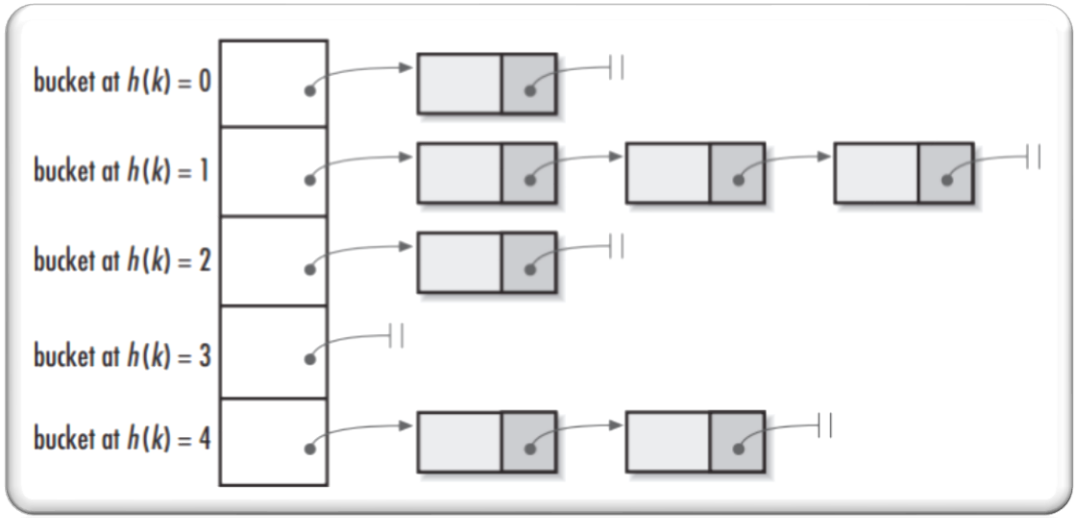
[Estructuras de Datos]



TABLAS HASH.

Copyright

TABLAS HASH

Se recomienda que todo lo documentado en las siguientes secciones sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

Introducción :: ¿Qué es una tabla hash?

- Una **tabla hash**, es una **ED** que asocia llaves o claves con valores.
- La **operación principal** que **soporta** de manera eficiente es la **búsqueda**.
  - Permite el **acceso** a los elementos a partir de una clave.
  - Funciona transformando la clave con una **función hash** en un **número** que identifica la **posición** (casilla o cubeta) donde la tabla hash localiza el valor deseado.
- Las **tablas hash** proveen tiempo constante de **búsqueda** promedio ***O(1)***, sin importar el número de elementos en la tabla.
  - En casos particularmente **malos**, el tiempo de búsqueda puede llegar a **O(n)**.
  - Esto **depende principalmente** de que la **función hash** sea "buena", esto es, que **distribuya** o no **uniformemente** las claves.

Tabla hash :: Implementaciones

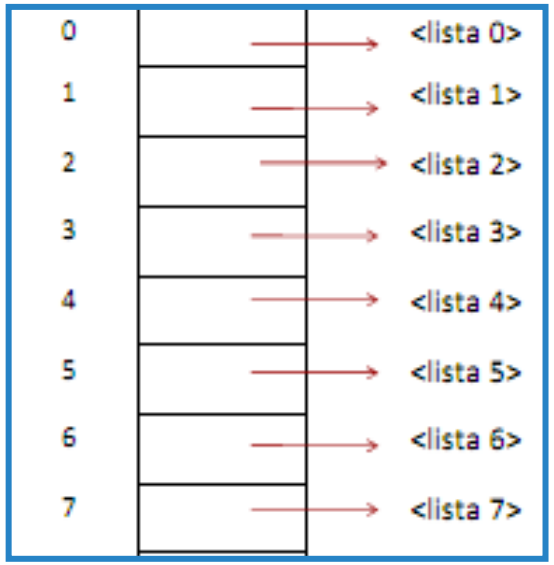
- En la materia veremos dos implementaciones posibles:
  - Tabla de hash **abierto** (*separate chaining*).
  - Tabla de hash **cerrado** (*open addressing*).
- Cada una de estas implementaciones trabaja de **forma diferente** la **resolución** de **colisiones**.
  - Recordar que una **colisión** sucede cuando la **función hash** retorna un mismo valor **hash** para dos claves distintas ($h(k1) = h(k2)$).
  - En esta situación, se debe proveer alguna **estrategia** que permita **resolver dónde** almacenar los valores cuyas claves tienen el mismo valor **hash**.
  - Se debe tener en cuenta que esta **estrategia** determina también **cómo** luego se deben, por ejemplo, **recuperar** y **eliminar** esos pares clave-valor.

HASH ABIERTO

El contenido de las siguientes transparencias (parcialmente) son de autoría de la Dra. M. L. Ganuza (mlg@cs.uns.edu.ar).

Tabla hash abierta :: Implementación

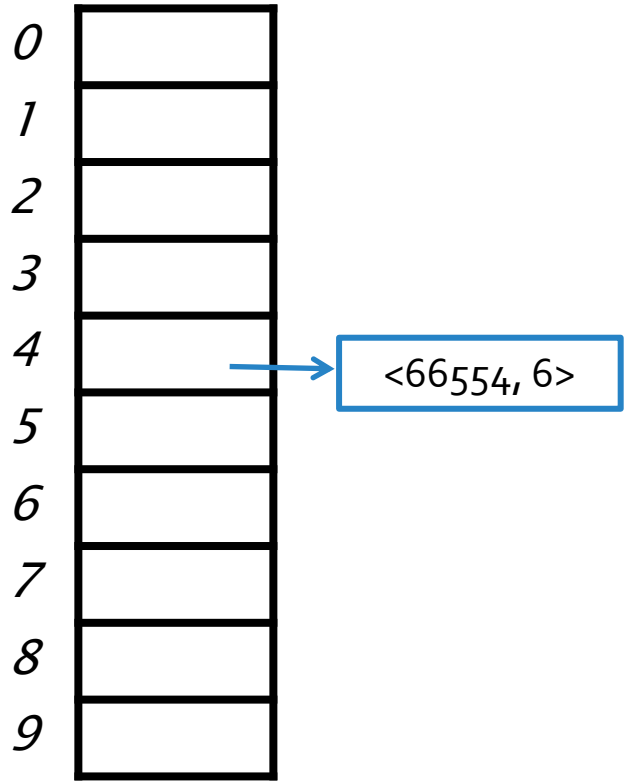
- La ED de la tabla hash consiste en un **arreglo** de **buckets**.
  - El **arreglo** será de **N** **buckets**.
  - Cada **bucket** es una **colección** de pares *clave-valor*.
  - La **función hash** toma una clave *k1* y retorna la **posición h(k1)** de un **bucket**, y el par *k1-v1* es almacenado en la **colección** de dicho **bucket** (**arreglo[h(k1)]**).



Esta tabla hash está implementada considerando:
- *Arreglo de 8 buckets.*
- *Cada bucket es una PositionList<Entrada<K,V>>*
- *La función hash está definida como h(k1) = hashCode(k1) mod 8*

Tabla hash abierta :: Ejemplo

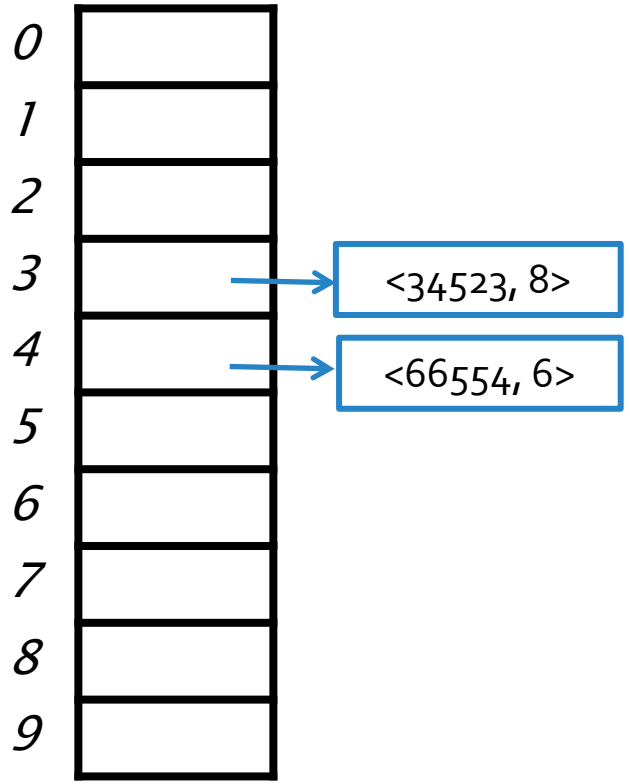
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	
< 34230, 3 >	
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

Tabla hash abierta :: Ejemplo

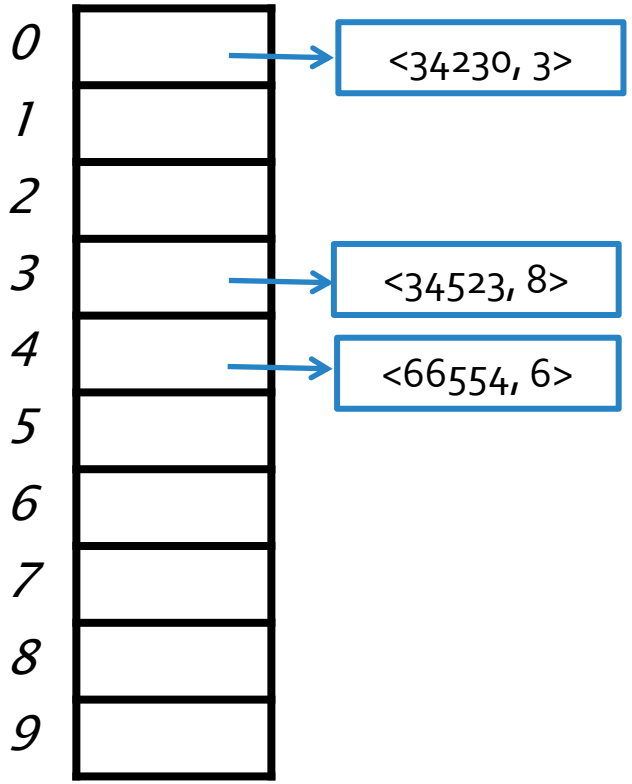
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

