Estructuras de Datos no Lineales

1.5.4. Tablas de dispersión

José Fidel Argudo Argudo
José Antonio Alonso de la Huerta
Mª Teresa García Horcajadas

Versión 1.1

Tablas de dispersión

- En una **tabla de dispersión (o tabla *hash*)** se almacenan los elementos de un conjunto mediante una función asociada (llamada *función hash*), que dado un elemento devuelve la posición exacta (o casi exacta) en la que se almacena.
- **Solución óptima al problema de la búsqueda** de un elemento de un conjunto, puesto que se requiere un **tiempo de orden constante** (determinado por el cálculo de la función *hash*) para localizar la posición de la tabla donde se encuentra el elemento buscado.
- Idea básica subyacente:
  - Cada elemento a almacenar se compone de una clave que lo identifica y, posiblemente, de otros datos asociados.
  - Las claves son números enteros en el rango [0, *M*-1].
  - La tabla se representa como un vector de tamaño *M*, en el cual el elemento con clave *i* se almacena en la posición *i*, es decir, la clave de un elemento corresponde directamente a su dirección en la tabla (función *hash*, $h(i) = i$).
- Esta situación ideal suele estar alejada de la realidad por varias razones:
  - El número de elementos a almacenar no está acotado.
  - El número de claves posibles tampoco está acotado o es muy grande en relación al número de elementos a almacenar.
  - Las claves no son números de tipo entero.

Tablas de dispersión

Funciones *hash*

- **Función *hash***: aplicación que transforma una clave de cierto tipo (número, cadena, fecha…) en una dirección (índice) de la tabla, $h: C \rightarrow [0, M-1]$, donde C es el conjunto de posibles claves y M el tamaño de la tabla (suficiente para almacenar los elementos previstos).
- **Colisión**: se produce cuando a dos claves distintas, x e y, les corresponde el mismo valor de la función *hash*, $h(x) = h(y)$. Entonces se dice que x e y son **claves sinónimas**.
- **Función hash perfecta**: asigna una dirección distinta a cada clave posible, no provoca colisiones entre claves. Implica que podemos disponer de suficiente capacidad en la tabla para todas las claves posibles ($\| C \| \leq M$).
- En la práctica, el número de claves posibles es mayor que el número de elementos a almacenar ($\| C \| > M$), por lo que **las colisiones son inevitables**.
- Propiedades de una buena función *hash*:
  - Eficiencia temporal: debe ser **rápida de calcular**, ya que el tiempo de búsqueda depende de ella.
  - Eficiencia espacial: debe ser una función **sobreyectiva**, o lo que es igual, a todas las posiciones les corresponde al menos una clave, es decir, no deja posiciones de la tabla sin usar.
  - **Distribuye uniformemente las claves por la tabla**: así las claves no tienden a concentrarse en un subconjunto de posiciones y se reduce la probabilidad de colisión.

Tablas de dispersión

Funciones *hash*

Dificultad de encontrar una buena función *hash*

- Supongamos que vamos a direccionar 91 posibles claves en una tabla de tamaño 13.

  Funciones *hash* posibles:

  $$13^{91} = 2{,}34 \cdot 10^{101} \text{ funciones}$$

  Funciones *hash* que distribuyen uniformemente las claves por toda la tabla, asignando exactamente 91/13 = 7 claves a cada dirección:

  $$\frac{91!}{7!^{13}} \cong 10^{92} \text{ funciones óptimas}$$

  Proporción de funciones óptimas:

  $$\frac{10^{92}}{2{,}34 \cdot 10^{101}} = \frac{1}{2{,}34 \cdot 10^{9}} = 4{,}27 \cdot 10^{-10} = \frac{4 \cdot 4{,}27}{4 \cdot 10^{10}} \cong \frac{17}{4 \cdot 10^{10}}$$

¡¡¡17 funciones óptimas de cada
40.000 millones de funciones posibles!!!

Por si fuera poco, además queremos que la función *hash* elegida se calcule con rapidez.

Tablas de dispersión

Funciones *hash*

- El estudio de las **funciones *hash*** se originó con la invención de las **tablas *hash*** como estructura de datos para la **búsqueda eficiente** de información asociada a una clave.
- Otras aplicaciones de las funciones *hash*:
  - Criptografía
  - Autenticación e integridad de información
  - Generación de números pseudoaleatorios
  - etc.
- Se han descubierto numerosas funciones *hash* para distintos tipos de claves con un comportamiento muy bueno para la búsqueda, aunque no es nuestro propósito estudiarlas.
- **La función *hash* la podemos construir en dos fases**:
  1. Definir (o seleccionar entre las muchas existentes) una función *hash* $\boldsymbol{H(x)}$ que transforme las claves en números enteros.

     La biblioteca estándar de C++ (cabecera `functional`) define funciones hash (mediante objetos función) que devuelven valores de tipo `size_t` para los tipos básicos del lenguaje y para algunos tipos de datos de la propia biblioteca como `string`, tales que para dos claves del mismo tipo distintas la probabilidad de colisión se aproxima a $2^{-sizeof(size\_t)}$.
  2. Trasladar cada número así obtenido a una dirección de la tabla, calculando su módulo respecto al tamaño M de la misma, $\boldsymbol{h(x) = H(x)\ mod\ M}$.

Tablas de dispersión

Resolución de colisiones

- ***Hashing* cerrado** o direccionamiento abierto: cuando se produce una colisión se busca una nueva posición libre en la tabla.

- ***Hashing* abierto**, encadenamiento separado o direccionamiento cerrado: cada posición de la tabla es un cubículo (*bucket*) con una estructura adecuada para admitir varias claves sinónimas.

- **Encadenamiento mezclado**: combinación de los dos métodos anteriores, en el que las claves que provocan colisiones se alojan en celdas libres formando listas enlazadas dentro de la tabla. Cada lista de celdas enlazadas almacena una serie de claves que han colisionado, aunque pueden tener iguales o distintos valores *hash* (no tienen por qué ser sinónimas).

Tablas de dispersión

Hashing cerrado

Se establece una **secuencia de exploración** de la tabla para localizar una casilla no ocupada, definida mediante una serie de M **funciones *hash*** $\{h_0, h_1, \ldots, h_{M-1}\}$ que devuelven M valores distintos y donde h_0 es la función *hash* original. Si hay una casilla disponible, alguna de estas funciones la encontrará y en ella se podrá insertar un nuevo elemento.

- **Exploración lineal**
  - $h_i = (h_0 + i \cdot \alpha) \; mod \; M$

    $\alpha \in \mathbb{N}$ y primo relativo con *M* para asegurar el recorrido de la tabla completa.
  - La distancia entre las celdas exploradas es la constante α.
  - Produce **agrupamientos primarios**: posiciones ocupadas a distancia α (consecutivas si $\alpha = 1$), que aumentan el tiempo de resolución de posteriores colisiones.
  - Los agrupamientos primarios provocan una degradación muy rápida de la eficiencia de las operaciones, porque tienden a unirse entre ellos y hacerse más grandes.

