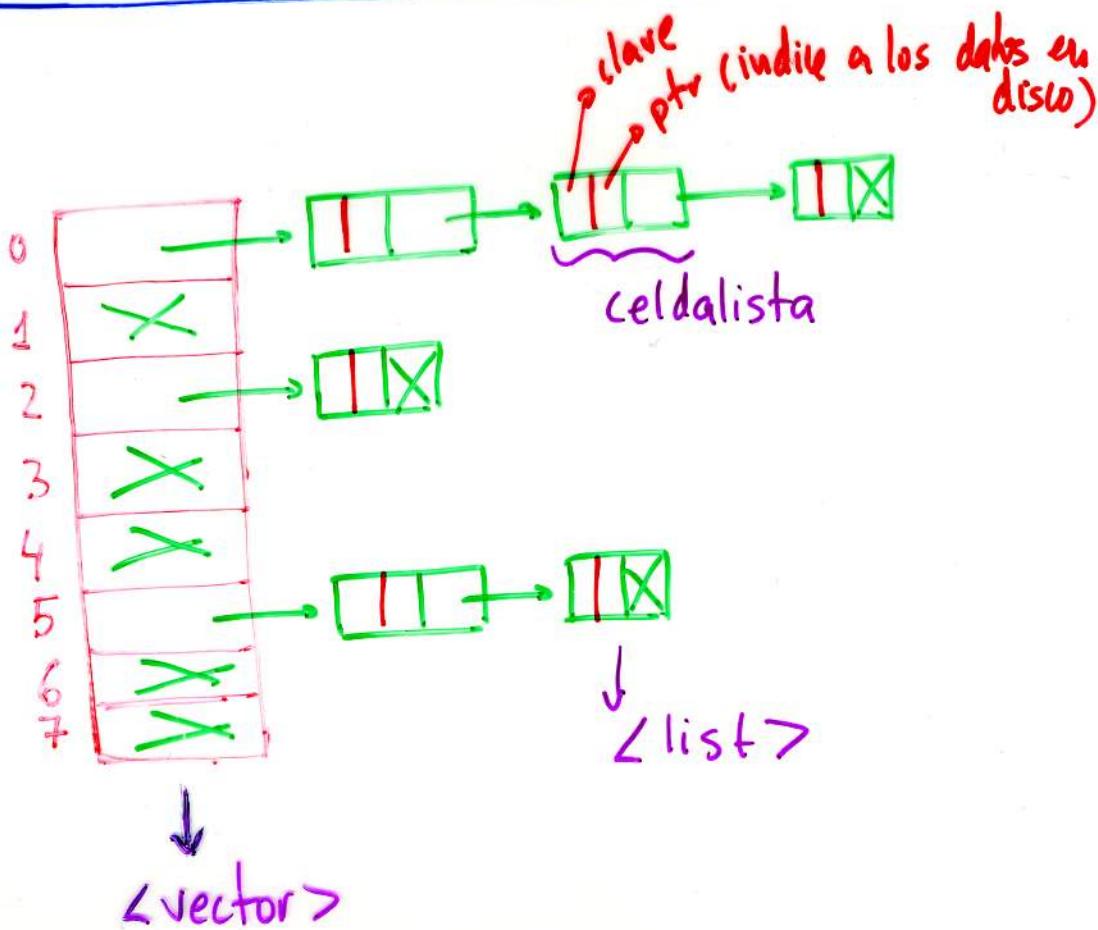
Estado final de la tabla hash.