Tabla hash abierta :: Ejemplo

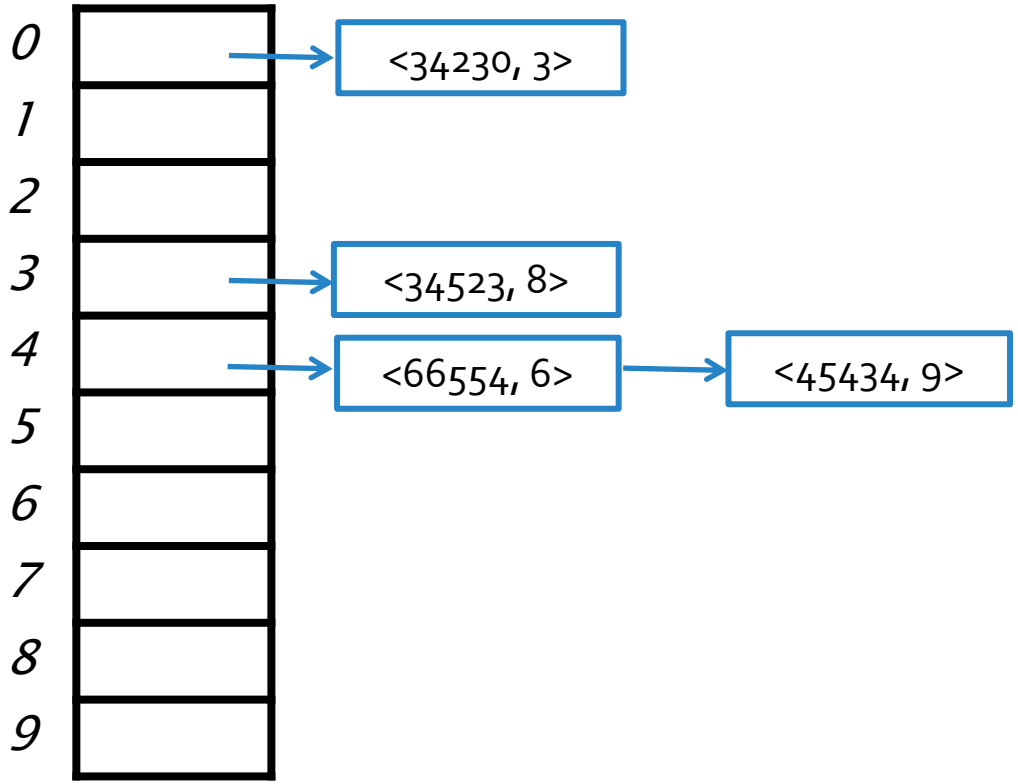
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

Tabla hash abierta :: Ejemplo

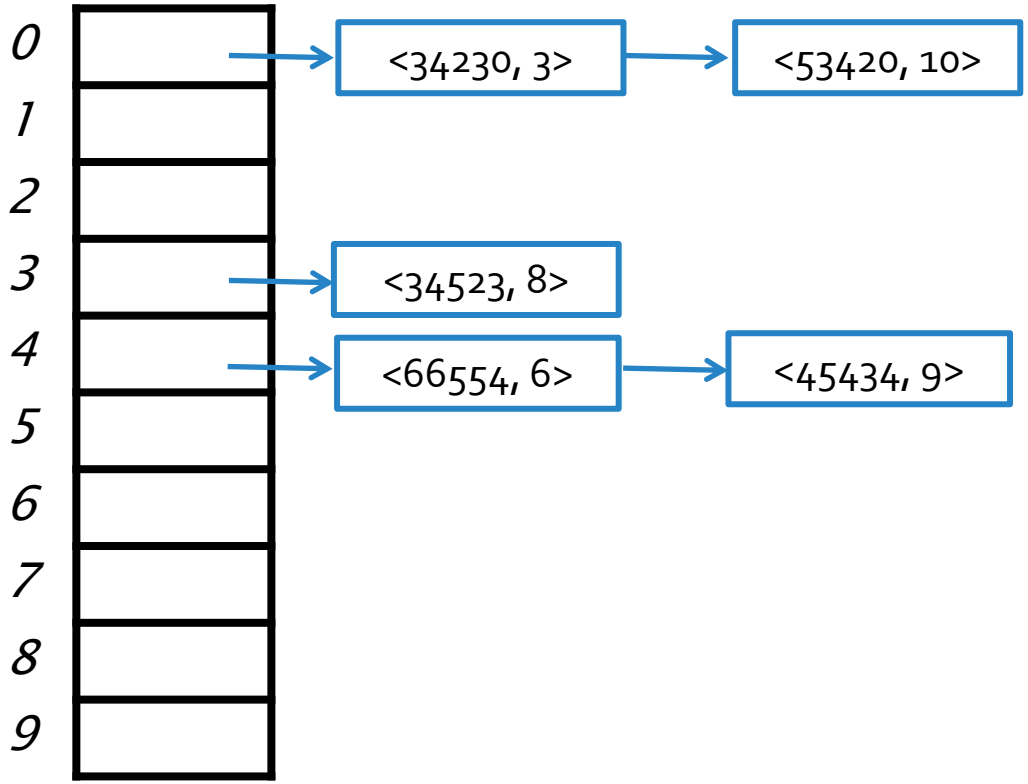
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	
< 61148, 7 >	

Tabla hash abierta :: Ejemplo

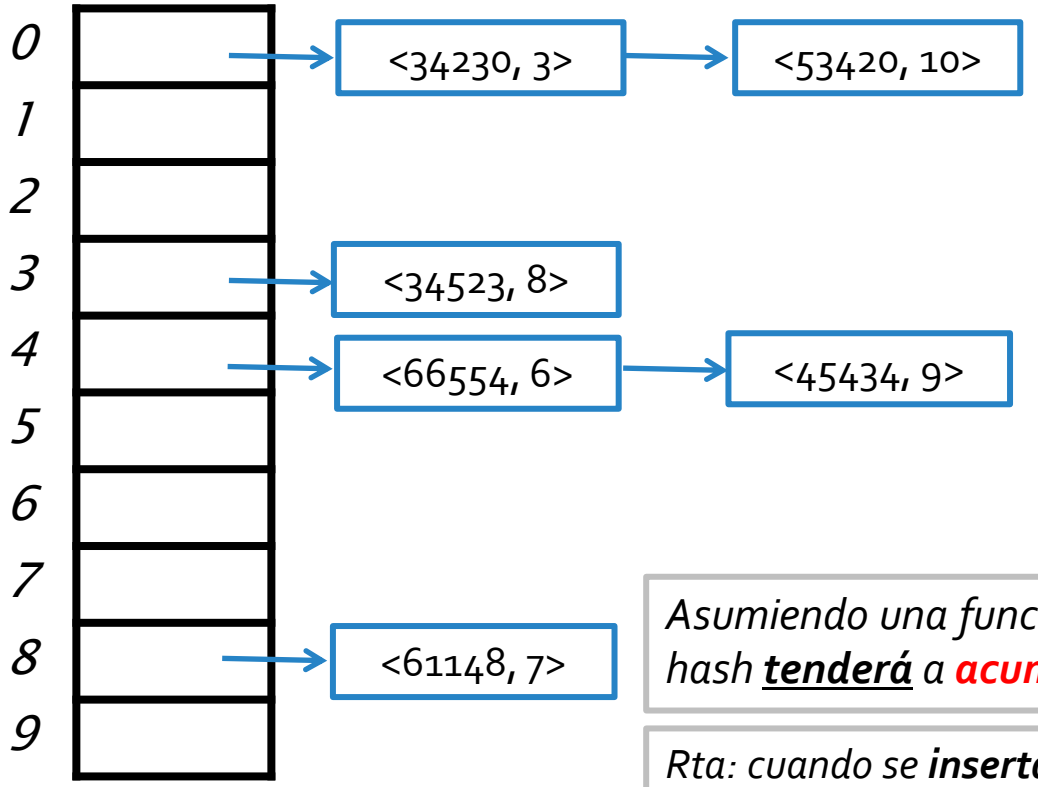
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	H(53420) = 0
< 61148, 7 >	

Tabla hash abierta :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



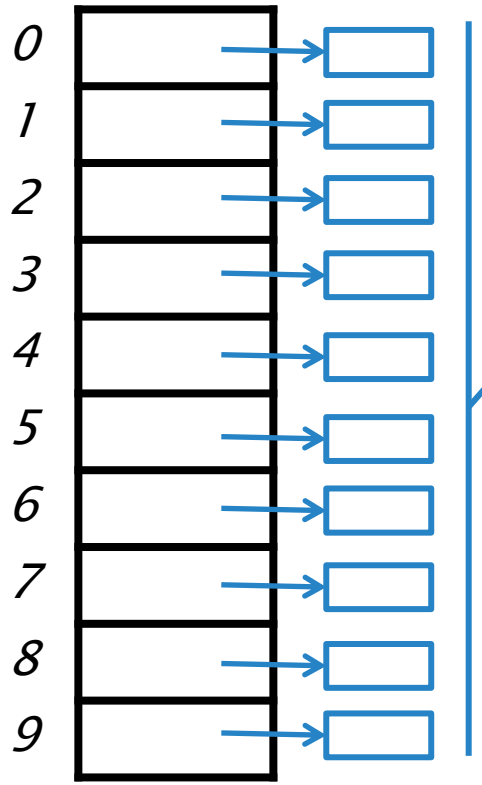
Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	H(53420) = 0
< 61148, 7 >	H(61148) = 8

*Asumiendo una función hash que distribuye uniformemente las claves, ¿cuándo la tabla hash **tenderá** a **acumular** más de un par en los diferentes buckets?*