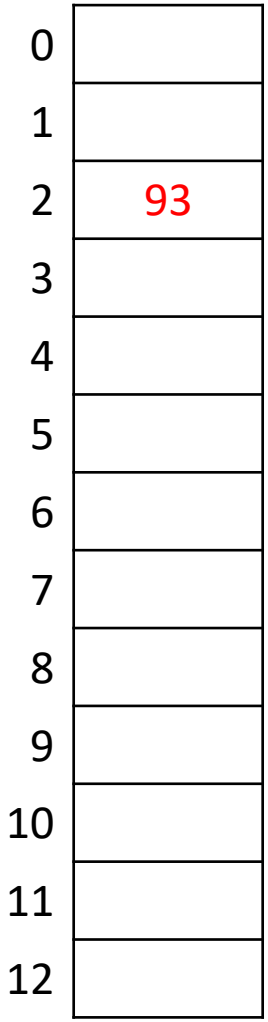
Tablas de dispersión

Hashing cerrado. Exploración lineal

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

Al insertarse en la posicion 2, y alfa = 1,
la secuencia de búsqueda sería 2,3,4,..,12



Tablas de dispersión

Hashing cerrado. Exploración lineal

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
31	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4

0	
1	
2	93
3	
4	
5	31
6	
7	
8	
9	
10	
11	
12	

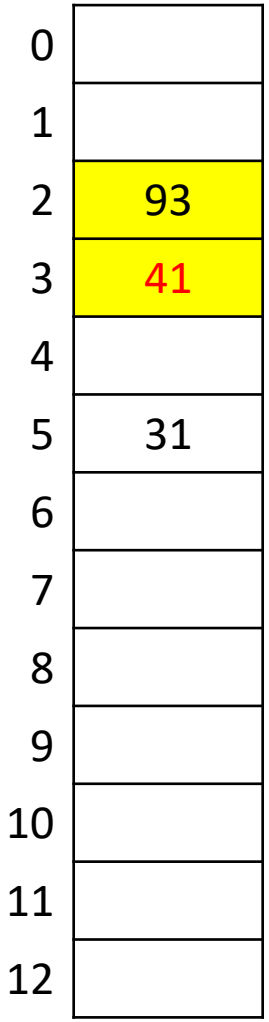
Tablas de dispersión

Hashing cerrado. Exploración lineal

- $h(x) = x \; mod \; 13$

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
31	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

Como 41 mon 13 = 2 y este está ocupado, se produce una sinonimia = colisión, como la secuencia de exploración es 1, se introduce en el 3



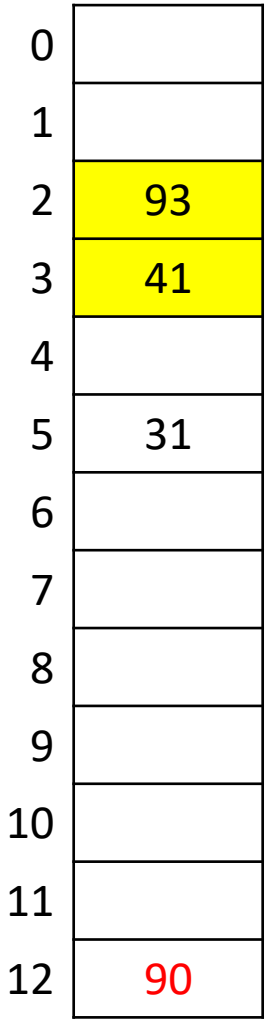
Agrupamiento primario

Tablas de dispersión

Hashing cerrado. Exploración lineal

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
31	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
90	12	12, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11



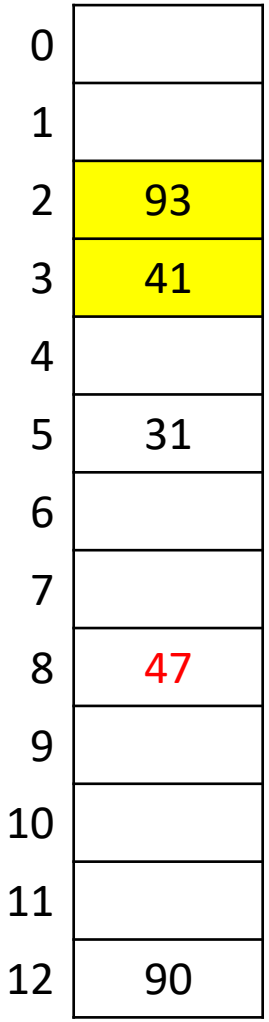
Agrupamiento primario

Tablas de dispersión

Hashing cerrado. Exploración lineal

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
31	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
90	12	12, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
47	8	8, 9, 10, 11, 12, 0, 1, 2, 3, 4, 5, 6, 7



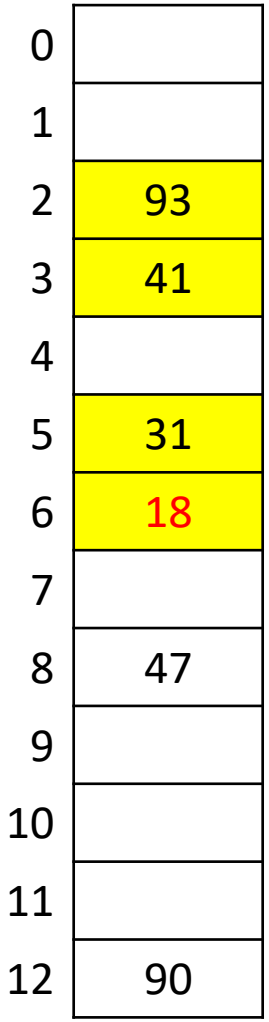
Agrupamiento primario

Tablas de dispersión

Hashing cerrado. Exploración lineal

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
31	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
90	12	12, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
47	8	8, 9, 10, 11, 12, 0, 1, 2, 3, 4, 5, 6, 7
18	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4



Agrupamiento primario

Agrupamiento primario

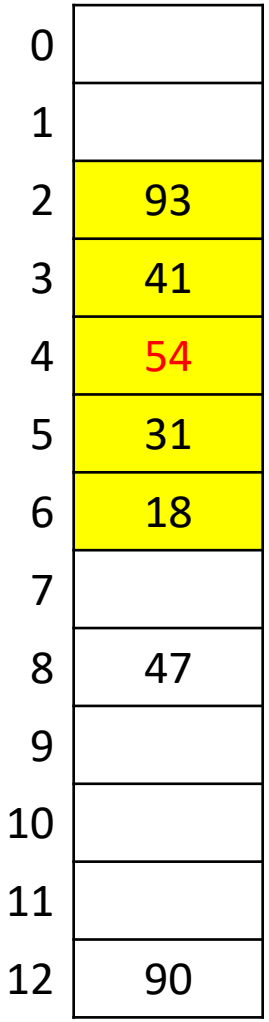
Tablas de dispersión

Hashing cerrado. Exploración lineal

- $h(x) = x \ mod \ 13$

Encontramos un agrupamiento primario, como 54 mod 13 = 2 (ocupado), va al 3 (ocupado), va al 4 (libre)
Por tanto, cualquier clave que de entre 2|6, va a tener que recorrer hasta 7 que es el siguiente libre.

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
31	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
90	12	12, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
47	8	8, 9, 10, 11, 12, 0, 1, 2, 3, 4, 5, 6, 7
18	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
54	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1



Agrupamiento primario

Tabla de dispersión

Hashing cerrado

- **Exploración cuadrática**
  - $\boldsymbol{h_i = (h_0 + i^2)\ mod\ M}$
  - La distancia entre las celdas exploradas aumenta linealmente, para lo que se puede usar cualquier función cuadrática, aunque funciones más complejas no mejoran significativamente los resultados.
  - Difícil encontrar una función de exploración que recorra toda la tabla, de no ser así, es posible que no se pueda insertar una clave en concreto a pesar de quedar posiciones libres. No obstante, si M es primo, los $M/2$ primeros valores generan localizaciones distintas, por tanto se resolverán las colisiones siempre que esté ocupada menos de la mitad de la tabla. Por ejemplo:

    $$M = 4091 \Rightarrow h_1 \neq h_2 \neq \cdots \neq h_{2045} = h_{2046}$$
    $$M = 4092 \Rightarrow h_1 \neq h_2 \neq \cdots \neq h_{63}, h_2 = h_{64}$$
  - Produce **agrupamientos secundarios** (a distancia no fija) de claves sinónimas debido a que la secuencia de exploración desde h_0 (+1, +4, +9, +16,…) sigue siendo idéntica para todas las claves.
  - Los agrupamientos secundarios se entrelazan o solapan pero no se unen para formar otros mayores, por lo que son menos perjudiciales para la eficiencia que los primarios.

