3 Tablas de Dispersión

Estructura de datos para gestionar colecciones de elementos

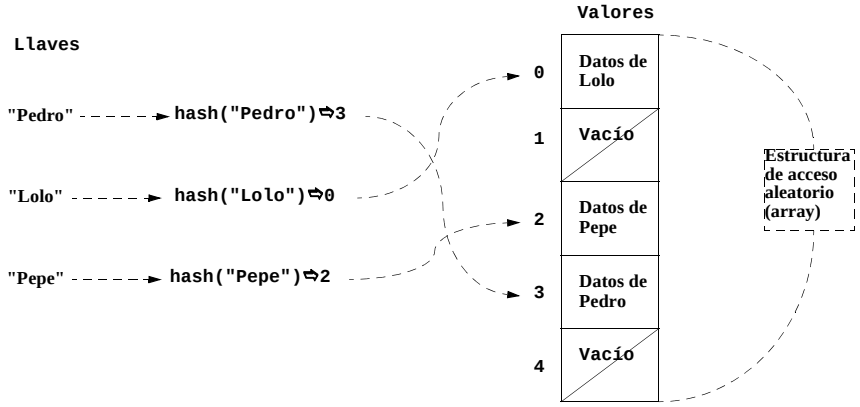
- **donde la búsqueda de un elemento puede hacerse en *O*(1) en un alto porcentaje de casos**
  - **mejora el tiempo de búsqueda en listas: *O*(n) (o *O*(log n) si la lista está ordenada)**

Constituyen la generalización del concepto de "array"

- **asocian cada *llave* (en array: cada valor del índice)**
- **con su *valor* correspondiente (en array: elemento que ocupa la posición indicada por el índice)**

También llamadas *tablas hash*

Esquema básico de una Tabla de Dispersión



hash( ): función de dispersión (también llamada "función *hash*")

Función de dispersión

Función que hace corresponder cada *llave* con un número entero

- **que es el *código de dispersión* (*hashcode*) de esa llave**

Características de una buena función de dispersión:

- **El código de dispersión debe ser un número dentro del rango de índices del array**
  - **suele utilizarse el operador módulo (% en Java) para lograrlo**
- **Debe generar una dispersión uniforme de las llaves en el array**
- **Debe poder computarse de manera muy eficiente**

Funciones de dispersión cuando la llave es un número entero

Primer método: ***función módulo***

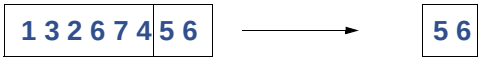
```
hash(llave) = llave % max
```

- **donde `max` es el tamaño de la tabla**

Inconvenientes:

- **sólo se usa la parte menos significativa de la llave**
- **puede producir distribuciones muy poco homogéneas**
  - **cuando alguna de las "terminaciones" de las llaves tiende a repetirse más que otras**

Ejemplo: `llave=13267456, max=100`



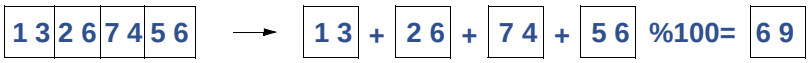
Segundo método: ***troceado en grupos de cifras***

- **divide la llave en n grupos de cifras**
- **el código de dispersión se calcula como el módulo de la suma de esos grupos**

$$\text{hash(llave)} = \left(\sum_{i=0}^{n-1} \left(\frac{\text{llave}}{\text{max}^i} \bmod \text{max} \right) \right) \bmod \text{max}$$

- **+ Ventaja: utiliza todas las cifras de la llave**
- **– Inconveniente: es algo más costosa de calcular**

Ejemplo: `llave=13267456, max=100, n=4`



Funciones de dispersión cuando la llave es una cadena de caracteres

Primer método: ***suma de los códigos ASCII de los caracteres***

$$\text{hash(llave)} = \left(\sum_{i=0}^{\text{length}-1} \text{ascii}(\text{llave}_i) \right) \% \text{ max}$$

- **+ Ventajas: sencilla y eficiente**
- **– Inconveniente: las llaves con los mismos caracteres (aunque estén en distinto orden) generan el mismo código de dispersión**

Ejemplo: `llave="hola", max=100`

```
hash("hola") = (ascii('h') + ascii('o')
              + ascii('l') + ascii('a')) % 100
    = (104 + 111 + 108 + 97) % 100 = 20
```

Funciones de dispersión cuando la llave es una cadena de caracteres (cont.)