*Rta: cuando se **inserta** una cantidad de pares clave-valor **mayor** que la **cantidad** de **buckets**. De esta forma, es **indispensable** realizar un **control del factor de carga** de la **tabla hash**.*

Tabla hash abierta :: Ejemplo

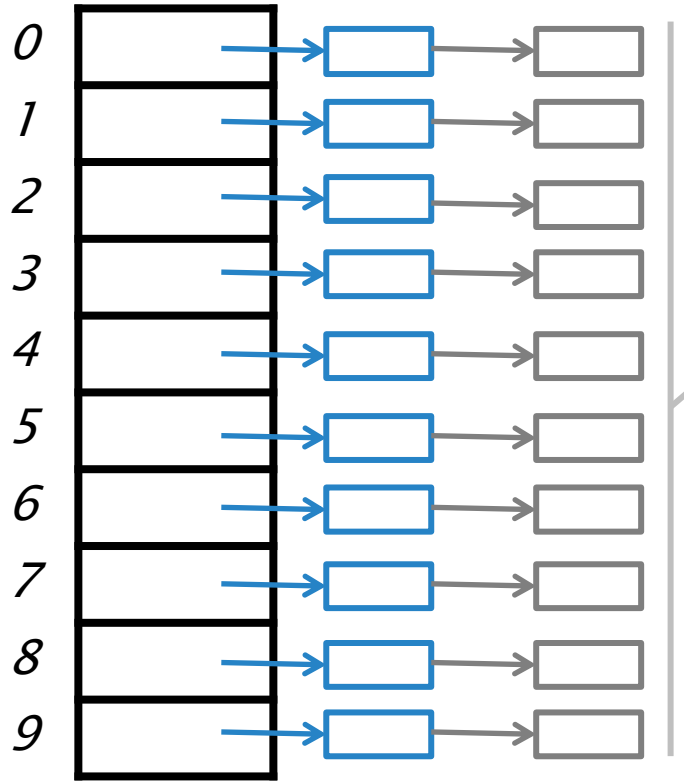
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



*Considerando la **inserción** de **10 pares** con claves k1,k2,…,k10, una **función hash** h(Clave) **distribuiría uniformemente**, en el **mejor** de los casos, los pares de la siguiente forma.*

Tabla hash abierta :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets**.



*Considerando la **inserción** de **10 pares** con claves k1,k2,...,k10, una **función hash** h(Clave) distribuiría uniformemente, en el mejor de los casos, los pares de la siguiente forma.*

*Una vez que se insertaron 10 pares, **independientemente** de que la función hash distribuya **uniformemente o no**, cada uno de los buckets comenzará a tener más de un elemento. Por ejemplo, nuevamente en el **mejor** de los casos, los siguientes 10 pares con claves k11,k12,...,k20, se insertarían de la siguiente manera.*

Asegurando un factor de carga λ, por ejemplo, $\lambda<0,9$, se garantiza que ante **una función hash "buena"**, cada **bucket** tendrá a lo sumo 1 elemento.

$$\lambda = \text{Cant. Pares} / \text{Cant. Buckets}$$

Tabla hash abierta :: Implementación

- Independientemente del **TDA** que se implemente, la **ED subyacente** de una **tabla hash abierta** consideraría:

```
public class MiTDAConHashAbierto<K, V> implements MiInterfazTDA<K,V> {
  protected PositionList<Entrada<K,V>> [] arreglo;
  protected int tamañoInicial = 13;

  public MiTDAConHashAbierto(){
    arreglo = (PositionList<Entrada<K,V>> []) new PositionList [tamañoInicial];
    for(int i=0; i<tamañoInicial; i++){
      arreglo[i] = new DoubleLinkedList<Entrada<K,V>>();
    }
  }
  protected int hash(K clave){
    return Math.abs(clave.hashCode() % arreglo.length);
  }

  //Operaciones propias del TDA
}
```

El arreglo de buckets de un tamaño fijo inicial.

Un constructor que se encargue de instanciar cada uno de los buckets, con la colección utilizada.

La definición de la función hash que se encarga de tomar una clave y determinar la posición (bucket) que la clave ubicará en la tabla hash.

Tabla hash abierta :: Implementación

- El **control** del **factor de carga** se realiza al momento de **agregar** elementos a la **tabla hash.**
- En caso de que se **supere** el **factor de carga**, se debe **redimensionar** el **arreglo de buckets**.

```
Algoritmo insertar(Clave, Valor):
  Entrada ← nueva entrada(clave,valor)
  Si ((cantidad_pares / arreglo.length) >= fc)
    redimensionar()
  Fin_si
  Bucket ← hash(Clave)
  Lista ← Arreglo[Bucket]
  Lista.insertar(entrada)
Fin
```

- ***proximo_primo(n)*** *retorna un primo m>n.*
- *En la* ***redimensión****, y como una buena política,* ***no se crean*** *entradas nuevas, sino se* ***reutilizan*** *las ya creadas. A la vez, se vacían los buckets del arreglo anterior que luego se* ***eliminarán****.*
- *Hash(Entrada.obtenerClave()) se* ***computa*** *respecto al nuevo arreglo que tiene al menos el* ***doble*** *de la dimensión que el arreglo anterior.*

```
Algoritmo redimensionar():
  NuevaDimension ← próximo_primo(cantidad_pares * 2)
  ArregloAnterior ← Arreglo
  Arreglo ← inicializar arreglo de NuevaDimension buckets
  Desde i←0 hasta ArregloAnterior.length - 1
    ListaAnterior ← ArregloAnterior[i]
    Mientras no_es_vacia(ListaAnterior)
      Entrada ← eliminar un elemento de ListaAnterior
      Bucket ← hash(Entrada.obtenerClave())
      ListaNueva ← Arreglo[Bucket]
      ListaNueva.insertar(Entrada)
    Fin_Mientras
  Fin_Desde
Fin
```

HASH CERRADO

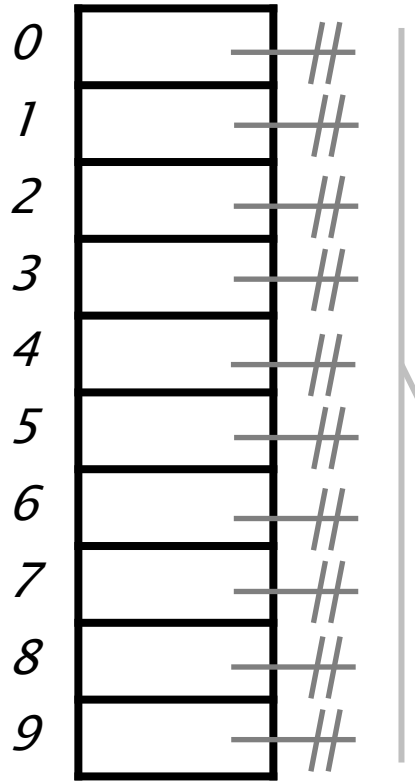
El contenido de las siguientes transparencias (parcialmente) son de autoría de la Dra. M. L. Ganuza (mlg@cs.uns.edu.ar).

Tabla hash cerrada :: Implementación

- La ED de la tabla hash consiste en un **arreglo** de **buckets**.
  - El **arreglo** será de **N** **buckets**.
  - Cada **bucket** **almacena** un único par *clave-valor*.
  - La **función** **hash** toma una clave *k1* y retorna la **posición** **h(k1)** de un **bucket**, y el par *k1-v1* es almacenado en dicha posición, si es que está libre.
  - Cuando se producen **colisiones**, se debe buscar un **próximo bucket libre** donde insertar un par, existiendo diferentes **políticas**:
  - Política de resolución **lineal**, **cuadrática**, o por **doble** **hash** de colisiones → en la materia solo utilizaremos la **política de resolución lineal**.

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

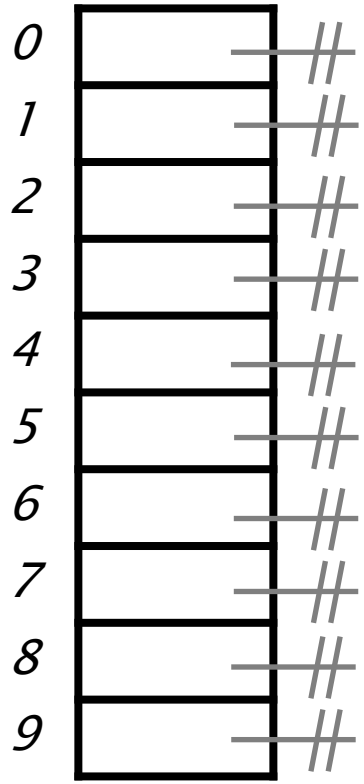


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	
< 34523, 8 >	
< 34230, 3 >	
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

- ***Inicialmente*** *todos los* ***buckets*** *del arreglo deben referenciar a un objeto común indicando que* ***nunca fueron utilizados****.*
- *En este caso, se eligió que los* ***buckets*** *referencien a null para modelar esta situación.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

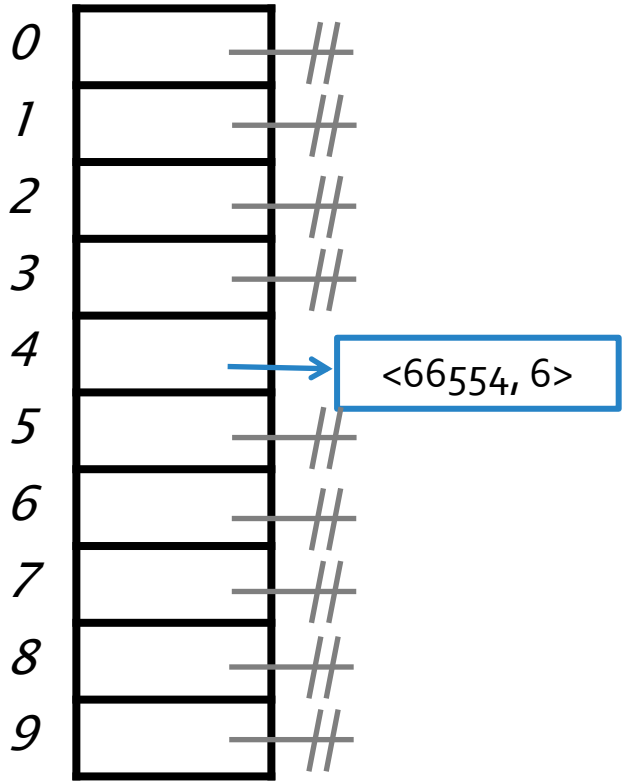


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	
< 34230, 3 >	
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