Tablas de dispersión

Hashing cerrado. Exploración cuadrática

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración $(h(x) + i^2) \bmod 13$
93	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3

0	
1	
2	93
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Tablas de dispersión

Hashing cerrado. Exploración cuadrática

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración $(h(x) + i^2) \bmod 13$
93	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
31	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6

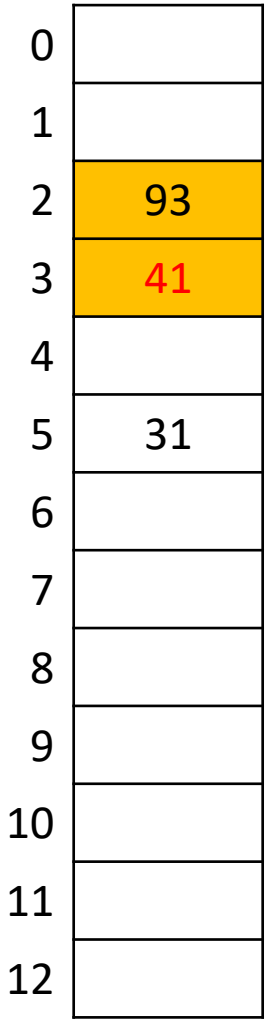
0	
1	
2	93
3	
4	
5	31
6	
7	
8	
9	
10	
11	
12	

Tablas de dispersión

Hashing cerrado. Exploración cuadrática

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración $(h(x) + i^2) \bmod 13$
93	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
31	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6
41	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3



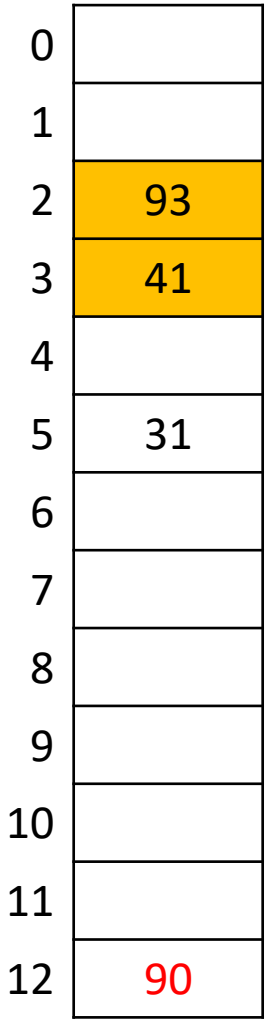
Agrupamiento secundario

Tablas de dispersión

Hashing cerrado. Exploración cuadrática

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración $(h(x) + i^2) \bmod 13$
93	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
31	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6
41	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
90	12	12, 0, 3, 8, 2, 11, 9, 9, 11, 2, 8, 3, 0



Agrupamiento secundario

Tablas de dispersión

Hashing cerrado. Exploración cuadrática

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración $(h(x) + i^2) \bmod 13$
93	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
31	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6
41	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
90	12	12, 0, 3, 8, 2, 11, 9, 9, 11, 2, 8, 3, 0
47	8	8, 9, 12, 4, 11, 7, 5, 5, 7, 11, 4, 12, 9

0	
1	
2	93
3	41
4	
5	31
6	
7	
8	47
9	
10	
11	
12	90

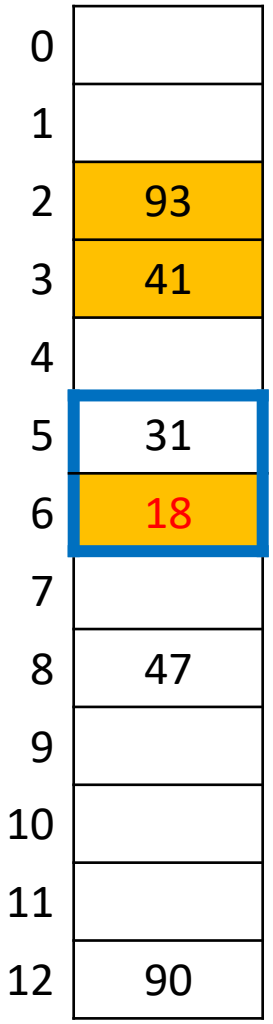
Agrupamiento secundario

Tablas de dispersión

Hashing cerrado. Exploración cuadrática

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración $(h(x) + i^2) \bmod 13$
93	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
31	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6
41	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
90	12	12, 0, 3, 8, 2, 11, 9, 9, 11, 2, 8, 3, 0
47	8	8, 9, 12, 4, 11, 7, 5, 5, 7, 11, 4, 12, 9
18	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6



Agrupamiento secundario

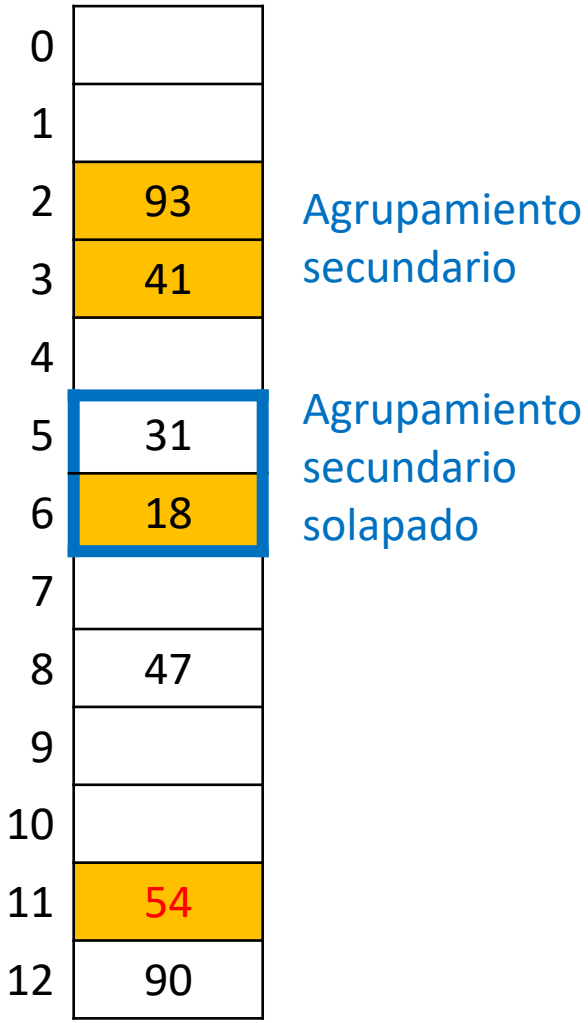
Agrupamiento secundario solapado

Tablas de dispersión

Hashing cerrado. Exploración cuadrática

- $h(x) = x \ mod \ 13$

x	h(x)	Secuencia de exploración $(h(x) + i^2) \bmod 13$
93	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
31	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6
41	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3
90	12	12, 0, 3, 8, 2, 11, 9, 9, 11, 2, 8, 3, 0
47	8	8, 9, 12, 4, 11, 7, 5, 5, 7, 11, 4, 12, 9
18	5	5, 6, 9, 1, 8, 4, 2, 2, 4, 8, 1, 9, 6
54	2	2, 3, 6, 11, 5, 1, 12, 12, 1, 5, 11, 6, 3



Tablas de dispersión

Hashing cerrado

- **Exploración aleatoria**
  - $\boldsymbol{h_i = (h_0 + x_i)\ mod\ M}$

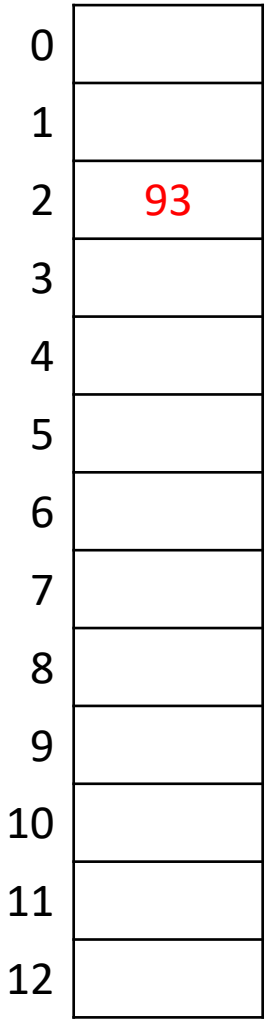
    donde $\boldsymbol{x_i}$ se obtiene con un generador de números pseudoaleatorios (GNA).
  - El GNA debe producir números con una distribución uniforme (todas las direcciones de la tabla deben tener igual probabilidad de ser generadas) y poseer un periodo suficientemente grande (mayor que el tamaño de la tabla, M) para permitir recorrer todas las posiciones.
  - La reproducción de la secuencia de exploración al buscar una clave en la tabla está garantizada, siempre que se use la misma semilla del GNA que cuando se insertó.
  - **Evita los agrupamientos de claves sinónimas** si se usa una semilla distinta para cada clave (puede ser la propia clave o una que dependa de ella), de modo que se generen secuencias de exploración diferentes.