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

Rendimiento

k	119	85	43	141	72	91	109	147	38	137	148	101	Total
n	1	1	1	1	2	1	1	2	1	3	4	3	21

CLASE TABLA HASH ABIERTA



Definición usando la STL

vector< list<celdalista>> tablahash;

hash_abierto.h

```
#include <list>
#include <vector>
using namespace std;

class CeldaLista {
private:
    int clave;
    int ptr; //indice a la posición de los datos
public:
    CeldaLista() : clave(-1), ptr(-1) {};
    CeldaLista(int c, int p) : clave(c), ptr(p) {};
    ~CeldaLista() {};
    // getters
    int & Clave() { return clave; }
    int & Datos() { return ptr; }
};

/** Cada celda de la lista enlazada contiene:
    - una clave
    - un indice a los datos
```

```
class TablaHash {  
private:  
    vector<list<CeldaLista>> tabla;  
    // La Tabla Hash es un vector de listas enlazadas  
    int fhash (int clave); // función hash  
    list<CeldaLista>::iterator EstaEnFila (int clave, int pos);  
    // Devuelve la posición (mediante un iterador) donde  
    // se encuentra la clave en la fila pos de la tabla hash  
    // Devuelve Ø, si no está  
public:  
    TablaHash (int tam); // constructor de tabla con un  
    // tamaño fijo  
    ~TablaHash();  
    bool Existe (int clave); // comprueba si la clave está en  
    // la tabla  
    bool Insertar (int clave, int ptrdatos);  
    // inserta un par (clave, ptrdatos). Devuelve true si  
    // se ha insertado y false en caso contrario.  
    bool Borrar (int clave); // borra la clave  
    bool Cambiar (int clave, int ptrdatos);  
    // cambia el valor ptrdatos asociado a la clave  
    int Obtener (int clave);  
    // Devuelve el ptrdatos asociado a la clave y -1 si la clave  
    // no está  
    void Imprimir(); // imprime en pantalla la tabla Hash  
};
```

hash_abierto.cpp

```
#include <iostream>
#include <cassert>
#include <hash_abierto.h>

using namespace std;

TablaHash::TablaHash (int tam)
{
    assert(tam > 0);
    tabla.resize(tam); // Fijamos un tamaño para la Tabla
}

int TablaHash::fhash (int clave)
{
    return clave % tabla.size(); // f(x) = x % tamaño
}

list<CeldaLista>::iterator TablaHash::EstaEnFile (int clave,
                                                    int pos)

{
    for (list<CeldaLista>::iterator encontrado = tabla[pos].begin(),
                                    encontrado != tabla[pos].end(); encontrado++)
        if ((*encontrado).clave() == clave)
            return encontrado;
    return {};
} // (*encontrado).clave() es equivalente
   // a encontrado->clave()
```

```

bool TablaHash::Existe (int clave)
{
    int pos = fhash (clave);
    return (EstaEnFila (clave, pos) != 0);
}

bool TablaHash::Insertar (int clave, int ptrdatos)
{
    int pos = fhash (clave);
    if (EstaEnFila (clave, pos) == 0)
        tabla [pos].push_back (CeldaLista (clave, ptrdatos));
    return true;
}
else return false; // NO se ha insertado (clave repetida)
}

bool TablaHash::Cambiar (int clave, int ptrdatos)
{
    int pos = fhash (clave);
    list <CeldaLista>::iterator dondeesta = EstaEnFila (clave, pos);

    if (dondeesta != 0)
        (*dondeesta).Datos () = ptrdatos;
    return true; // dondeesta -> Datos()
}
else return false; // la clave no está en la tabla
}

```

```

int TablaHash::Obtener (int clave)
{
    int pos = fhash (clave);
    list<CeldaLista>::iterator fil = EstaEnFila (clave, pos);
    return (fil != 0) ? fil->Datos() : -1;
}

bool TablaHash::Borrar (int clave)
{
    int pos = fhash (clave);
    list<CeldaLista>::iterator dondeesta = EstaEnFila (
        clave, pos);
    if (dondeesta != 0)
    {
        tabla[pos].erase (dondeesta);
        return true;
    }
    else return false; // La clave no estar en la tab
}

void TablaHash::Imprimir()
{
    for (unsigned int i = 0; i < tabla.size(); i++)
    {
        cout << "Fila" << i << ":" ;
        for (list<CeldaLista>::iterator p = tabla[i].begin();
             p != tabla[i].end(); p++)
            cout << "(" << (*p).clave() << "."
                  << (*p).Datos() << ")";
        cout << endl;
    }
}

```

Estructuras duality

Problema: Mantener una escala de equipos de fútbol en la que cada equipo está en una posición. Los nuevos equipos se agregan al final. Cada equipo puede reemplazar al que está inmediatamente encima y caso de ganarle se intercambia con él.