- *Como el bucket de la posición 4 referencia a null, esto es, nunca fue utilizado, se puede almacenar el par en dicho bucket.*

Tabla hash cerrada :: Ejemplo

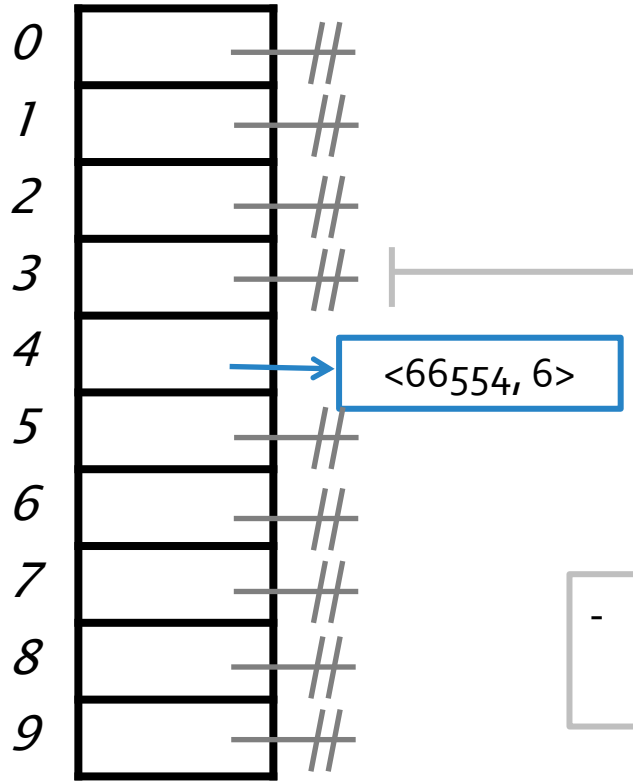
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	
< 34230, 3 >	
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

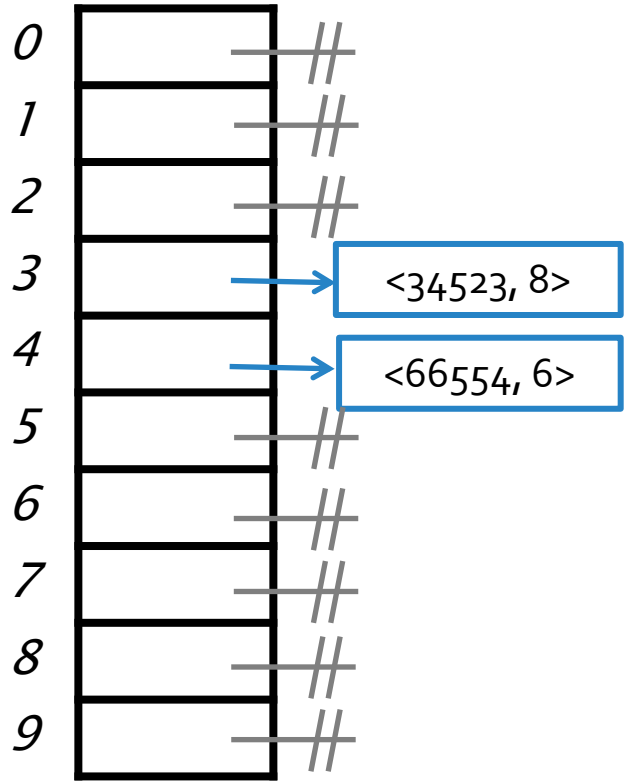


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

- *Como el bucket de la posición 3 referencia a null, esto es, nunca fue utilizado, se puede almacenar el par en dicho bucket.*

Tabla hash cerrada :: Ejemplo

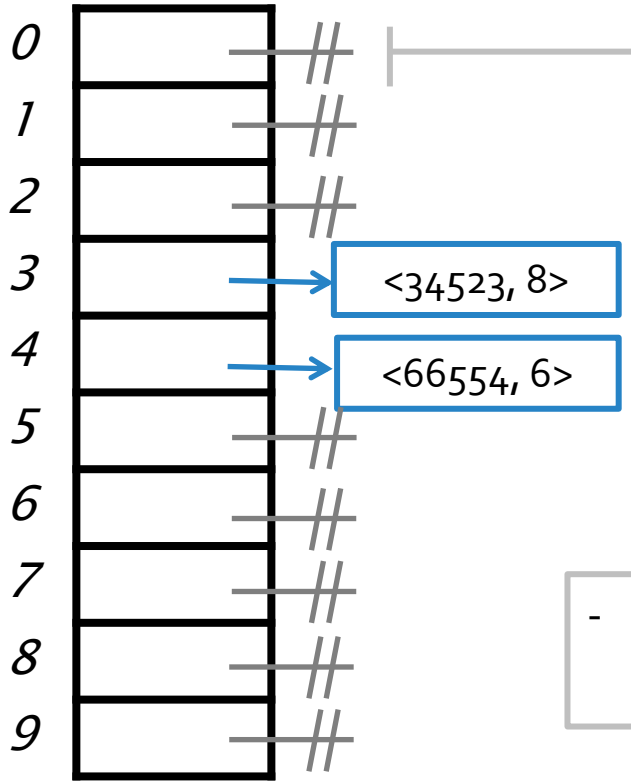
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

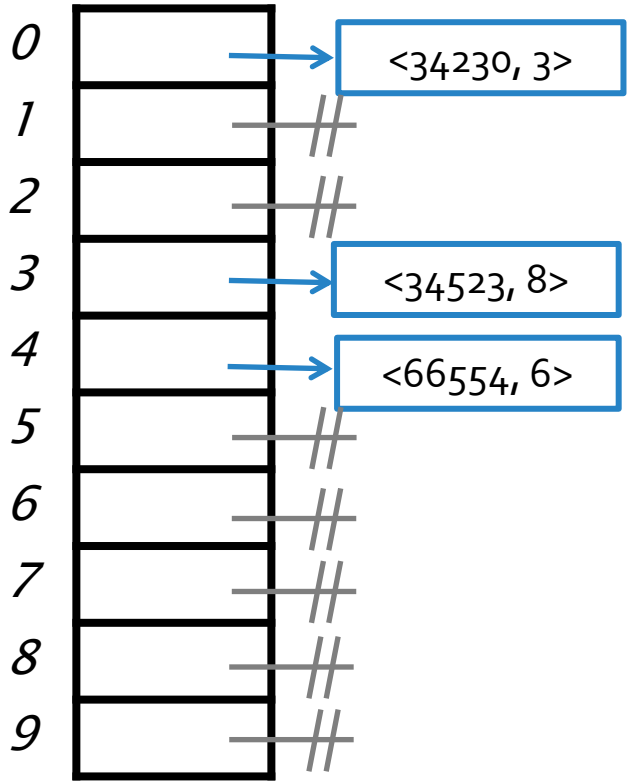


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

- *Como el bucket de la posición o referencia a null, esto es, nunca fue utilizado, se puede almacenar el par en dicho bucket.*

Tabla hash cerrada :: Ejemplo

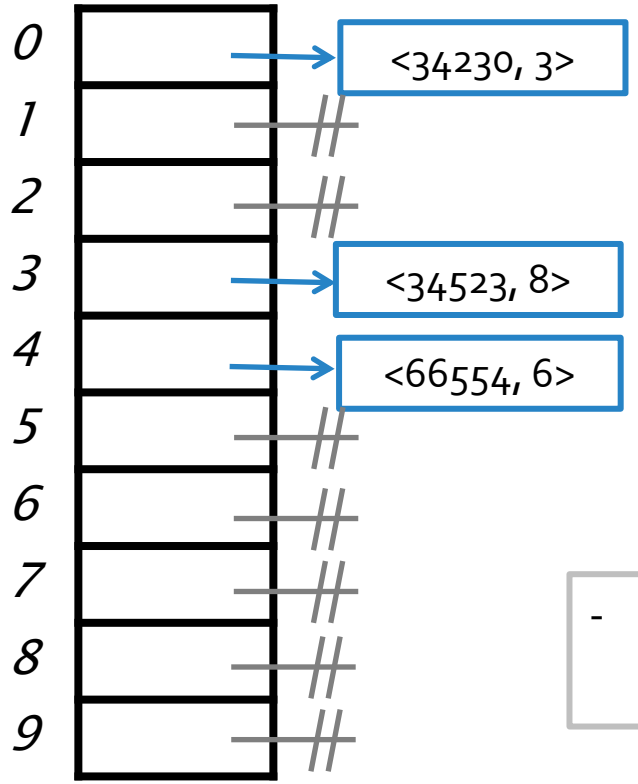
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	
< 53420,10 >	
< 61148, 7 >	

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

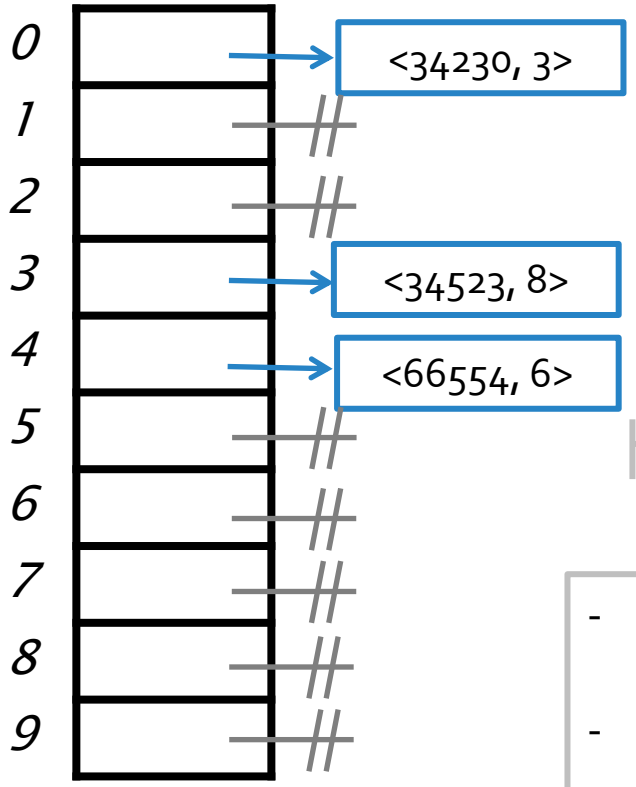


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	
< 61148, 7 >	

- *Como el bucket de la posición 4 está ocupado, se produce una colisión y hay que resolverla.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