Tablas de dispersión

Hashing cerrado. Exploración aleatoria

- $h(x) = x \ mod \ 13$

x	h(x)	srand(x+1)	Secuencia de exploración (h(x) + rand()) mod 13
93	2	srand(94)	2, 2, 9, 5, 10, 9, 12, …



Tablas de dispersión

Hashing cerrado. Exploración aleatoria

- $h(x) = x \ mod \ 13$

x	h(x)	srand(x+1)	Secuencia de exploración (h(x) + rand()) mod 13
93	2	srand(94)	2, 2, 9, 5, 10, 9, 12, …
31	5	srand(32)	5, 9, 5, 6, 9, 4, 4, …

0	
1	
2	93
3	
4	
5	31
6	
7	
8	
9	
10	
11	
12	

Tablas de dispersión

Hashing cerrado. Exploración aleatoria

- $h(x) = x \ mod \ 13$

x	h(x)	srand(x+1)	Secuencia de exploración (h(x) + rand()) mod 13
93	2	srand(94)	2, 2, 9, 5, 10, 9, 12, …
31	5	srand(32)	5, 9, 5, 6, 9, 4, 4, …
41	2	srand(42)	2, 0, 0, 2, 9, 12, 7, …

0	41
1	
2	93
3	
4	
5	31
6	
7	
8	
9	
10	
11	
12	

Tablas de dispersión

Hashing cerrado. Exploración aleatoria

- $h(x) = x \ mod \ 13$

x	h(x)	srand(x+1)	Secuencia de exploración (h(x) + rand()) mod 13
93	2	srand(94)	2, 2, 9, 5, 10, 9, 12, …
31	5	srand(32)	5, 9, 5, 6, 9, 4, 4, …
41	2	srand(42)	2, 0, 0, 2, 9, 12, 7, …
90	12	srand(91)	12, 8, 0, 6, 4, 3, 6, …

0	41
1	
2	93
3	
4	
5	31
6	
7	
8	
9	
10	
11	
12	90

Tablas de dispersión

Hashing cerrado. Exploración aleatoria

- $h(x) = x \ mod \ 13$

x	h(x)	srand(x+1)	Secuencia de exploración (h(x) + rand()) mod 13
93	2	srand(94)	2, 2, 9, 5, 10, 9, 12, …
31	5	srand(32)	5, 9, 5, 6, 9, 4, 4, …
41	2	srand(42)	2, 0, 0, 2, 9, 12, 7, …
90	12	srand(91)	12, 8, 0, 6, 4, 3, 6, …
47	8	srand(48)	8, 10, 9, 8, 7, 11, 7, …

0	41
1	
2	93
3	
4	
5	31
6	
7	
8	47
9	
10	
11	
12	90

Tablas de dispersión

Hashing cerrado. Exploración aleatoria

- $h(x) = x \ mod \ 13$

x	h(x)	srand(x+1)	Secuencia de exploración (h(x) + rand()) mod 13
93	2	srand(94)	2, 2, 9, 5, 10, 9, 12, …
31	5	srand(32)	5, 9, 5, 6, 9, 4, 4, …
41	2	srand(42)	2, 0, 0, 2, 9, 12, 7, …
90	12	srand(91)	12, 8, 0, 6, 4, 3, 6, …
47	8	srand(48)	8, 10, 9, 8, 7, 11, 7, …
18	5	srand(19)	5, 0, 6, 5, 6, 4, 12, …

0	41
1	
2	93
3	
4	
5	31
6	18
7	
8	47
9	
10	
11	
12	90

Tablas de dispersión

Hashing cerrado. Exploración aleatoria

- $h(x) = x \ mod \ 13$

x	h(x)	srand(x+1)	Secuencia de exploración (h(x) + rand()) mod 13
93	2	srand(94)	2, 2, 9, 5, 10, 9, 12, …
31	5	srand(32)	5, 9, 5, 6, 9, 4, 4, …
41	2	srand(42)	2, 0, 0, 2, 9, 12, 7, …
90	12	srand(91)	12, 8, 0, 6, 4, 3, 6, …
47	8	srand(48)	8, 10, 9, 8, 7, 11, 7, …
18	5	srand(19)	5, 0, 6, 5, 6, 4, 12, …
54	2	srand(55)	2, 7, 5, 0, 9, 3, 10, …

0	41
1	
2	93
3	
4	
5	31
6	18
7	54
8	47
9	
10	
11	
12	90

Tablas de dispersión

Hashing cerrado

- ***Hashing* doble**
  - $\boldsymbol{h_i = (h_0 + i \cdot h') \bmod M}$

    donde h' es una segunda función *hash*.
  - Método parecido a la exploración lineal, pero con una diferencia fundamental: la distancia entre las celdas exploradas es variable, puesto que se rastrea con incrementos de h' posiciones, un valor dependiente de cada clave.
  - **Evita los agrupamientos de claves sinónimas** debido a que la secuencia de exploración es distinta para todas las claves.
  - La función *hash* secundaria debe cumplir algunas condiciones:
    - ✓ $h'(x) \neq 0$ para cualquier clave x, de lo contrario no sirve para resolver colisiones, pues todas las funciones de exploración coincidirían con la función *hash* original, h_0.
    - ✓ $h' \neq h_0$. Si ambas funciones fueran iguales, la secuencia de exploración sería idéntica para claves sinónimas y provocaría agrupamientos primarios (mismo caso de la exploración lineal).
    - ✓ $h'(x)$ y M deben ser primos relativos para garantizar que se recorre toda la tabla. Si M es primo, sirve cualquier función *hash* h' que devuelva valores en el rango $[1, M']$, con $M' < M$.

