Estructuras de Datos

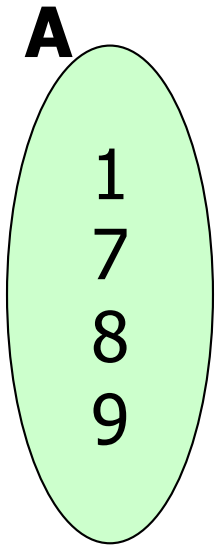
Conjuntos y Mapas

Adriana Collaguazo Jaramillo, Mg.

espol Escuela Superior Politécnica del Litoral

CONCEPTOS

- Conjunto
  - Colección de miembros
  - Cada miembro puede ser un conjunto o
  - Un elemento primitivo(átomo)
  - Todos los miembros son distintos



- Atomos
  - Enteros, caracteres o cadenas
  - En un conjunto S, los átomos son del mismo tipo
  - Ordenados linealmente por una relación “<“
    - Para cualquier a y b en S, o a<b o a==b o a>b es verdadero
    - Para todo a, b y c en S, si a <b y b< c; a<c

OPERACIONES ELEMENTALES

- Si A y B son conjuntos
- **A** $\cup$ **B** es la unión de A y B
  - Conjunto de elementos que son miembros de A, de B o de ambos
- **A** $\cap$ **B** es la intersección de A y B
  - Conjunto de elementos que pertenecen tanto a A como a B
- **A** – **B** es la diferencia
  - Conjunto de elementos de A que no pertenecen a B
- Ejemplo:
  - A = {a,b,c} y B = {b,d}
  - A $\cup$ B = {a,b,c,d}; A $\cap$ B = {b} y A – B = {a,c}

OPERACIONES DEL TDA CONJUNTO

- **Básicas**
  - Conjunto Unión(Conjunto A, Conjunto B)
  - Conjunto Intersección(Conjunto A, Conjunto B)
  - Conjunto Diferencia(Conjunto A, Conjunto B)
- **Generales**
  - bool EsMiembro(Conjunto A, Info x)
    - Determina si el elemento x se encuentra en el conjunto A
  - void Vacia(Conjunto A)
    - Convierte al conjunto A en un conjunto vacio

REPRESENTACIONES

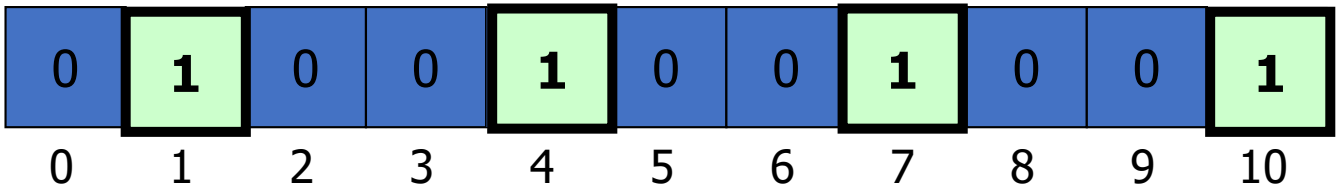
- Hay varias formas de representar un conjunto:
  - Usando vectores de bits
  - Usando listas ordenadas

Vectores de Bits

VECTORES DE BITS

- Si el conjunto que se va a manejar
  - Es subconjunto del conjunto de enteros positivos,
  - Puede usarse vectores de bits
- Un vector de bits es un arreglo booleano
  - Si el elemento de posicion i almacena
    - 1: i es miembro del conjunto
    - 0: i no es miembro del conjunto

$$A = \{4,1,10,7\}$$



DECLARACION DEL TDA

- Se necesita
  - Un vector de bits
  - Un tamaño maximo

ALGUNAS OPERACIONES

```
Conjunto Interseccion(Conjunto A, Conjunto B){
      int i;
      Conjunto C;
      for(i = 0; i < MAX;i++)
            C[i] = A[i] && B[i];
      return C;
}
Conjunto Union(Conjunto A, Conjunto B){
      int i;
      Conjunto C;
      for(i = 0; i < MAX;i++)
            C[i] = A[i] || B[i];
      return C;
}
```