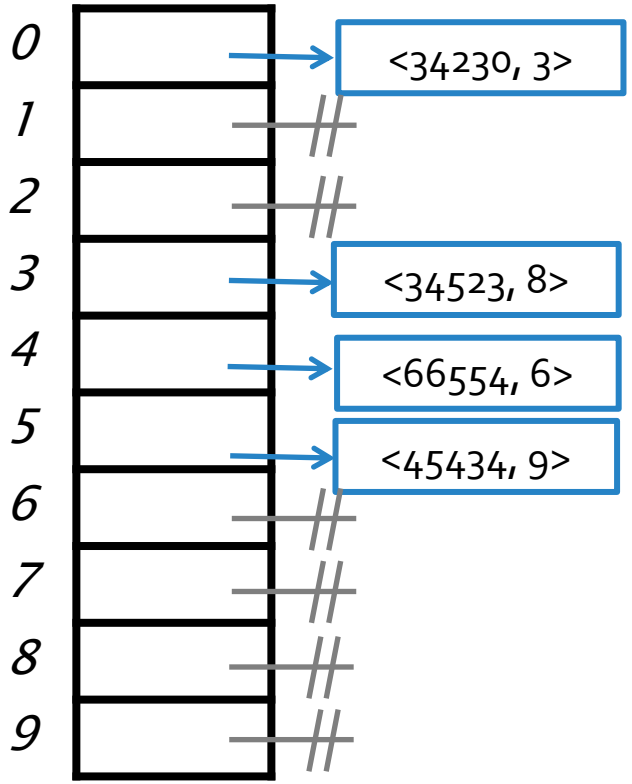


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	
< 61148, 7 >	

- *Como el bucket de la posición 4 está ocupado, se produce una colisión y hay que resolverla.*
- *En efecto, según la política, se debe buscar la próxima ubicación libre a partir de la posición 4 y de forma lineal, esto es, la posición 5.*

Tabla hash cerrada :: Ejemplo

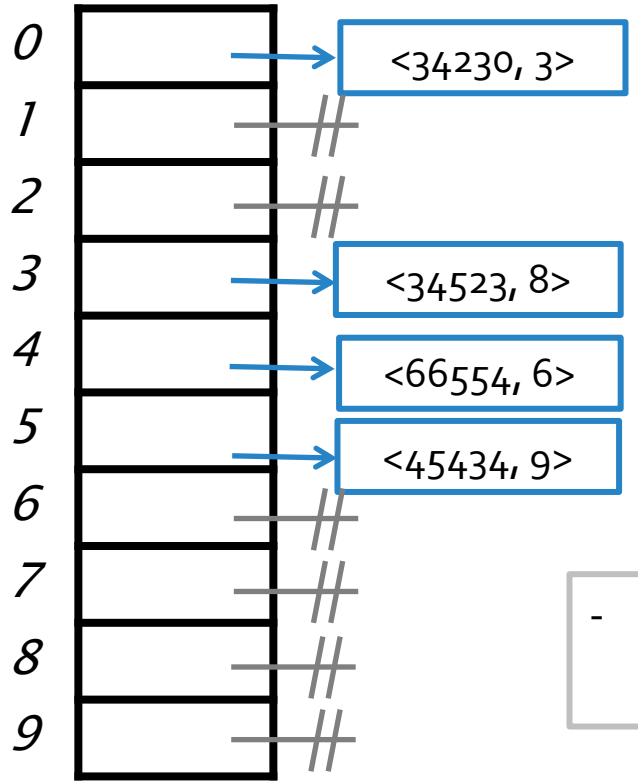
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	
< 61148, 7 >	

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

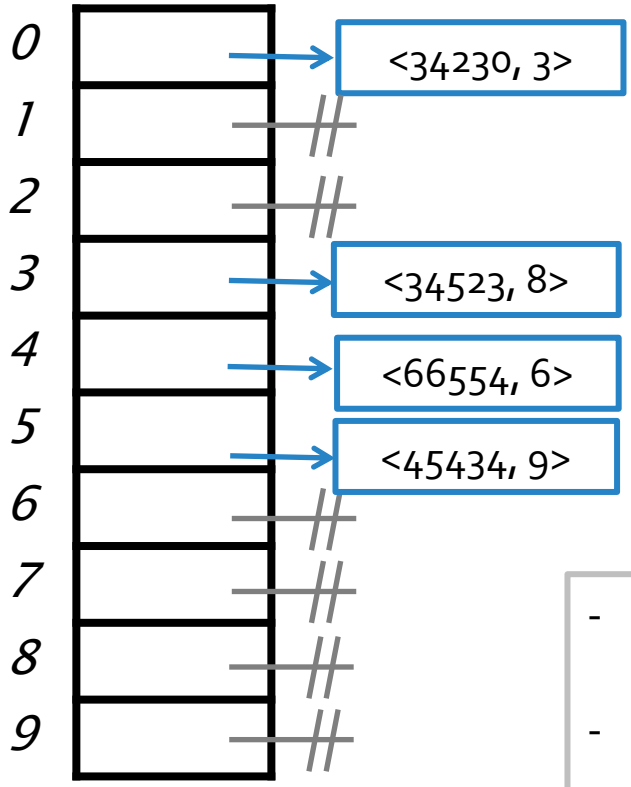


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	H(53420) = 0
< 61148, 7 >	

- *Como el bucket de la posición 0 está ocupado, se produce una colisión y hay que resolverla.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

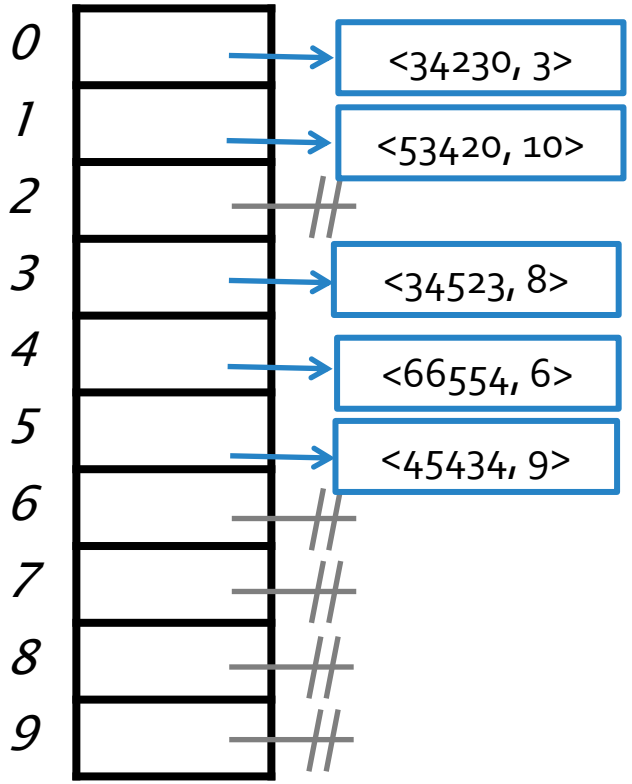


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	H(53420) = 0
< 61148, 7 >	

- *Como el bucket de la posición 0 está ocupado, se produce una colisión y hay que resolverla.*
- *En efecto, según la política, se debe buscar la próxima ubicación libre a partir de la posición 0 y de forma lineal, esto es, la posición 1.*

Tabla hash cerrada :: Ejemplo

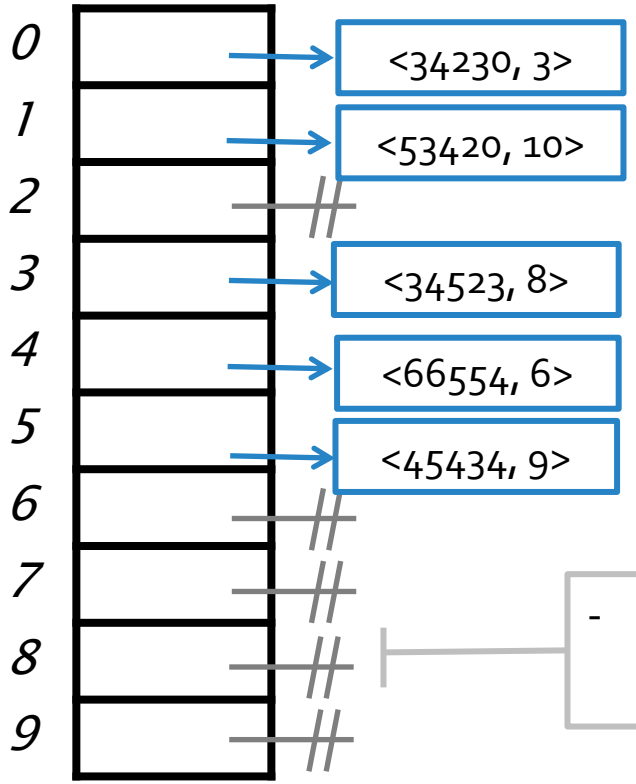
- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	H(53420) = 0
< 61148, 7 >	