Tablas de dispersión

Hashing cerrado

- ***Hashing* doble**

  Ejemplos de funciones *hash* secundarias:

  Sea M' un primo menor que M,

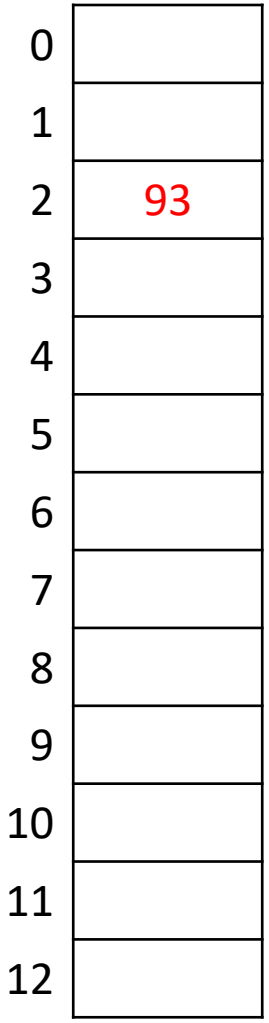
  - $h'(x) = M' - (x \ mod \ M')$
  - $h'(x) = 1 + (x \ mod \ M')$

Tablas de dispersión

Hashing cerrado. *Hashing* doble

- $h(x) = x \ mod \ 13$
- $h'(x) = 11 - (x \ mod \ 11)$

x	h(x)	h'(x)	Secuencia de exploración (h(x) + i · h'(x)) mod 13
93	2	6	2, 8, 1, 7, 0, 6, 12, 5, 11, 4, 10, 3, 9

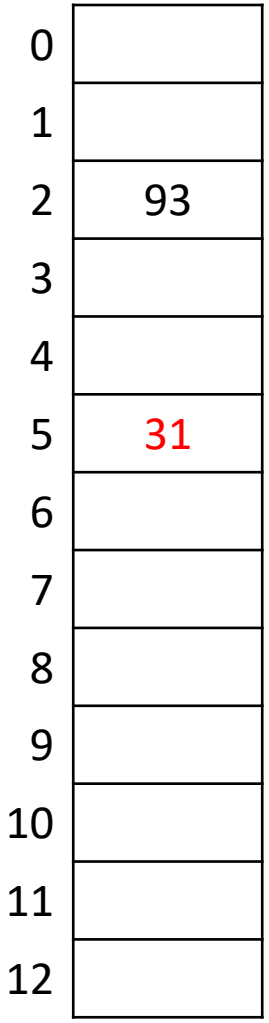


Tablas de dispersión

Hashing cerrado. *Hashing* doble

- $h(x) = x \ mod \ 13$
- $h'(x) = 11 - (x \ mod \ 11)$

x	h(x)	h'(x)	Secuencia de exploración (h(x) + i · h'(x)) mod 13
93	2	6	2, 8, 1, 7, 0, 6, 12, 5, 11, 4, 10, 3, 9
31	5	2	5, 7, 9, 11, 0, 2, 4, 6, 8, 10, 12, 1, 3

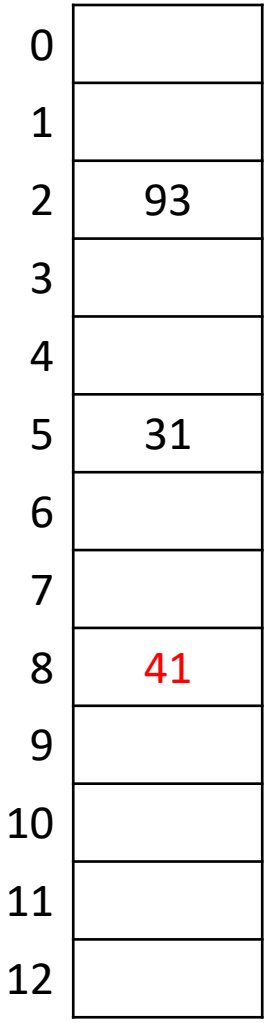


Tablas de dispersión

Hashing cerrado. *Hashing* doble

- $h(x) = x \ mod \ 13$
- $h'(x) = 11 - (x \ mod \ 11)$

x	h(x)	h'(x)	Secuencia de exploración (h(x) + i · h'(x)) mod 13
93	2	6	2, 8, 1, 7, 0, 6, 12, 5, 11, 4, 10, 3, 9
31	5	2	5, 7, 9, 11, 0, 2, 4, 6, 8, 10, 12, 1, 3
41	2	3	2, 5, 8, 11, 1, 4, 7, 10, 0, 3, 6, 9, 12



Tablas de dispersión

Hashing cerrado. *Hashing* doble

- $h(x) = x \ mod \ 13$
- $h'(x) = 11 - (x \ mod \ 11)$

x	h(x)	h'(x)	Secuencia de exploración (h(x) + i · h'(x)) mod 13
93	2	6	2, 8, 1, 7, 0, 6, 12, 5, 11, 4, 10, 3, 9
31	5	2	5, 7, 9, 11, 0, 2, 4, 6, 8, 10, 12, 1, 3
41	2	3	2, 5, 8, 11, 1, 4, 7, 10, 0, 3, 6, 9, 12
90	12	9	12, 8, 4, 0, 9, 5, 1, 10, 6, 2, 11, 7, 3

0	
1	
2	93
3	
4	
5	31
6	
7	
8	41
9	
10	
11	
12	90

Tablas de dispersión

Hashing cerrado. *Hashing* doble

- $h(x) = x \ mod \ 13$
- $h'(x) = 11 - (x \ mod \ 11)$

x	h(x)	h'(x)	Secuencia de exploración (h(x) + i · h'(x)) mod 13
93	2	6	2, 8, 1, 7, 0, 6, 12, 5, 11, 4, 10, 3, 9
31	5	2	5, 7, 9, 11, 0, 2, 4, 6, 8, 10, 12, 1, 3
41	2	3	2, 5, 8, 11, 1, 4, 7, 10, 0, 3, 6, 9, 12
90	12	9	12, 8, 4, 0, 9, 5, 1, 10, 6, 2, 11, 7, 3
47	8	8	8, 3, 11, 6, 1, 9, 4, 12, 7, 2, 10, 5, 0

0	
1	
2	93
3	47
4	
5	31
6	
7	
8	41
9	
10	
11	
12	90

Tablas de dispersión

Hashing cerrado. *Hashing* doble

- $h(x) = x \ mod \ 13$
- $h'(x) = 11 - (x \ mod \ 11)$

x	h(x)	h'(x)	Secuencia de exploración (h(x) + i · h'(x)) mod 13
93	2	6	2, 8, 1, 7, 0, 6, 12, 5, 11, 4, 10, 3, 9
31	5	2	5, 7, 9, 11, 0, 2, 4, 6, 8, 10, 12, 1, 3
41	2	3	2, 5, 8, 11, 1, 4, 7, 10, 0, 3, 6, 9, 12
90	12	9	12, 8, 4, 0, 9, 5, 1, 10, 6, 2, 11, 7, 3
47	8	8	8, 3, 11, 6, 1, 9, 4, 12, 7, 2, 10, 5, 0
18	5	4	5, 9, 0, 4, 8, 12, 3, 7, 11, 2, 6, 10, 1

0	
1	
2	93
3	47
4	
5	31
6	
7	
8	41
9	18
10	
11	
12	90

Tablas de dispersión

Hashing cerrado. *Hashing* doble

- $h(x) = x \ mod \ 13$
- $h'(x) = 11 - (x \ mod \ 11)$

x	h(x)	h'(x)	Secuencia de exploración (h(x) + i · h'(x)) mod 13
93	2	6	2, 8, 1, 7, 0, 6, 12, 5, 11, 4, 10, 3, 9
31	5	2	5, 7, 9, 11, 0, 2, 4, 6, 8, 10, 12, 1, 3
41	2	3	2, 5, 8, 11, 1, 4, 7, 10, 0, 3, 6, 9, 12
90	12	9	12, 8, 4, 0, 9, 5, 1, 10, 6, 2, 11, 7, 3
47	8	8	8, 3, 11, 6, 1, 9, 4, 12, 7, 2, 10, 5, 0
18	5	4	5, 9, 0, 4, 8, 12, 3, 7, 11, 2, 6, 10, 1
54	2	1	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

0	
1	
2	93
3	47
4	54
5	31
6	
7	
8	41
9	18
10	
11	
12	90

Tablas de dispersión

Hashing cerrado. Búsqueda

Búsqueda

Seguir la secuencia de exploración hasta que se cumpla alguna de las siguientes condiciones:

- Se encuentra la clave buscada. Búsqueda con éxito.
- Se encuentra una casilla libre. Búsqueda sin éxito.
- Se han recorrido todas las casillas. Búsqueda sin éxito.

Distinción entre casilla *libre* y *ocupada*:

- Marcar las casillas libres con un valor «ilegal» (uno que no corresponda a ningún elemento).
- Añadir un campo lógico, libre/ocupada.