Listas Ordenadas

CON LISTAS ORDENADAS

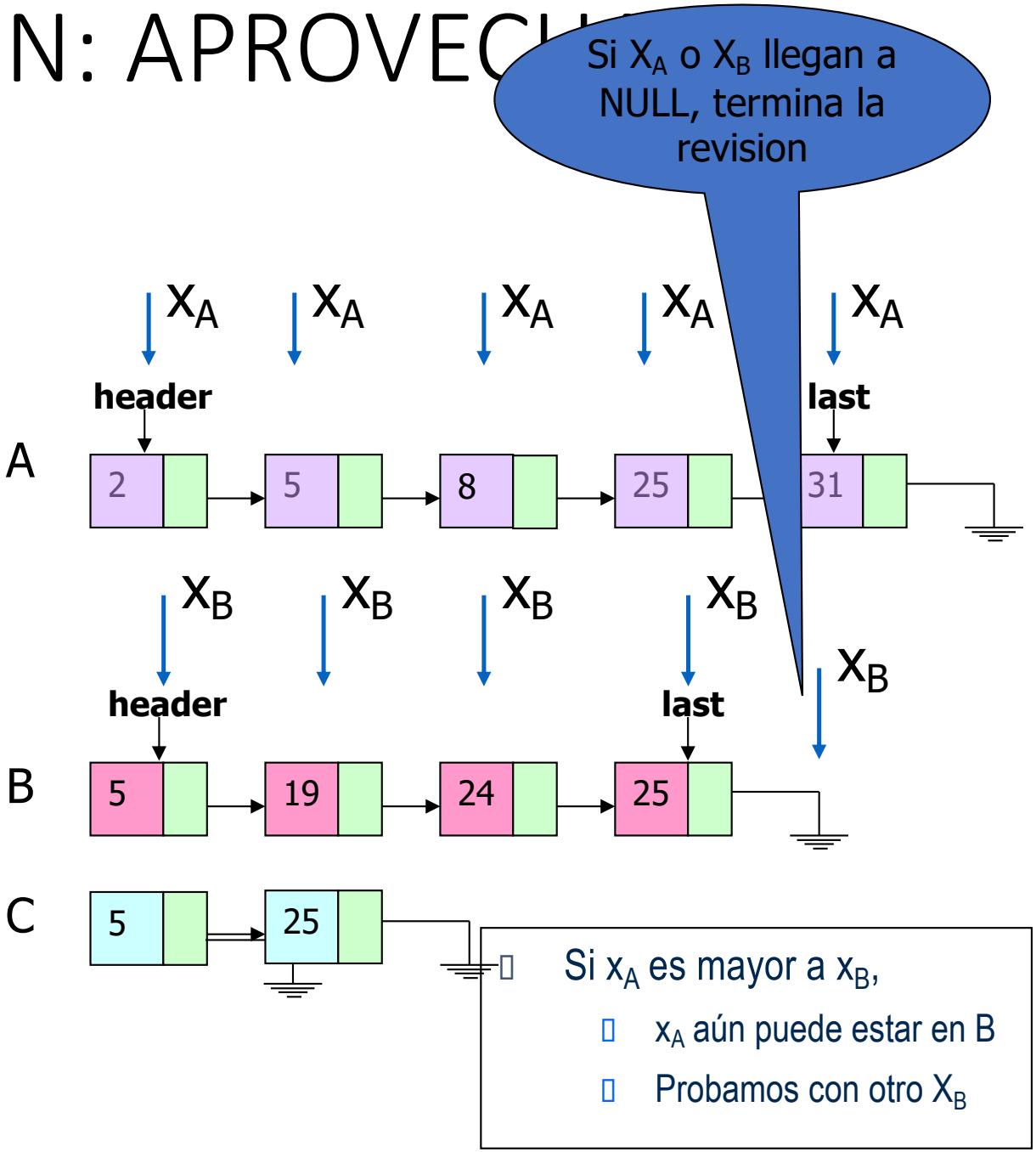
- No hay desperdicio de espacio
- No es solo para conjuntos de enteros
- Representa cualquier tipo de conjunto

- El ***ordenamiento***
  - Ayuda a la eficiencia de la implementación
  - Si queremos saber si x es miembro del conjunto
    - No tenemos que revisar toda la lista
    - Solo hasta encontrar un elemento >= x

INTERSECCION: APROVECH… ORDEN

- A ∩ B. Ejemplo:
  - A = {25,5,8,2,31} y
  - B = {24,5,25,19}
- Comenzamos un elemento de A(x_A)

- Comparamos uno a uno con los elementos de B(x_B)
- Si x_A es menor que x_B,
  - x_A ya no puede estar en B
  - Pasamos a otro x_A
- Si x_A es igual a x_B,
  - Añadir x_A al nuevo conjunto
  - Pasamos a otro x_A y otro X_B



EJERCICIO EN CLASE

- Implemente las operaciones
  - Unión y Diferencia usando listas ordenadas

- Recuerde aprovechar el orden!