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

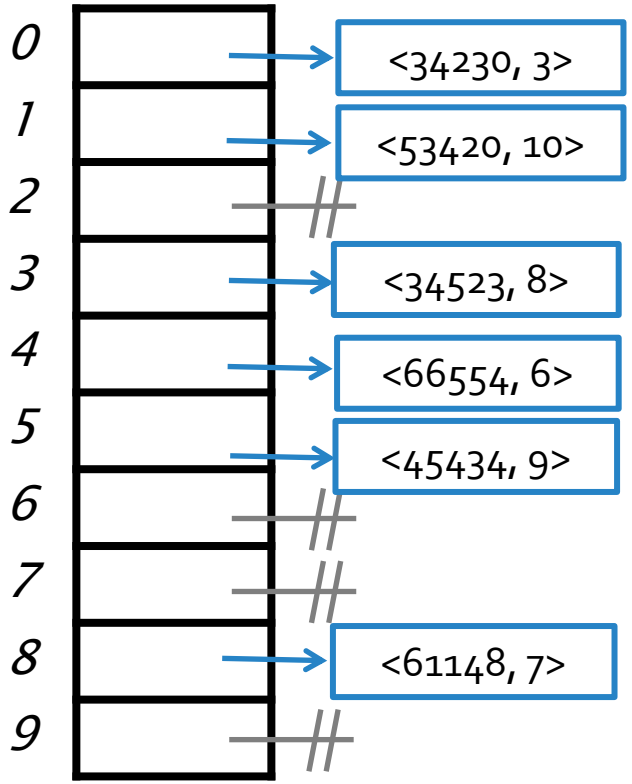


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	H(53420) = 0
< 61148, 7 >	H(61148) = 8

- *Como el bucket de la posición 8 referencia a null, esto es, nunca fue utilizado, se puede almacenar el par en dicho bucket.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

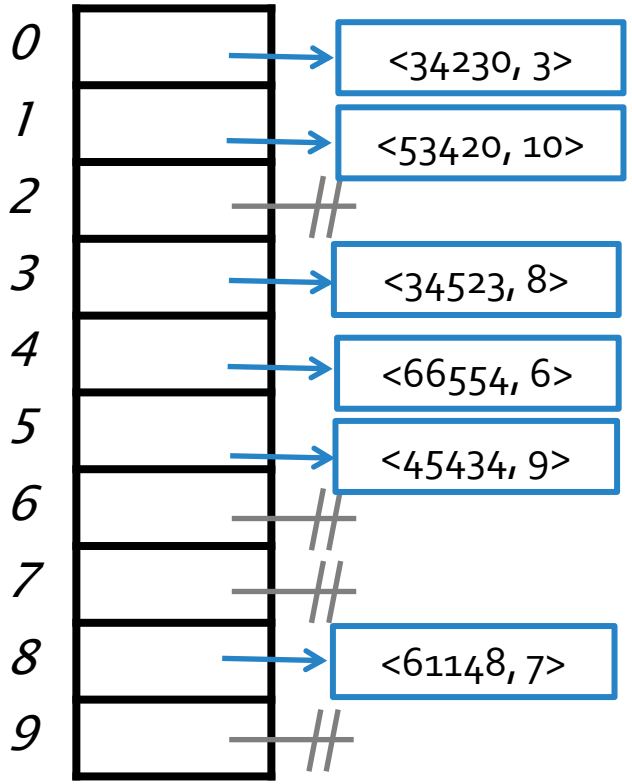


Par <Clave, Valor>	H(Clave) = Clave mod 10
< 66554,6 >	H(66554) = 4
< 34523, 8 >	H(34523) = 3
< 34230, 3 >	H(34230) = 0
< 45434, 9 >	H(45434) = 4
< 53420,10 >	H(53420) = 0
< 61148, 7 >	H(61148) = 8

*Notar (más allá del ejemplo) que la **búsqueda lineal** de un bucket libre también es circular respecto al arreglo. Suponiendo que la **posición 9 esté ocupada**, cuando un nuevo par debe ocupar dicha ubicación, la política de resolución de colisiones indicará que el **próximo bucket libre** está en la posición 2.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.

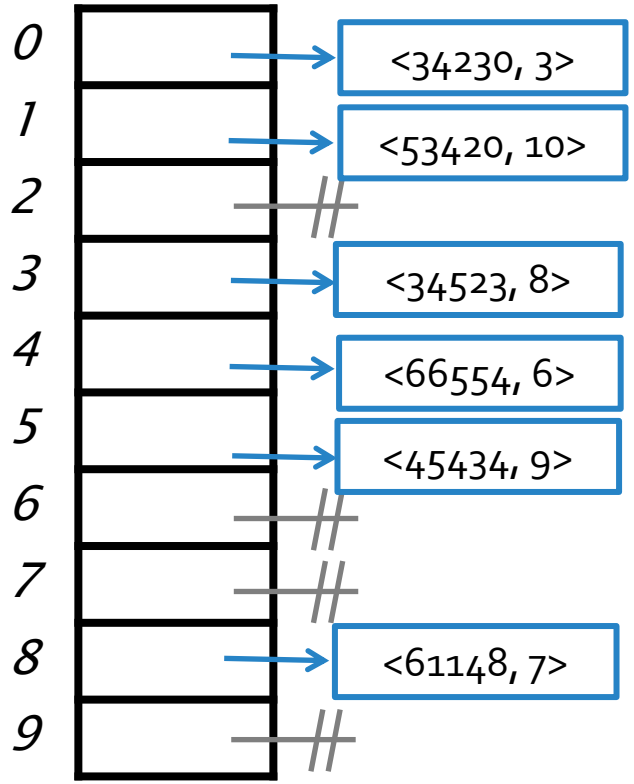


- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	
< 66554,6 >	
< 45434, 9 >	

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



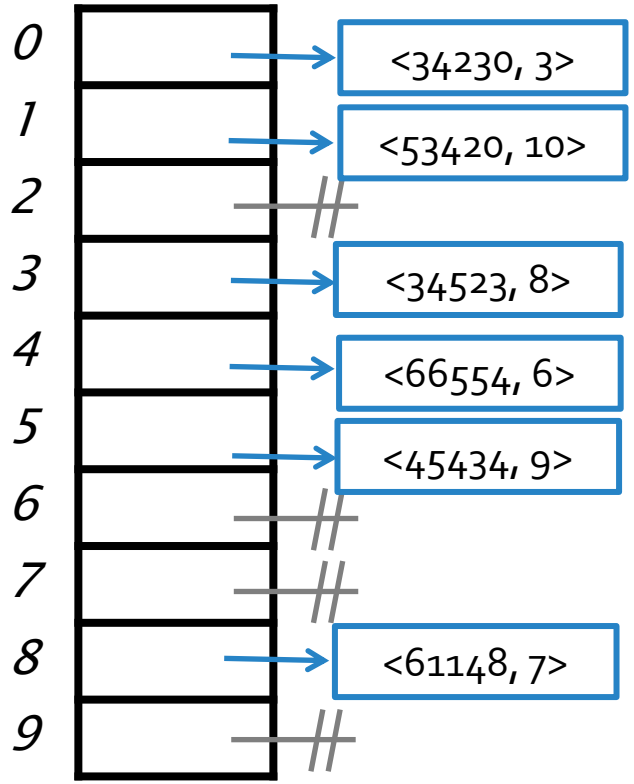
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	
< 45434, 9 >	

- *Como el bucket de la posición o no tiene la clave buscada, y considerando que almacena una entrada con otra clave, se debe buscar linealmente.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



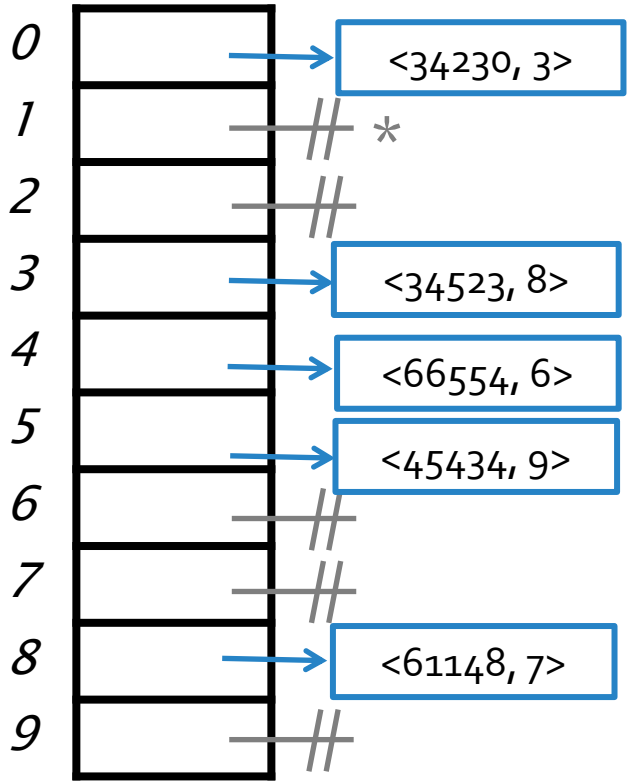
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	
< 45434, 9 >	

- *Como el bucket de la posición o no tiene la clave buscada, y considerando que almacena una entrada con otra clave, se debe buscar linealmente.*
- *En efecto, el próximo bucket, el de la posición 1, almacena el par con la clave buscada.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



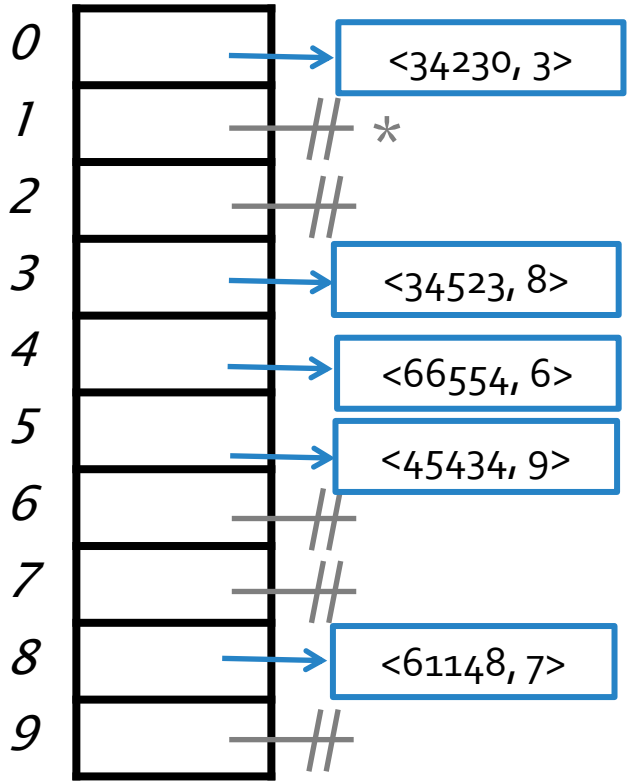
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	
< 45434, 9 >	