Tablas de dispersión

Hashing cerrado. Eliminación

- Exploración lineal
- $h(x) = x \; mod \; 13$

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
93	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
31	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1
90	12	12, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
47	8	8, 9, 10, 11, 12, 0, 1, 2, 3, 4, 5, 6, 7
18	5	5, 6, 7, 8, 9, 10, 11, 12, 0, 1, 2, 3, 4
54	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

0	✓
1	✓
2	93
3	41
4	54
5	31
6	18
7	✓
8	47
9	✓
10	✓
11	✓
12	90

Tablas de dispersión

Hashing cerrado. Eliminación

- Exploración lineal
- $h(x) = x \ mod \ 13$
- Eliminar 41

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

0	✓
1	✓
2	93
3	41
4	54
5	31
6	18
7	✓
8	47
9	✓
10	✓
11	✓
12	90

Tablas de dispersión

Hashing cerrado. Eliminación

- Exploración lineal
- $h(x) = x \; mod \; 13$
- Eliminar 41 (marcar casilla libre, ✓)

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

0	✓
1	✓
2	93
3	✓
4	54
5	31
6	18
7	✓
8	47
9	✓
10	✓
11	✓
12	90

Tablas de dispersión

Hashing cerrado. Eliminación

- Exploración lineal
- $h(x) = x \ mod \ 13$
- Eliminar 41 (marcar casilla libre, ✓)

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

- Buscar 54

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
54	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

- Casilla 3 libre (✓). Búsqueda sin éxito. Fallo.

0	✓
1	✓
2	93
3	✓
4	54
5	31
6	18
7	✓
8	47
9	✓
10	✓
11	✓
12	90

Tablas de dispersión

Hashing cerrado. Eliminación

- Exploración lineal
- $h(x) = x \ mod \ 13$
- Eliminar 41 (marcar casilla borrada, ✗)

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

- Distinción entre casilla *libre* (✓), *borrada* (✗) y *ocupada*.

0	✓
1	✓
2	93
3	✗
4	54
5	31
6	18
7	✓
8	47
9	✓
10	✓
11	✓
12	90

Tablas de dispersión

Hashing cerrado. Eliminación

- Exploración lineal
- $h(x) = x \ mod \ 13$
- Eliminar 41 (marcar casilla borrada, ×)

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
41	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

- Distinción entre casilla *libre* (✓), *borrada* (×) y *ocupada*.
- Buscar 54

x	h(x)	Secuencia de exploración (h(x) + i) mod 13
54	2	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 1

- Encontrada clave 54 en casilla 4. Búsqueda con éxito.

0	✓
1	✓
2	93
3	×
4	54
5	31
6	18
7	✓
8	47
9	✓
10	✓
11	✓
12	90

Tablas de dispersión

Hashing cerrado

- **Búsqueda**

  Seguir la secuencia de exploración hasta:
  - encontrar la clave buscada (búsqueda con éxito);
  - encontrar una casilla *libre* (búsqueda sin éxito);

    o bien,
  - haber recorrido toda la tabla (búsqueda sin éxito).

- **Eliminación**

  Seguir la secuencia de exploración hasta:
  - encontrar la clave a eliminar y marcar la casilla como *borrada*;

    o bien,
  - encontrar una casilla *libre* o haber recorrido toda la tabla (la clave no existe).

- **Inserción**

  Seguir la secuencia de exploración hasta:
  - encontrar la clave a insertar (la clave ya existe);
  - encontrar una casilla *libre* y realizar la inserción en ella;

    o bien,
  - haber recorrido toda la tabla y realizar la inserción en una casilla *borrada*, si existe, o si no existe, concluir que la tabla está llena.

Tablas de dispersión

Hashing cerrado

Degradación de la eficiencia de las operaciones

- Los métodos de resolución de colisiones mediante *hashing* cerrado hacen más probables colisiones posteriores:

  Las claves que colisionan se almacenan en direcciones que, en principio, estarían reservadas para otras claves cuyo valor *hash* directo son esas mismas direcciones, por lo que cuando se inserten estas últimas se producirán nuevas colisiones que agravarán el problema.

- En consecuencia, la eficiencia de las operaciones se degrada rápidamente a medida que se va llenando la tabla.

Tablas de dispersión

Hashing abierto

- En una **tabla *hash* abierta** (también llamada con direccionamiento cerrado o encadenamiento separado) cada posición es un cubículo (*bucket*) con una estructura adecuada para alojar varios elementos que tienen claves sinónimas.
- La estructura más sencilla consiste en una **tabla de M listas enlazadas**, cuya longitud media, $\frac{n}{M}$ (siendo *n* el número de claves insertadas), determina el tiempo medio de búsqueda, puesto que es de esperar que la función *hash* distribuya las claves uniformemente y no las acumule en unos pocos cubículos.
- Seleccionando M suficientemente grande las listas serán bastante cortas y, en consecuencia, no supone una mejora importante de la eficiencia de la búsqueda mantener las listas ordenadas o sustituir las listas por árboles de búsqueda.
- La búsqueda o eliminación de un elemento con clave k se reduce a buscar o eliminar dicho elemento en la lista $h(k)$.
- La inserción se puede realizar eficientemente al principio de la lista, una vez comprobado que el elemento no está.

Tablas de dispersión

Hashing abierto

- $h(x) = x \; mod \; 13$

x	h(x)
93	2

0	()
1	()
2	(93)
3	()
4	()
5	()
6	()
7	()
8	()
9	()
10	()
11	()
12	()

Tablas de dispersión

Hashing abierto

- $h(x) = x \; mod \; 13$

x	h(x)
93	2
31	5

0	()
1	()
2	(93)
3	()
4	()
5	(31)
6	()
7	()
8	()
9	()
10	()
11	()
12	()

Tablas de dispersión

Hashing abierto

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5
41	2

0	()
1	()
2	(41, 93)
3	()
4	()
5	(31)
6	()
7	()
8	()
9	()
10	()
11	()
12	()

Tablas de dispersión

Hashing abierto

- $h(x) = x \; mod \; 13$

x	h(x)
93	2
31	5
41	2
90	12

0	()
1	()
2	(41, 93)
3	()
4	()
5	(31)
6	()
7	()
8	()
9	()
10	()
11	()
12	(90)

Tablas de dispersión

Hashing abierto

- $h(x) = x \; mod \; 13$

x	h(x)
93	2
31	5
41	2
90	12
47	8

0	()
1	()
2	(41, 93)
3	()
4	()
5	(31)
6	()
7	()
8	(47)
9	()
10	()
11	()
12	(90)

Tablas de dispersión

Hashing abierto

- $h(x) = x \; mod \; 13$

x	h(x)
93	2
31	5
41	2
90	12
47	8
18	5

0	()
1	()
2	(41, 93)
3	()
4	()
5	(18, 31)
6	()
7	()
8	(47)
9	()
10	()
11	()
12	(90)

Tablas de dispersión

Hashing abierto

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5
41	2
90	12
47	8
18	5
54	2

0	()
1	()
2	(54, 41, 93)
3	()
4	()
5	(18, 31)
6	()
7	()
8	(47)
9	()
10	()
11	()
12	(90)