Conjuntos en Java

Java Classes e Interfaces

Interface **`Set`** para colecciones genéricas de elementos que no admiten duplicados

`TreeSet`

`HashSet`

`LinkedHashSet`

https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

Resumen Conjuntos

Un conjunto es una colección que no contiene elementos duplicados

En Java, la interfaz `Set` modela la abstracción de conjuntos matemáticos

`Set<String>`

```
Set<Book> set1 = new HashSet();

Set<Book> set2 = new LinkedHashSet<>();

Set<Book> set3 = new TreeSet<>(); // Asume orden natural; si no, lanza excepción
```

Para ignorar el orden natural, un `TreeSet` debe instanciarse con un comparador:

```
Set<Book> set4 = new TreeSet<>((book1, book2) -> {
        return book1.getYear() - book2.getYear();
});
```

Diferencia entre `HashSet`, `LinkedHashSet` y `TreeSet`

Principal diferencia:

Orden en que se devuelven los elementos del conjunto al invocar el método **`iterator()`**

`HashSet`: Los elementos se retornan **sin ningún orden** en particular

`LinkedHashSet`: Los elementos se retornan de acuerdo al **orden de inserción**

`TreeSet`: Los elementos del conjunto se devuelven en **orden ascendente***

*de acuerdo al orden natural o de acuerdo al comparador utilizado

TDA Mapa o Diccionario

TDA DICCIONARIO

- Un conjunto puede estar limitado
  - Y no necesitar operaciones como Intersección y Unión
- Típicamente se necesita
  - Añadir elemento a un Conjunto: Insertar
  - Consultar si un elemento existe: EsMiembro
  - Eliminar elementos de un Conjunto: Suprimir
- Así, limitado, el conjunto es un Diccionario

DISPERSION O HASHING

h(1) = 4

h(8) = 0

h(2) = 2

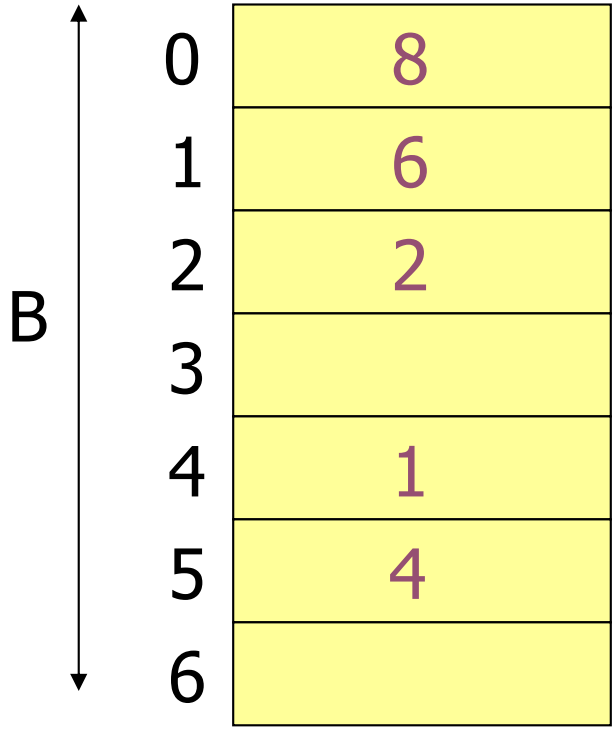
h(6) = 1

h(4) = 5

- Dado una tabla de B filas,

  A = {4,6,2,8,1}

  - Cada fila se llama: *Cubeta-Slot-Bucket*
- Queremos distribuir los elementos en las cubetas
  - Se usará una función de dispersión
  - Con la clave del elemento,
    - La función calculará a que cubeta deberá ir dicho elemento



FUNCION DE DISPERSION

- También: Función de conversión o Hash: h(x)
- Transforma una clave en índice
  - La clave puede ser un string o un entero
- ¿Dos claves pueden tener el mismo índice?
  - Si, este caso se conoce como **colisión**
  - Para estos casos hay dos estrategias de colisión
- La función hash debe
  - Distribuir índices uniformemente (pocas colisiones)
  - Generar un índice entre 0 y B-1
  - Tener un algoritmo sencillo

POSIBLES H(x)