- *Como el bucket de la posición o no tiene la clave buscada, y considerando que almacena una entrada con otra clave, se debe buscar linealmente.*
- *En efecto, el próximo bucket, el de la posición 1, almacena el par con la clave buscada.*
- *Luego, se elimina dicha entrada. Referenciará a null el bucket de la posición 1 (*).*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



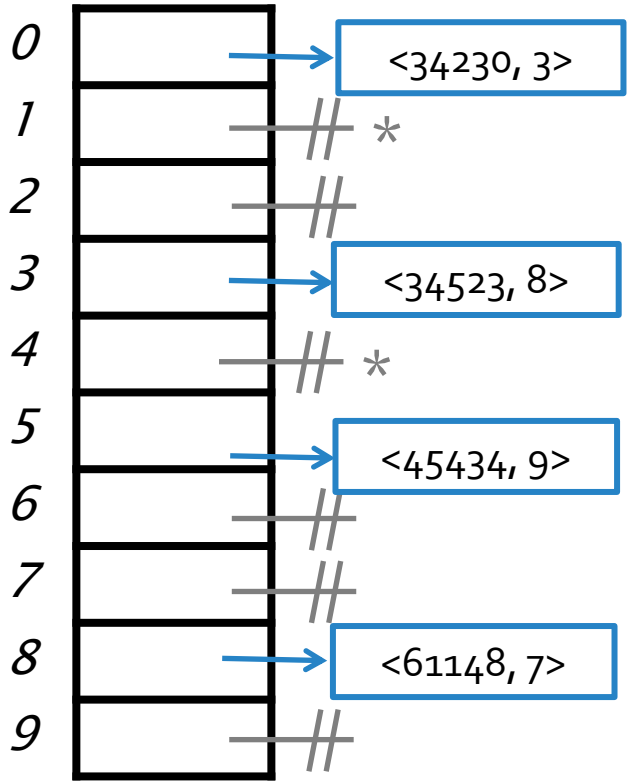
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	

- *Como el bucket de la posición 4 tiene la clave buscada, se elimina dicha entrada.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



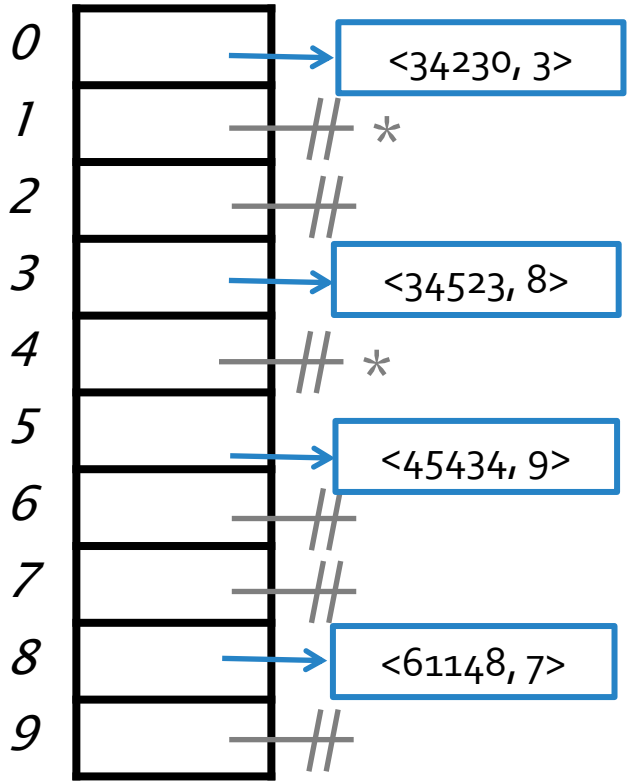
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	

- *Como el bucket de la posición 4 tiene la clave buscada, se elimina dicha entrada.*
- *Luego, el bucket de la posición 4 referenciará a null (*).*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



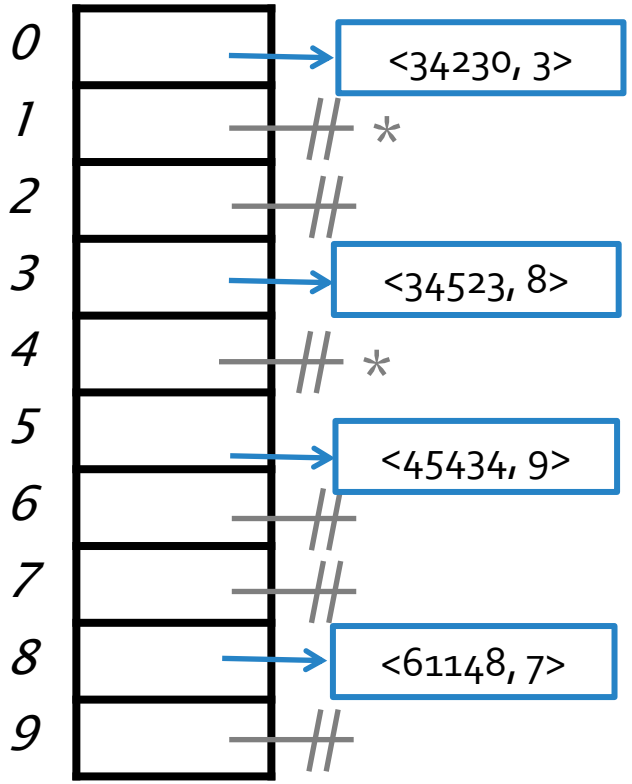
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	H(45434) = 4

- *Como el bucket de la posición 4 referencia a null, esto es, según la convención utilizada nunca fue utilizado: ¿se debe continuar buscando?*
- *El problema surge a partir de eliminar una entrada y referenciar a null (buckets *).*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



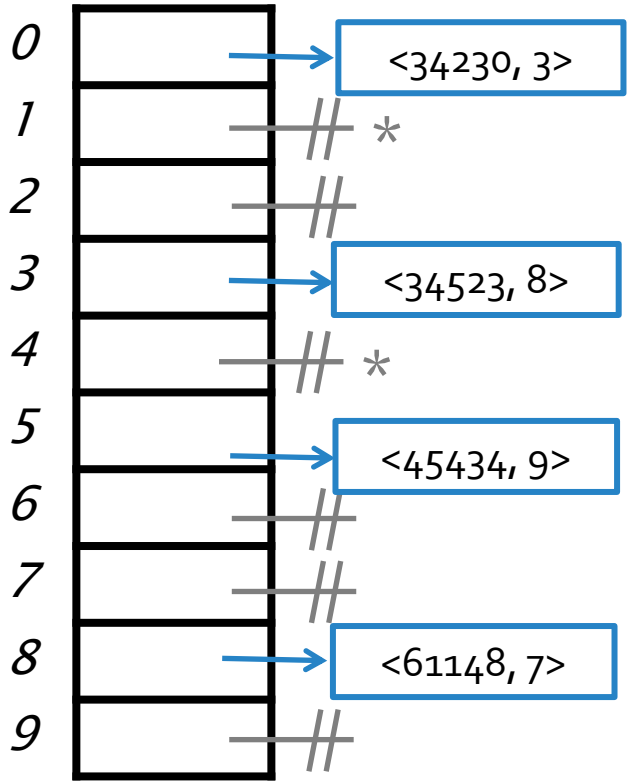
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	H(45434) = 4

- *Para diferenciar un bucket que nunca se utilizó de uno que sí y del que se removió una entrada, se utiliza una referencia a un objeto denominado disponible.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



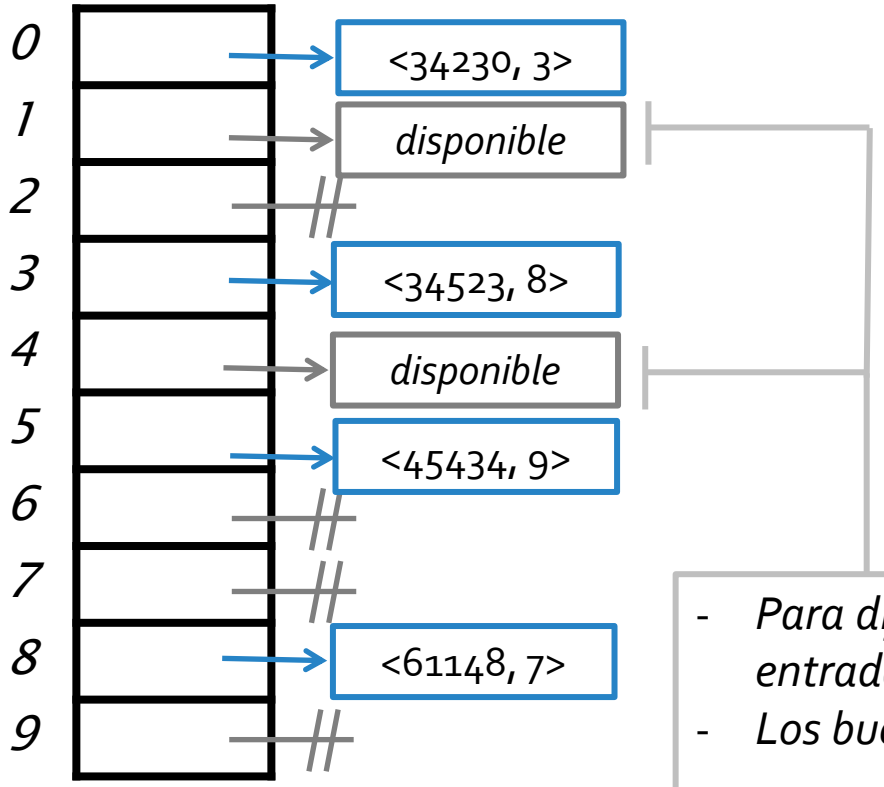
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	H(45434) = 4