Operaciones

- **Agregar (equipo)**: Agrega el equipo "equipo" en el último lugar
- **Retirar (equipo)**: Devuelve el nombre del equipo en la posición $i-1$ si "equipo" está en la posición i , $i > 1$
- **Cambiar (i)**: Intercambia los equipos que están en las posiciones i e $i-1$ $i > 1$

Solución 1

Vector de equipos v con $v[i]$ el nombre del equipo i -ésimo, manteniendo un cursor al último equipo insertado

R. Madrid	0
Valencia	1
Deportivo	2
At. Madrid	3
Sevilla	4
Betis	5
	6

último

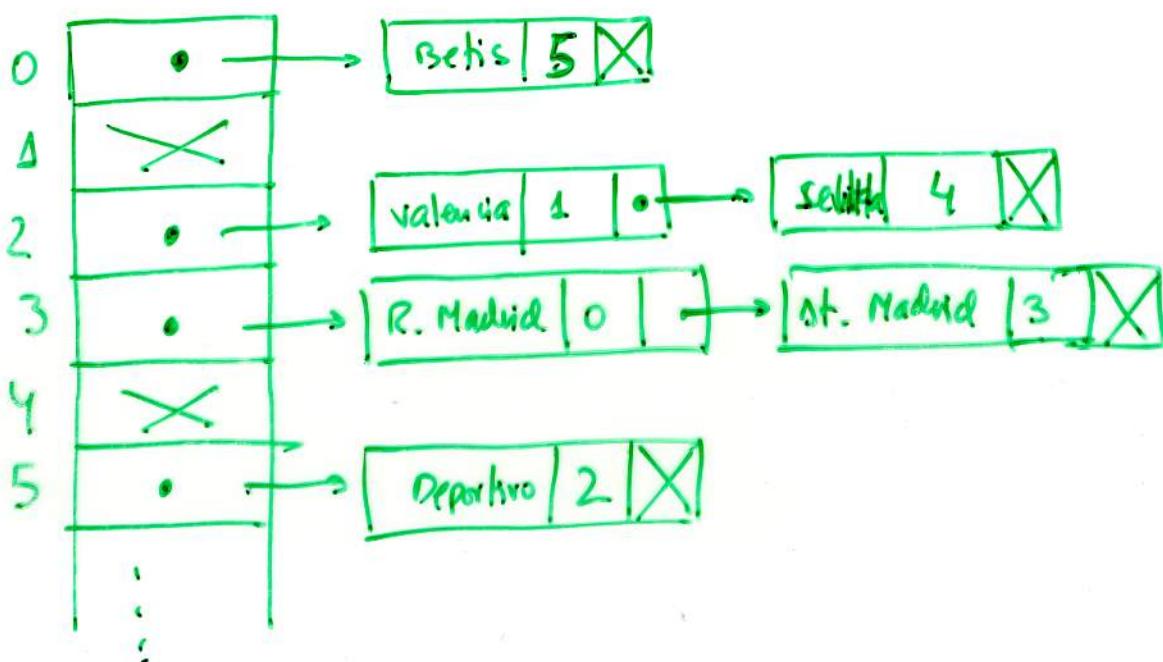
Agregar (equipo) $\rightarrow O(1)$

Rotar (i) $\rightarrow O(1)$

Retirar (equipo) $\rightarrow O(n)$

Solución 2

Tabla hash abierta construida en base al nombre de los equipos. Cada elemento de las listas enlazadas contiene el nombre del equipo y su posición en la clasificación.