- **Aritmética modular**
  - El índice se obtiene: clave%B
  - Conviene que ncubetas sea primo
- **Mitad del cuadrado**
  - Elevar al cuadrado la clave
  - Tomar los dígitos de una determinada posición
  - El número de dígitos depende del rango del índice
  - Ejemplo:
    - Si B: 100 y clave = 256,
    - Tomaremos 2 dígitos de posición: 1 y 2 (desde derecha)
    - índice: 256^2 = 65536 = Tomamos 63

POSIBLES H(x)

- **Truncamiento**
  - Tomar dígitos directamente de la clave
  - En posiciones fijas
  - Ejemplo:
    - B = 1000, tomar 3 dígitos siempre en posiciones 1, 2, 5
    - clave = 72588495, índice = 598
- **Plegamiento**
  - Consiste en dividir la clave en trozos
  - Luego aplicarles alguna operación
  - Ejemplo: “DAVID”
    - Sumar los códigos ASCII
    - Tomar el residuo para B, y ese es el índice

TRATAMIENTO DE COLISIONES

- *Hashing Abierto*
  - Cada cubeta almacena una colección de elementos
- *Hashing Cerrado*
  - Cada elemento va en una cubeta
  - Si ha colision se aplica *redispersion*
    - Aplicar otra funcion hash de reserva
  - Y se intenta de nuevo