Segundo método: *suma con pesos de los códigos ASCII de los caracteres*

- **al código ASCII de cada carácter se le suma un "peso" que depende de la posición que ocupa en la llave**

$$\text{hash(llave)} = \left(\sum_{i=0}^{\text{length}-1} \text{ascii}((\text{llave}_i) \times \text{peso}^i) \right) \quad \% \quad \text{max}$$

+ **Ventaja: el código depende de los caracteres que forman la llave y también del orden que ocupan dichos caracteres**
- **Inconveniente: más costosa de calcular**

Ejemplo: `llave="hola", max=100, peso=37`

```
hash("hola") =
  (ascii('h')*37^3 + ascii('o')*37^2
 + ascii('l')*37^1 + ascii('a')*37^0) % 100
 = (104*50653 + 111*1369 + 108*37 + 97) % 100 = 64
```


Funciones de dispersión cuando la llave es una cadena de caracteres (cont.)

Es posible simplificar el cálculo utilizando la siguiente propiedad:

$$A_3X^3 + A_2X^2 + A_1X^1 + A_0X^0 = (((A_3)X + A_2)X + A_1)X + A_0$$

- **nos ahorramos calcular X^i**

Nos permite representar la función *hash* mediante la siguiente expresión recursiva:

$$\text{hash}(\text{llave}_i) = \begin{cases} \text{ascii}(\text{llave}_0) \\ ((\text{ascii}(\text{llave}_{i-1}) \times \text{max}) + \text{ascii}(\text{llave}_i)) \end{cases}$$

$$\text{hash(llave)} = \text{hash}(\text{llave}_{\text{length}}) \quad \text{mod} \quad \text{max}$$

Para cadenas largas, el resultado obtenido podría superar el valor máximo almacenable en un entero

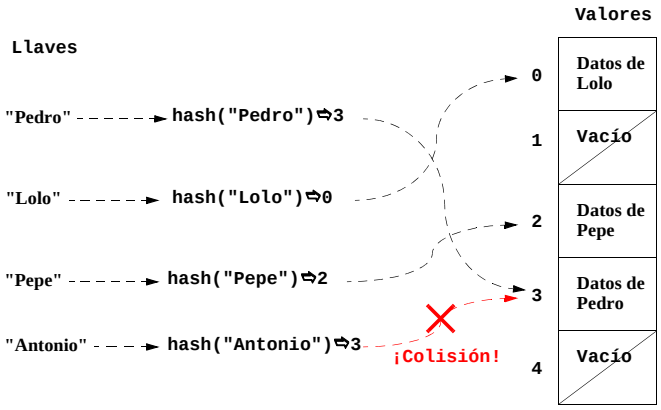
- **en ese caso se podría obtener un resultado negativo**
- **si esto ocurre le sumamos `max` para ponerlo en positivo**

Funciones de dispersión cuando la llave es una cadena de caracteres (cont.)

Ejemplo: cálculo de la función *hash* para cadenas de caracteres

```
public static int hashCode(String s, int max) {
    int cod=0;

    for (int i=0; i<s.length(); i++) {
        cod=(cod*37+ s.charAt(i));
    }

    cod = cod % max;

    if (cod<0) {
        cod=cod+max;
    }

    return cod;
}
```

Problema de las colisiones



Métodos para la resolución de colisiones

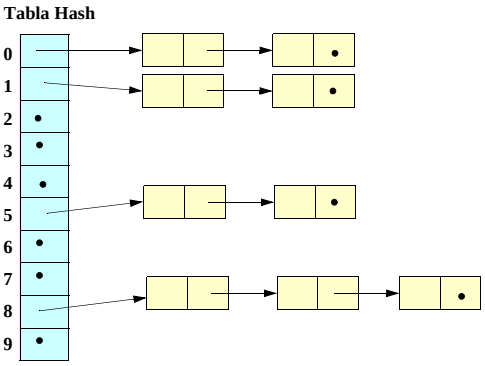
Métodos:

- **encadenamiento o *dispersión abierta***
- **por exploración o *dispersión cerrada***
  - **exploración lineal**
  - **exploración cuadrática**
  - **otras exploraciones**

Su eficiencia depende del grado de ocupación de la tabla

- **dispersión abierta: se puede funcionar bien con una ocupación del 100% (pero no conviene que crezca mucho más)**
- **dispersión cerrada: conviene que nunca se supere el 75% de ocupación**

Resolución de colisiones: dispersión abierta



Cada elemento de la tabla *hash* es una lista enlazada

- **Cada valor se guarda en la lista correspondiente a su código de dispersión**

Este método sólo es eficiente cuando la cantidad de colisiones es pequeña, y las listas son cortas

- **Para ello se necesita que la tabla tenga un tamaño mayor o igual que el número de elementos a almacenar**
- **y que, además, la función *hash* sea muy homogénea**

Dispersión abierta: implementación de las operaciones

- **`asignaValor`: asigna una llave con un valor**
  - **Calcula el código de dispersión de la llave**
  - **Busca si ya hay un valor para esa llave en la lista correspondiente**
  - **Si le hay, sustituye el viejo valor por el nuevo**
  - **Si no, añade el nuevo valor a la lista, por ejemplo al principio (ya que es más eficiente en una lista enlazada simple)**
- **`elimina`: elimina el valor asociado a una llave**
  - **Calcula el código de dispersión de la llave**
  - **Busca si hay un valor para esa llave en la lista correspondiente**
  - **Si le hay, le elimina de la lista**
  - **Si no le hay, informa de que no existe**

- **`obtenValor`: obtiene el valor asociado a una llave**
  - **Calcula el código de dispersión de la llave**
  - **Busca si hay un valor para esa llave en la lista correspondiente**
  - **Si le hay, le retorna**
  - **Si no, indica que no existe**
- **`hazNulo`: elimina todas las asociaciones**
  - **Vacía (hace nulas) todas las listas de la tabla**

Resolución de colisiones: dispersión cerrada

La tabla contiene directamente los valores asociados a las llaves

Si se detecta una colisión, se intenta calcular un nuevo código de dispersión, hasta que se encuentra una posición vacía

$$hash_i(x) = (hash(x) + next(i)) \bmod max$$

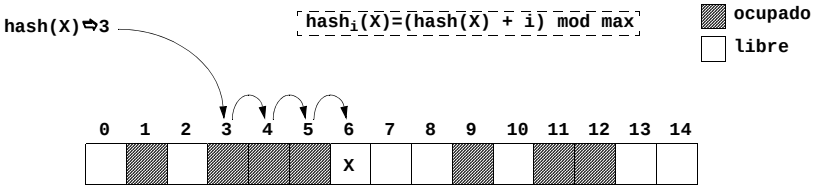
La tabla debe ser bastante mayor que el número de relaciones

- **recomendado al menos el 50% de las celdas vacías**

La función `next(i)` es la estrategia de exploración:

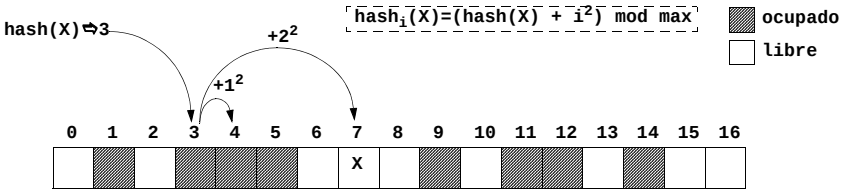
- **lineal: `next(i)=i`**
- **cuadrática: `next(i)=i`^{2}**
- **doble dispersión: `next(i)=i*hash`$_{alt}$`(x)`**