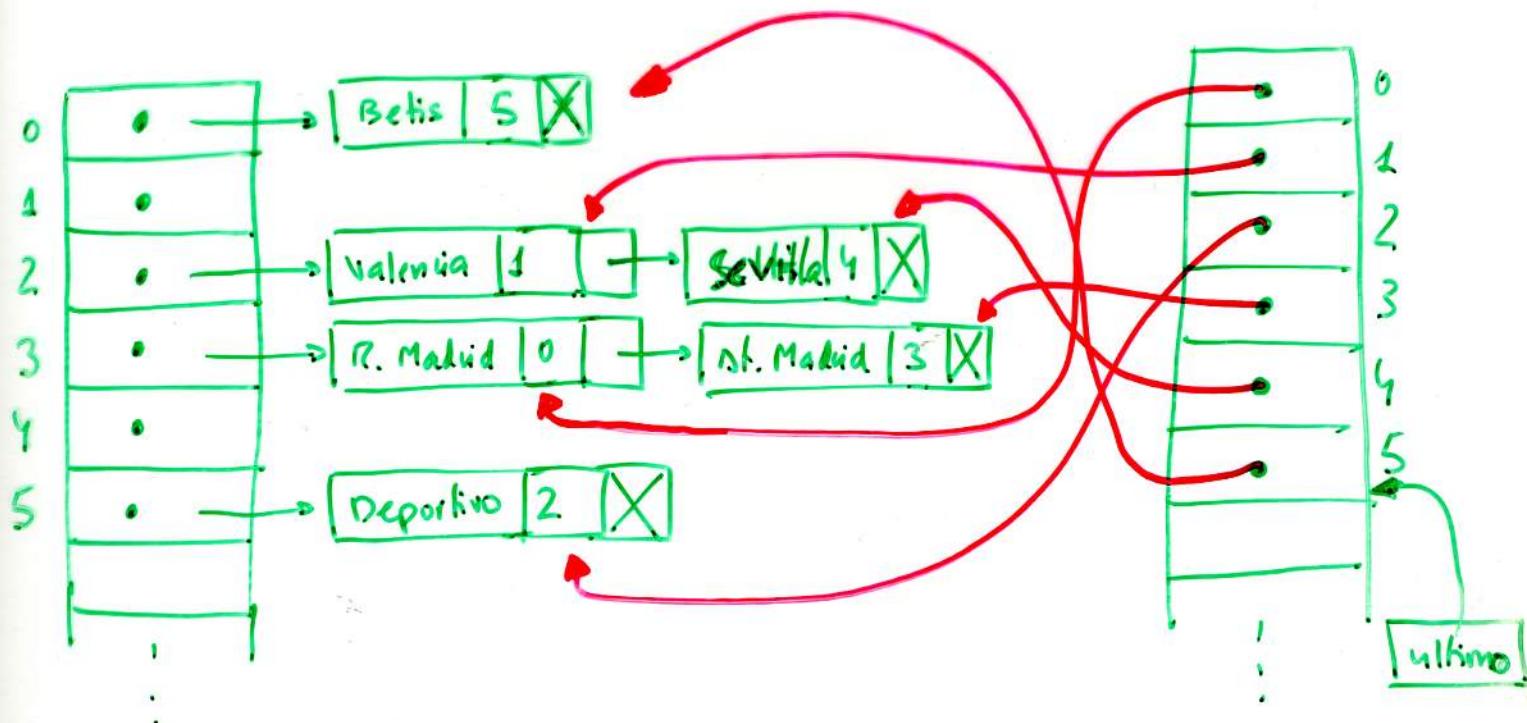


Agregar (equipo) $\rightarrow O(1)$

Cambiar (i) $\rightarrow O(n)$

Retirar (equipo) $\rightarrow O(n)$

Solución híbrida



Agregar (equipo) $\longrightarrow O(1)$

Cambiar (i) $\longrightarrow O(1)$

Retirar (equipo) $\longrightarrow O(1)$