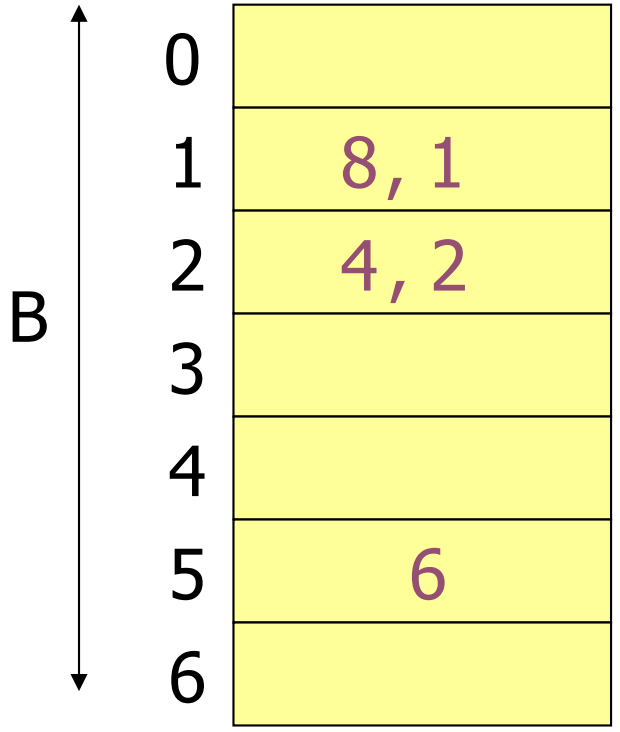
HASHING ABIERTO

h(1) = 1
h(8) = 1
h(2) = 2
h(6) = 5
h(4) = 2

- Supongamos que

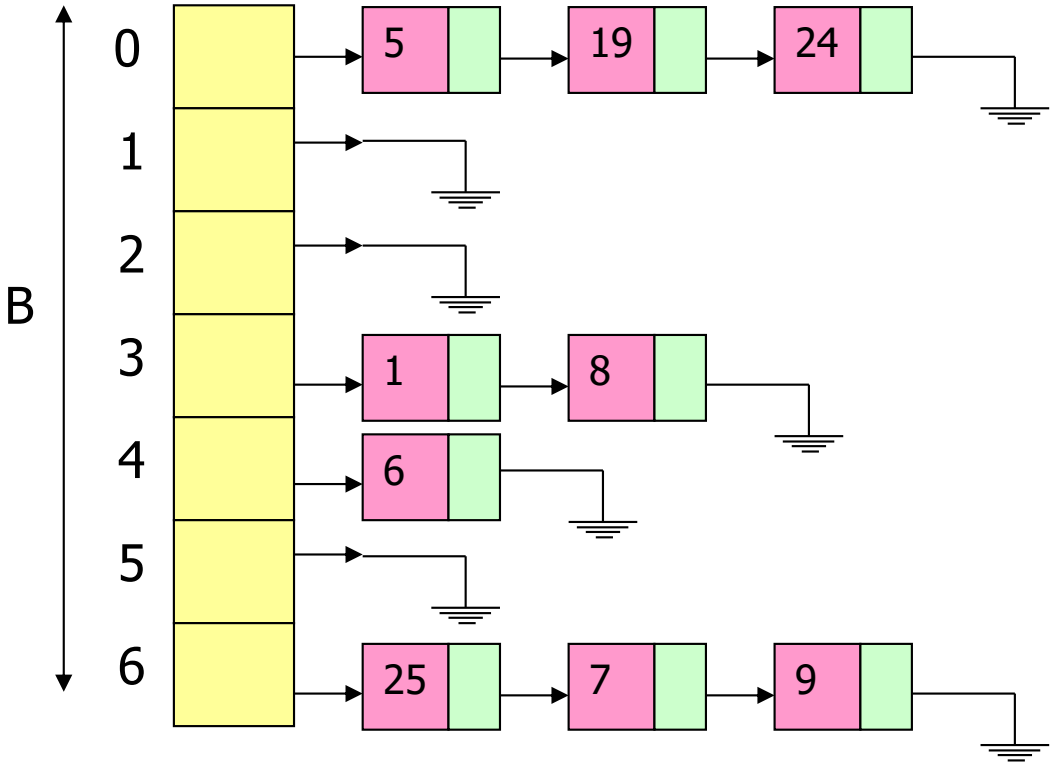
  A = {4,6,2,8,1}

  - Hay colision entre d1,d2,..dn, es decir
  - H(d1) = H(d2) = H(dn)
- Todas estos elementos
  - Se asignan a la misma cubeta
  - Son parte de la lista de elementos de dicha cubeta
- La cubeta no tiene limites de almacenamiento



DECLARACION

- Cada Cubeta tiene una lista
    - De elementos asociados a la misma
- La tabla es un arreglo de cubetas



HASHING CERRADO

- Una cubeta para un elemento
- Si se da una colisión
  - Se aplica otra función hash, $h_i(x)$
  - Donde i, es el número de colisión para el elemento x
  - Se intenta de nuevo(redispersión)
- Si hay elementos en todas las cubetas
  - La tabla está llena
  - No se puede insertar el elemento