Exploración lineal



- **Ocupación de la tabla: λ=`celdasOcupadas/totalNumCeldas`**
- **Número *medio* de intentos para encontrar celda libre=`1/(1-`λ`)`**
  - **si están ocupadas la mitad de las celdas (λ=`0.5`) en promedio se necesitan 2 intentos**
  - **Si la ocupación es mayor, el número de intentos crece mucho**
- **Si la función *hash* no es homogénea la eficiencia empeora**
- **Tienden a formarse bloques de celdas ocupadas que degradan la prestaciones**

Exploración cuadrática



- **Evita en gran medida el agrupamiento de celdas ocupadas que ocurría con la exploración lineal**
- **Es preciso que la *ocupación siempre* sea *inferior a la mitad***
- **El *tamaño* de la tabla debe ser un *número primo***

Teorema: Dada una tabla de dispersión que utiliza exploración cuadrática, si su tamaño es un número primo siempre podremos insertar un nuevo elemento en la tabla si su ocupación es estrictamente inferior a 0.5. Además, durante el proceso de inserción, no se visita la misma posición más de una vez

Exploración por doble dispersión

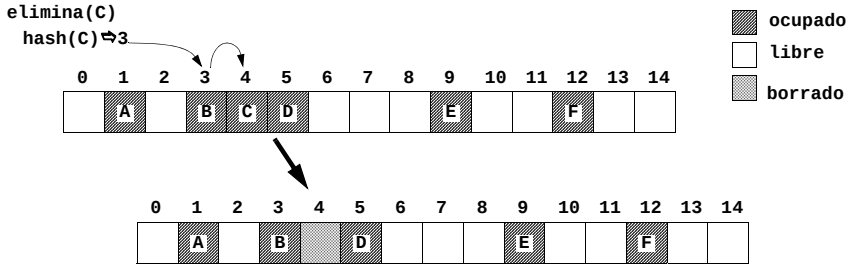
$$hash_i(x) = (hash(x) + i \times hash_{alt}(x)) \bmod max$$

Permite eliminar la agrupación secundaria que se produce en ocasiones con la exploración cuadrática

$hash_{alt}(x)$ nunca debe dar cero

- **ej.: R-(x mod R), siendo R primo**

Borrado con dispersión cerrada



Debe usarse la técnica del "*borrado perezoso*":

- **marcamos la celda como borrada (borrado≠libre)**
- **al buscar, si encontramos una celda marcada como borrada continuamos la búsqueda**
- **al añadir, puede ocuparse una celda marcada como borrada**

A veces será necesario "recrear" la tabla

Dispersión cerrada: implementación de las operaciones

- **`asignaValor`: asigna una llave con un valor**
  - **Calcula el código de dispersión de la llave**
  - **Utilizando la estrategia de exploración busca una celda libre o marcada como borrada**
  - **Asigna el valor a la celda**
- **`elimina`: elimina el valor asociado a una llave**
  - **Calcula el código de dispersión de la llave**
  - **Utilizando la estrategia de exploración busca una celda con el valor de la llave buscado**
    - **si encuentra una celda marcada como borrada, continúa la búsqueda**
  - **Si la encuentra, la marca como borrada**
  - **Si encuentra una celda libre, informa de que no existe ningún elemento con esa llave en la tabla**

- **`obtenValor`: obtiene el valor asociado a una llave**
  - **Calcula el código de dispersión de la llave**
  - **Utilizando la estrategia de exploración busca una celda con el valor de la llave buscado**
    - **si encuentra una celda marcada como borrada, continúa la búsqueda**
  - **Si la encuentra, retorna el valor**
  - **Si encuentra una celda libre, informa de que no existe ningún elemento con esa llave en la tabla**

- **`hazNulo`: elimina todas las asociaciones**
  - **Marca como libres todas las celdas de la tabla**

Eficiencia de las operaciones

Eficiencia temporal de las operaciones (tanto con dispersión abierta como con dispersión cerrada):

Operación	Array
asignaValor	O(≈1)
elimina	O(≈1)
obtenValor	O(≈1)
hazNulo	O(m) (1)

(1) Donde `m` es el tamaño del array utilizado para implementar la tabla