Tablas de dispersión

Encadenamiento mezclado

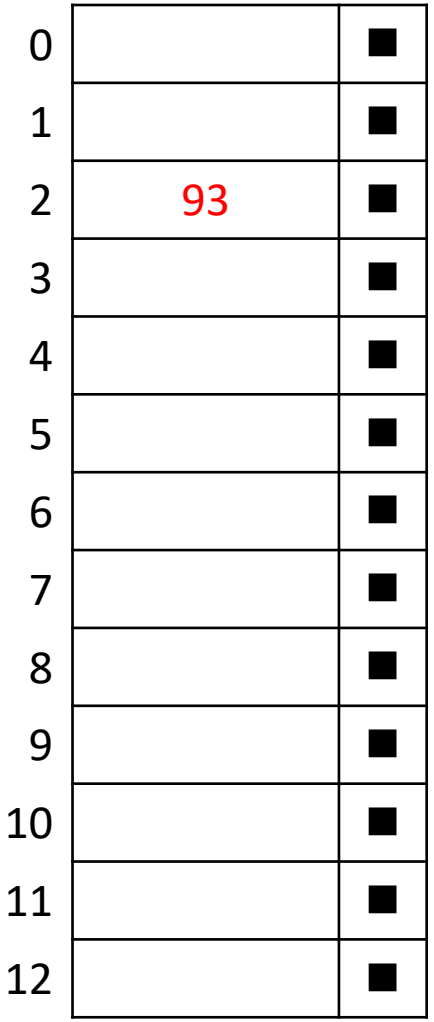
- Combinación de los métodos anteriores donde las claves se insertan en casillas libres de la tabla (como en el *hashing* cerrado), pero aquellas que la función *hash* lleva a casillas ocupadas se enlazan (al igual que en el *hashing* abierto) formando listas dentro de la tabla.
- A diferencia del *hashing* abierto, cada lista de casillas enlazadas puede contener una mezcla de claves sinónimas y no sinónimas.
- A pesar de que la tabla esté casi llena, las búsquedas pueden ser mucho más rápidas que en el *hashing* cerrado, puesto que sólo habrá que rastrear el fragmento de lista que comienza en la casilla $h(k)$ para localizar la clave k.
- Por el contrario, no es tan rápido buscar como en una tabla *hash* abierta, ya que la lista a recorrer puede ser más larga por contener una mezcla de claves no sinónimas.
- Con objeto de acercar la eficiencia a la del *hashing* abierto, existen variantes del método de encadenamiento mezclado que dividen la tabla en dos zonas: la principal, donde la función *hash* direcciona las claves, y otra de desbordamiento para los elementos que colisionan. La zona principal también es utilizada para las colisiones cuando la zona de desbordamiento se llena; mientras esto no ocurra, cada lista contendrá únicamente claves sinónimas, igual que en el *hashing* abierto.

Tablas de dispersión

Encadenamiento mezclado

- $h(x) = x \ mod \ 13$

x	h(x)
93	2

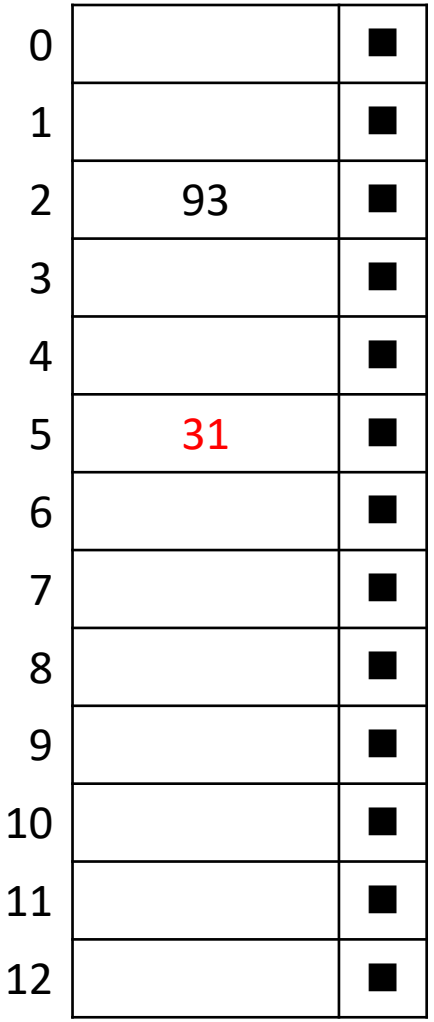


Tablas de dispersión

Encadenamiento mezclado

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5



Tablas de dispersión

Encadenamiento mezclado

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5
41	2

0		■
1		■
2	93	12
3		■
4		■
5	31	■
6		■
7		■
8		■
9		■
10		■
11		■
12	41	■

Tablas de dispersión

Encadenamiento mezclado

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5
41	2
90	12

0		■
1		■
2	93	12
3		■
4		■
5	31	■
6		■
7		■
8		■
9		■
10		■
11	90	■
12	41	11

Tablas de dispersión

Encadenamiento mezclado

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5
41	2
90	12
47	8

0		■
1		■
2	93	12
3		■
4		■
5	31	■
6		■
7		■
8	47	■
9		■
10		■
11	90	■
12	41	11

Tablas de dispersión

Encadenamiento mezclado

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5
41	2
90	12
47	8
18	5

0		■
1		■
2	93	12
3		■
4		■
5	31	10
6		■
7		■
8	47	■
9		■
10	18	■
11	90	■
12	41	11

Tablas de dispersión

Encadenamiento mezclado

- $h(x) = x \ mod \ 13$

x	h(x)
93	2
31	5
41	2
90	12
47	8
18	5
54	2

0		■
1		■
2	93	12
3		■
4		■
5	31	10
6		■
7		■
8	47	■
9	54	■
10	18	■
11	90	9
12	41	11

Tablas de dispersión
Eficiencia

- **Caso mejor**: no hay colisiones, hay una clave en cada casilla de la tabla.
  - El tiempo de búsqueda lo determina el cálculo de la función *hash*, que es independiente del número de elementos almacenados, por tanto, $O(1)$.
- **Caso peor**: todos los elementos almacenados colisionan.
  - Búsqueda secuencial, $O(n)$.
  - Caso muy improbable si disponemos de una buena función *hash*.
- **Caso promedio**:
  - Supondremos que se inserta una serie de n elementos aleatorios y que la función *hash* los distribuye de manera uniforme por toda la tabla.
  - El **número de comparaciones** al buscar un elemento depende de si está o no en la tabla.

    Ejemplo: buscar la clave k en una tabla *hash* abierta.

    Búsqueda sin éxito: se debe recorrer la lista $h(k)$ completa.

    Búsqueda con éxito: en promedio se recorrerá la mitad de la lista $h(k)$.
  - El tiempo de búsqueda también depende del **factor de carga** de la tabla: relación entre el número de elementos almacenados y el tamaño de la tabla, $\alpha = \frac{n}{M}$.

    Ejemplo: búsqueda sin éxito en una tabla *hash* cerrada.