h(1) = 1

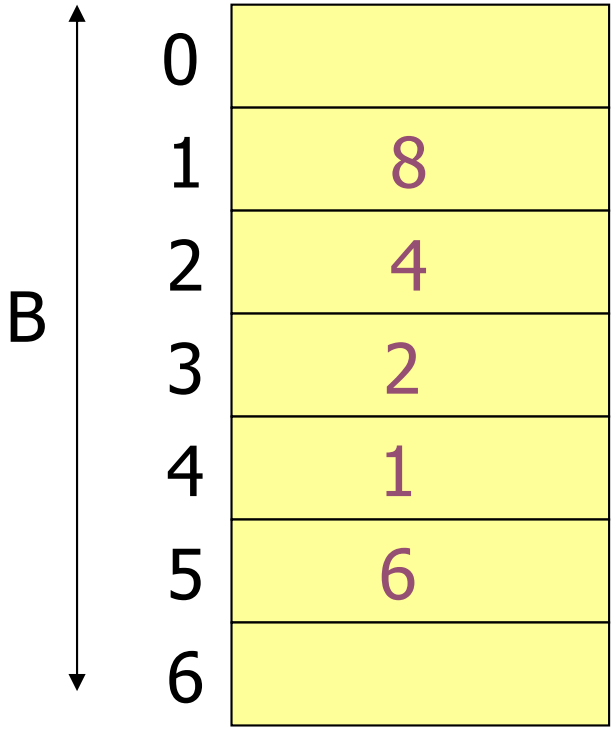
h(8) = 1

h(2) = 2

h(6) = 5

h(4) = 2

A = {4,6,2,8,1}



$h_3(1) = 4$

$h_2(1) = 3$

$h_1(1) = 2$

$h_1(2) = 3$

POSIBLES $H_i(X)$

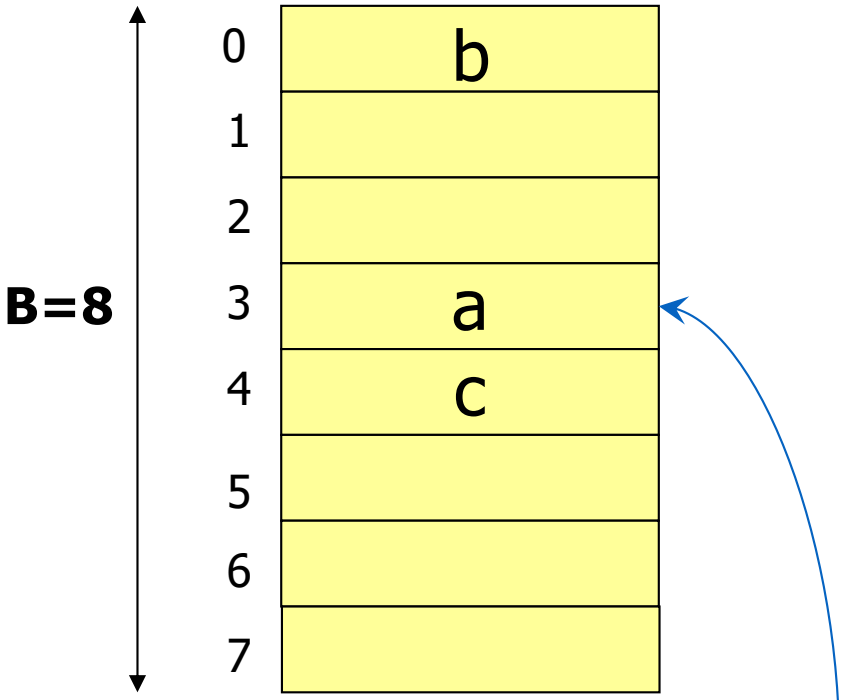
- Rehash Lineal
  - $h_i(x) = (h(x) + i) \% B$
  - Agrupa cubetas llenas en grandes bloques consecutivos
- Rehash Cuadrático
  - $hi(x) = (h(x) + i^2) \% B$

EJEMPLO

Redispersar: h1(d) = h(d) + 1

- Suponga que B = 8 y que
- Elementos a,b,c,d,e tienen
  - h(a) = 3, h(b) = 0, h(c) = 4, h(d) = 3, h(e) = 3
- Insertemos con redispersión lineal

- Busquemos los elementos
  - f, b, d
- Eliminar a, y
- Busquemos otra vez d



Insertar: a, cubeta = 3

Insertar: b, cubeta = 0

Insertar: c, cubeta = 4

Insertar: d, cubeta = 3

Resumen Mapas

Un mapa es un objeto que **relaciona claves y valores**

Requiere una función de ***hash*** o función de dispersión `h(x)`

Cuatro funciones de dispersión conocidas:

- Aritmética modular
- Mitad del cuadrado
- Truncamiento
- Plegamiento

Una función de dispersión `h(x)` puede producir colisiones

En **hashing abierto**, las colisiones no son un problema: cada cubeta almacena una colección de valores

En **hashing cerrado**, para resolver una colisión hay que re-dispersar (aplicar una función de rehash)

Cuatro funciones de re-dispersión conocidas:

- Rehash lineal
- Rehash cuadrático
- Rehash aleatorio
- Rehash con doble función

Mapas en Java

Resumen Mapas en Java

En Java, no se admite claves duplicadas

Una clave debe estar asociada a un solo valor

```
Map<Integer, String>
```

```
Map<Integer, String> map1 = new HashMap<>();
Map<Integer, String> map2 = new LinkedHashMap<>();
Map<Integer, String> map3 = new TreeMap<>(); // Claves con orden natural

map.put(3310, “Valor 1”); // inserción de clave y valor
map.get(3310) // retorna valor asociado a la clave 3310
```

Para ignorar el orden natural, un `TreeMap` debe instanciarse con un comparador:

```
Map<Integer, String> map1 = new TreeMap<>((i1, i2) -> {
        return ...
});
```

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

Diferencia entre HashMap, LinkedHashMap y TreeMap

Principal diferencia

Orden en que se devuelven las claves del mapa al invocar el método **`iterator()`** del conjunto de retornado por el método **`keySet()`**

```
Set<Integer> keySet = map.keySet();
Iterator<Integer> iterator = keySet.iterator();
while (iterator.hasNext()) {
        Integer key = iterator.next();
        String value = map.get(key);
        System.out.println("Clave: " + key + " Valor: " + value);
}
```

`HashMap`: Las claves se retornan **sin ningún orden** en particular.

`LinkedHashMap`: Las claves se retornan de acuerdo al **orden de inserción.**

`TreeMap`: Las claves se devuelven en **orden ascendente de las claves o usando el comparador de las claves.**

`HashMap`

No garantía sobre orden de iteración. Distinto al agregar nuevos elementos.

`LinkedHashMap`

Iterará en el orden en que se colocaron las entradas en el mapa

`TreeMap`

Iterará de acuerdo con el *orden natural* de las claves

HashMap

LinkedHashMap

TreeMap

HashMap

HashSet

LinkedHashMap

LinkedHashSet

TreeMap

Treeset