- *Para diferenciar un bucket que nunca se utilizó de uno que sí y del que se removió una entrada, se utiliza una referencia a un objeto denominado disponible.*
- *Los buckets *, al momento de eliminarse, se indican como disponibles.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



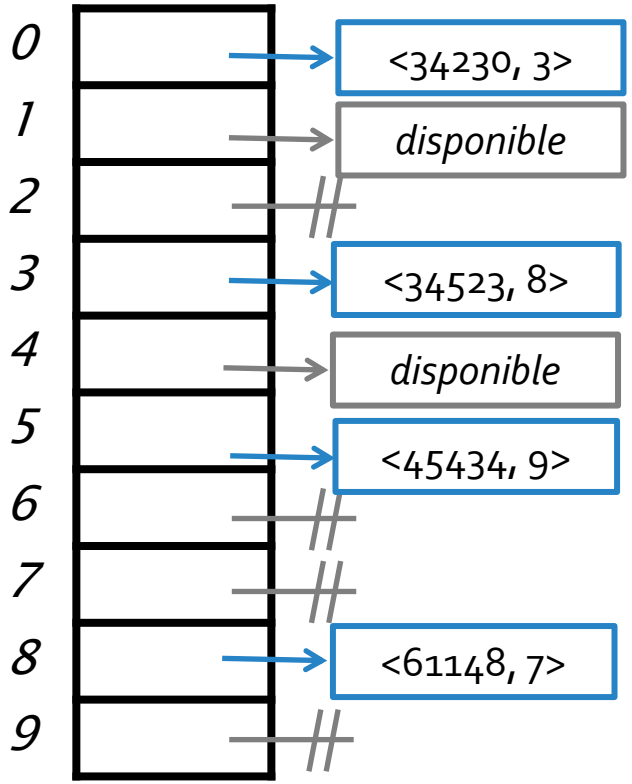
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	H(45434) = 4

- *Para diferenciar un bucket que nunca se utilizó de uno que sí y del que se removió una entrada, se utiliza una referencia a un objeto denominado disponible.*
- *Los buckets *, al momento de eliminarse, se indican como disponibles.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



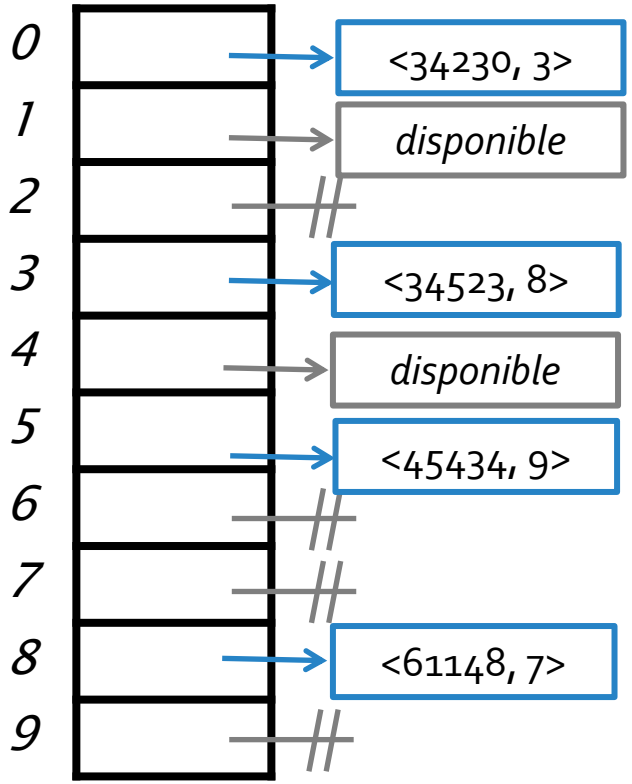
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	H(45434) = 4

- *Como el bucket de la posición 4 referencia a disponible, esto indica que este bucket fue utilizado en algún momento, por lo que es claro que la clave 45434 puede haberse almacenado en otro bucket (según resolución lineal de colisiones) y por lo tanto, se debe continuar buscando.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



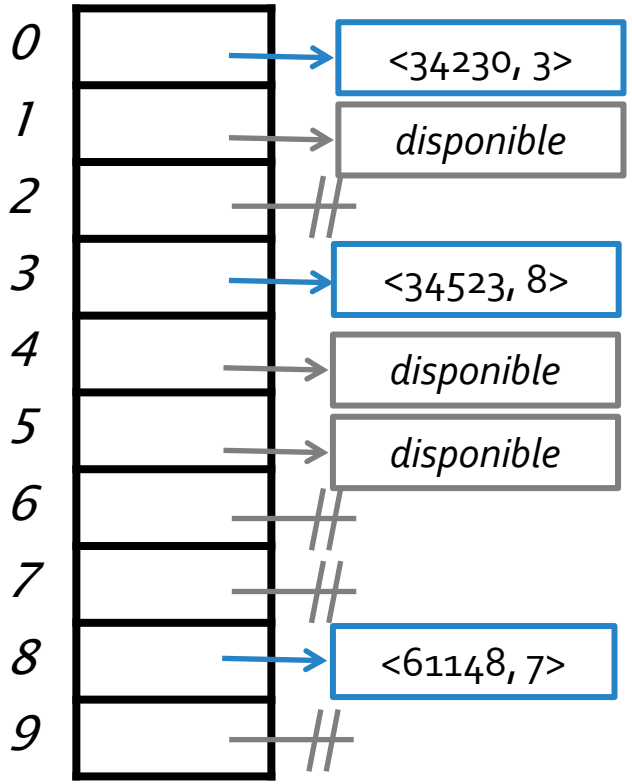
- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	H(45434) = 4

- *Se buscará en todos los próximos buckets (linealmente) hasta bien se halle la clave buscada, un bucket referenciando a null, o bien se haya considerado toda la tabla.*
- *En este caso la búsqueda finaliza porque se halla la clave buscada.*

Tabla hash cerrada :: Ejemplo

- Consideremos una **ED** para almacenar pares <*Integer,Integer*>, con una **tabla hash** compuesta por un **arreglo** de **10 buckets** y utilizando una política de **resolución lineal** de colisiones.



- *Al momento de **buscar**, o **eliminar** pares, hay que tener en cuenta la política utilizada para **determinar correctamente** cuándo se debe **continuar buscando**, o cuándo no.*
- *Consideremos el siguiente ejemplo donde se quieren eliminar entradas de la tabla.*

Par <Clave, Valor>	H(Clave) = Clave mod 10
< 53420, 10>	H(53420) = 0
< 66554,6 >	H(66554) = 4
< 45434, 9 >	H(45434) = 4

- *Se buscará en todos los próximos buckets (linealmente) hasta bien se halle la clave buscada, un bucket referenciando a null, o bien se haya considerado toda la tabla.*
- *En este caso la búsqueda finaliza porque se halla la clave buscada.*
- *Luego, se elimina dicha entrada indicando que el bucket está disponible.*

Tabla hash cerrada :: Implementación

- Independientemente del **TDA** que se implemente, la **ED subyacente** de una **tabla hash cerrada** consideraría:

```
public class MiTDAConHashCerrado<K, V> implements MiInterfazTDA<K,V> {
  protected Entrada<K,V> [] arreglo;
  protected int tamañoInicial = 13;

  protected final Entrada<K,V> bucket_no_usado = new Entrada<K,V>(null,null);
  protected final Entrada<K,V> bucket_disponible = new Entrada<K,V>(null,null);

  public MiTDAConHashCerrado(){
    arreglo = (Entrada<K,V>[]) new Entrada [tamañoInicial];
    for(int i=0; i<tamañoInicial; i++){
      arreglo[i] = bucket_no_usado;
    }
  }
  protected int hash(K clave){
    return Math.abs(clave.hashCode() % arreglo.length);
  }

  //Operaciones propias del TDA
}
```

El arreglo de buckets de un tamaño fijo inicial.

Los objetos comunes para referenciar un bucket no usado o disponible.

Un constructor que se encargue de instanciar cada uno de los buckets, indicando que nunca se utilizaron.

La definición de la función hash que se encarga de tomar una clave y determinar la posición (bucket) ubicará en la tabla hash.

Tabla hash cerrada :: Implementación

- El **control** del **factor de carga** se realiza al momento de **agregar** elementos a la **tabla hash.**
- En caso de que se **supere** el **factor de carga**, se debe **redimensionar** el **arreglo de buckets**.

```
Algoritmo insertar(Clave, Valor):
  Entrada ← nueva entrada(clave,valor)
  Si ((cantidad_pares / arreglo.length) >= fc)
    redimensionar()
  Fin_si
  Bucket ← hash(Clave)
  insertar_segun_politica(Entrada, Bucket)
Fin
```

```
Algoritmo redimensionar():
  NuevaDimension ← próximo_primo(cantidad_pares * 2)
  ArregloAnterior ← Arreglo
  Arreglo ← inicializar arreglo de NuevaDimension buckets
  Desde i←0 hasta ArregloAnterior.length - 1
    Entrada ← ArregloAnterior[i]
    Si (Entrada no es null) y (Entrada no es disponible)
      Bucket ← hash(Entrada.obtenerClave())
      insertar_segun_politica(Entrada, Bucket)
    Fin_Si
  Fin_Desde
Fin
```

- ***proximo_primo(n)*** *retorna un primo m>n.*
- *En la* ***redimensión****, y como una buena política,* ***no se crean*** *entradas nuevas, sino se* ***reutilizan*** *las ya creadas.*
- *Hash(Entrada.obtenerClave()) se* ***computa*** *respecto al nuevo arreglo que tiene al menos el* ***doble*** *de la dimensión que el arreglo anterior.*

Fin de la presentación.