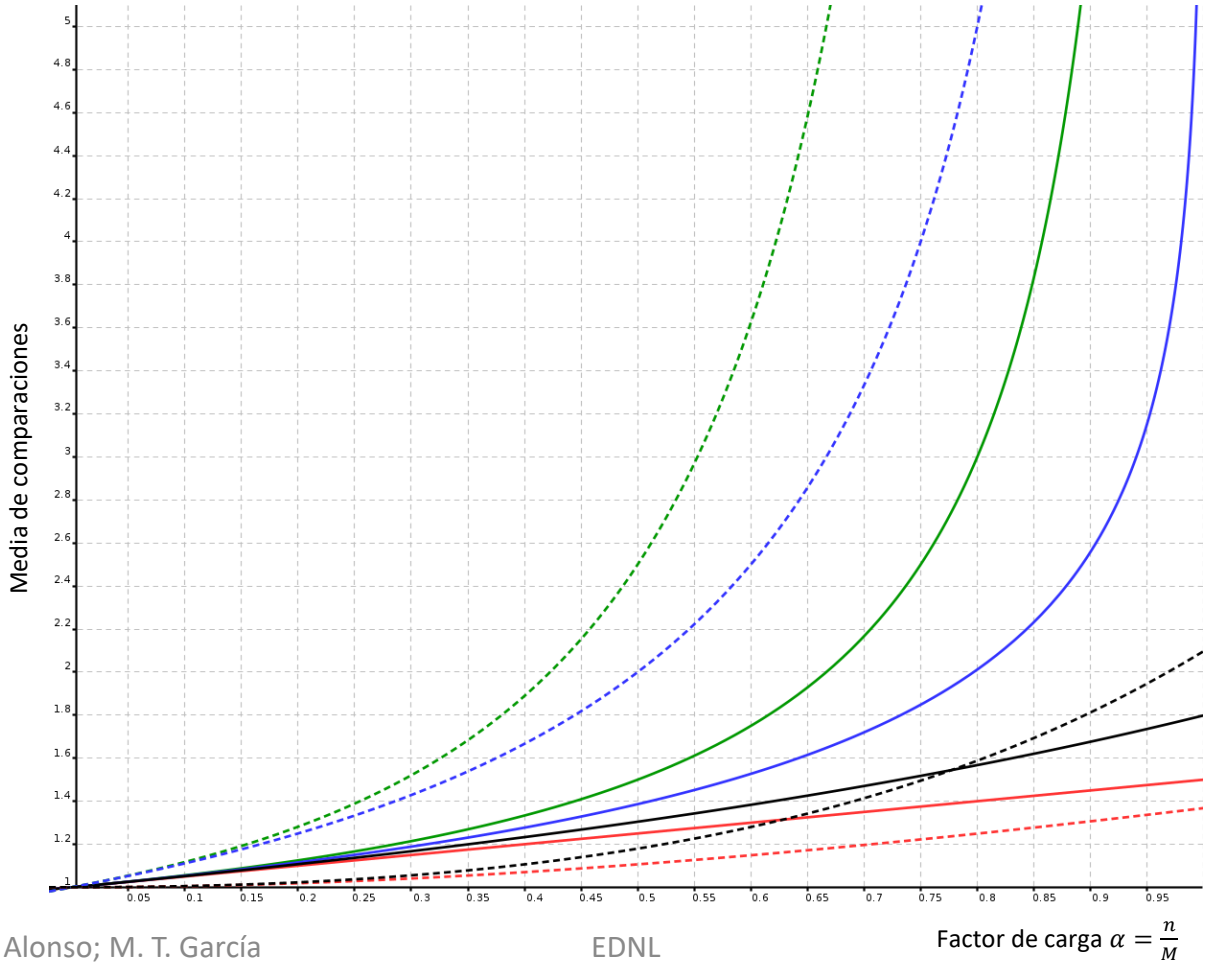
    Tabla llena: se recorrerá la tabla completa.

    Tabla medio llena: en el peor caso se recorrerá la mitad de la tabla.
  - En la siguiente diapositiva se muestran los resultados del análisis en el caso promedio para los diferentes métodos de resolución de colisiones. El gráfico representa la eficiencia de las operaciones, medida en número de comparaciones, en función del factor de carga.

Tablas de dispersión

Eficiencia

Elim./Búsq. con éxito —— Inser./Búsq. sin éxito ------



Exploración lineal

Exploración cuadrática/aleatoria/ hashing doble

Encadenamiento mezclado

Hashing abierto

Tablas de dispersión

Comparación entre métodos

- En general, si el factor de carga se mantiene por debajo de un umbral, el número medio de comparaciones está acotado por una constante y la búsqueda tiene un coste promedio $O(1)$.
- El *hashing* cerrado requiere prever el máximo de elementos a almacenar, pero además, hay que tener en cuenta que el rendimiento se degrada muy rápido a medida que la tabla se va llenando. Posibles soluciones: redimensionamiento y *rehashing* si el factor de carga excede un límite.
- El *hashing* cerrado requiere un tamaño de la tabla mayor que el *hashing* abierto, porque se necesita espacio extra que ayude a resolver las colisiones, pero la cantidad total de memoria usada es menor, ya que no se necesitan los punteros de las listas.
- La exploración lineal es el método menos eficiente, con una curva de crecimiento muy pronunciada, como cabía esperar, debido al rápido deterioro de la eficiencia que provocan los agrupamientos primarios. Aunque esto no lo descarta, pues puede resultar suficiente para algunas aplicaciones, sobre todo considerando que la implementación no es complicada.
- La exploración cuadrática tiene una eficiencia equiparable a la exploración aleatoria o al *hashing* doble, lo que nos confirma que los agrupamientos secundarios no afectan tanto como los primarios.

Tablas de dispersión

Comparación entre métodos

- La exploración cuadrática, aleatoria o el *hashing* doble requieren en media menos comparaciones para buscar una clave que la exploración lineal, lo que significa que se puede usar una tabla de menor tamaño (no con tanto espacio extra) para obtener los mismos tiempos de búsqueda. Por ejemplo, se necesitan menos de 5 comparaciones en media para una búsqueda con y sin éxito si la tabla está ocupada en menos del 99% y 80%, respectivamente, frente al 88% y 66% con exploración lineal.
- La eficiencia del encadenamiento mezclado está próxima a la del *hashing* abierto, debido a la mejora que aporta respecto al *hashing* cerrado la separación de las claves en listas diferentes para reducir el número de comparaciones.
- El *hashing* abierto es el que refleja mayor eficiencia, pero a costa de consumir más memoria con los punteros de las listas. Incluso se puede disminuir el tiempo promedio de las búsquedas sin éxito a la mitad manteniendo las listas ordenadas, aunque no se apreciará mucho la ganancia, dado que dicho tiempo ya es muy bajo.
- La eficiencia temporal no es el único factor a considerar para seleccionar uno de los diversos métodos, también hay que tener en cuenta otros como son la memoria necesaria o la complejidad de la implementación.

Tablas de dispersión

Redimensionamiento y *rehashing*

- Recuperación de la eficiencia cuando el factor de carga supera cierto umbral, por ejemplo:
  - En hashing cerrado, si $\alpha \geq 0{,}9$
  - En hashing abierto, si $\alpha \geq 2$
- **Redimensionamiento**: creación de una nueva tabla *hash* de mayor tamaño (por ejemplo, el doble) en la que se vuelven a insertar los elementos actuales.
  - La inserción de todos los elementos en la tabla nueva llevará, en promedio, menos tiempo que el que se necesitó para almacenarlos en la tabla antigua, aún así es una tarea muy costosa, pero se compensará con la mayor eficiencia de las operaciones posteriores en la nueva tabla.
- **Rehashing**: reinserción de los elementos de una tabla *hash* cerrada para suprimir las casillas marcadas como borradas.
  - Recordemos que las casillas borradas se pueden reutilizar en las inserciones, pero también se deben recorrer en las búsquedas para asegurar que el algoritmo es correcto.
  - Por tanto, en una tabla *hash* cerrada con un factor de carga bajo, pero con un porcentaje alto de casillas borradas, las búsquedas serán ineficientes.
  - Un *rehashing* mejorará la eficiencia de las operaciones posteriores sin aumentar la memoria ocupada.

Tablas de dispersión

Comparación con árboles de búsqueda

- Las tablas de dispersión son algo más fáciles de implementar y ofrecen tiempos de búsqueda $O(1)$ en media (si se limita el factor de carga), aunque pueden llegar a ser $O(n)$ en el caso peor (altamente improbable).
- La ventaja de los árboles de búsqueda es que son dinámicos (no es necesario prever el número máximo de elementos a almacenar) y la eficiencia de los equilibrados está garantizada en $O(\log n)$.
- Un árbol de búsqueda puede necesitar demasiado espacio para los punteros, especialmente si los elementos que se almacenan en los nodos son pequeños.
- Los árboles de búsqueda ofrecen un conjunto de operaciones más amplio, la más importante es el acceso a los elementos en orden, la cual permite realizar otras relacionadas con el orden como obtener el mínimo o el máximo, los menores o mayores que cierto valor o, por poner otro ejemplo, buscar el menor elemento mayor que cierto valor.
- **Conclusión**: si lo que prima en una aplicación es la velocidad de búsqueda, la elección correcta es una tabla de dispersión con una buena función *hash* y un adecuado control del factor de carga. No obstante, si necesitamos los elementos ordenados, debemos seleccionar un árbol de búsqueda para almacenar los datos.