Estructuras de Datos
Clase 9 – Tablas de Hash

Dr. Sergio A. Gómez
http://cs.uns.edu.ar/~sag

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, Argentina

Tablas de Hash: Motivaciones

Se desea tener una implementación versátil de conjuntos, diccionarios y mapeos donde las operaciones de inserción, recuperación y eliminación tengan un tiempo esperado constante.

Dos implementaciones posibles:

1. Tabla de hash abierto (separate chaining en GT)
2. Tabla de hash cerrado (open addressing en GT)

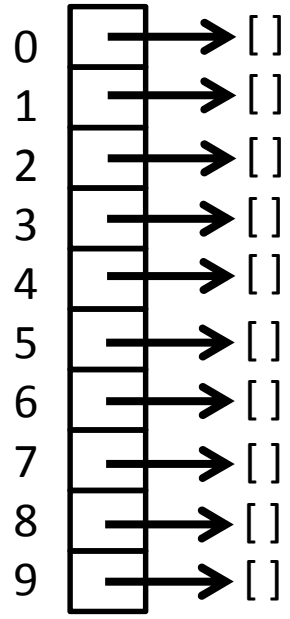
Hash Abierto: Definiciones

- Arreglo de buckets: Un arreglo de cubetas para implementar una tabla de hash es un arreglo A de N componentes, donde cada celda de A es una colección de pares clave-valor.
- Función de hash h: Dada una clave k y un valor v, h(k) es un entero en el intervalo [0,N-1] tal que la entrada (k,v) se inserta en la cubeta A[h(k)].
- Colisión: Dadas dos claves k_1 y k_2 tales que $k_1 \neq k_2$, se produce una colisión cuando $h(k_1) = h(k_2)$.
- Nota: Las entradas de una cubeta tienen claves colisionadas.
- "Buena" función de hash: Una función de hash h es "buena" si h distribuye las claves *uniformemente* en el intervalo [0,N-1].
- Intuititivamente, todas las cubetas tienen aproximadamente el mismo tamaño.

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N (Nota: N debiera ser primo, e.g. 11, o 13, o 17, etc.; usamos 10 para hacer fácilmente los cálculos en el ejemplo). Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();



Hay 10 buckets (cubetas) numeradas de 0 a 9. Cada uno almacenará una colección de entradas. Cada bucket se inicializa con una lista vacía (o podría ser null para ahorrar memoria)

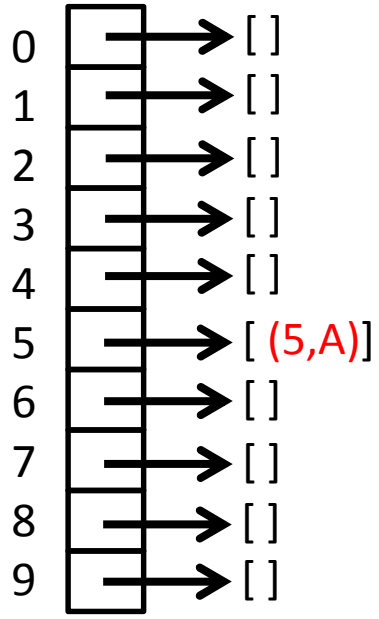
Nota: N en realidad debe ser primo, usaremos N=10 para facilitar los cálculos en el ejemplo.

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();
M.put(5, 'A')



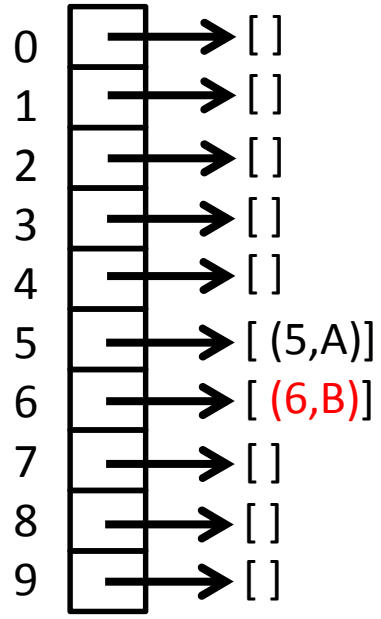
Como h(5) = 5 mod 10 = 5, la entrada (5,A) se inserta al final de la lista A[5].

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();
M.put(5, ‘A’)
M.put(6, ‘B’)



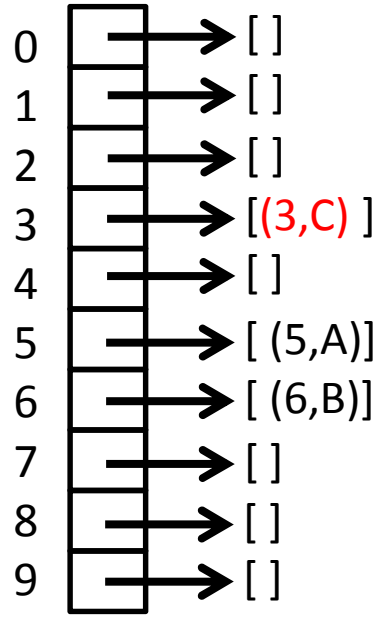
Como h(6) = 6 mod 10 = 6, la entrada (6,B) se inserta al final de la lista A[6].

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();
M.put(5, ‘A’)
M.put(6, ‘B’)
M.put(3, ‘C’)



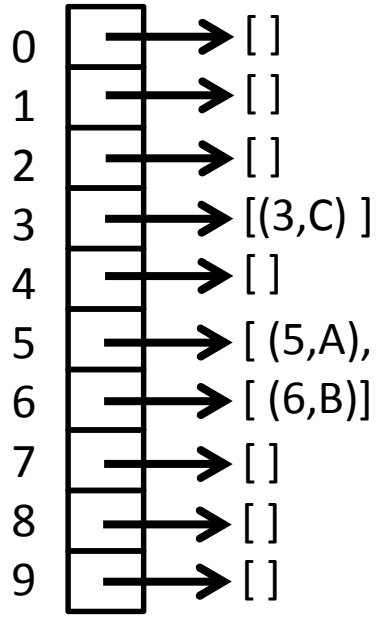
Como h(3) = 3 mod 10 = 3, la entrada (3,C) se inserta al final de la lista A[3].

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();
M.put(5, ‘A’)
M.put(6, ‘B’)
M.put(3, ‘C’)
M.put(15, ‘D’)



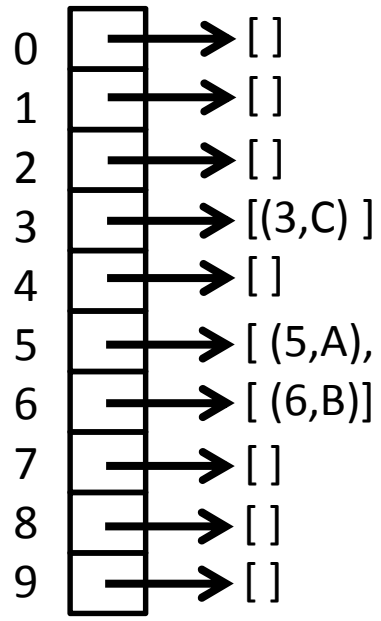
Como h(15) = 15 mod 10 = 5, la entrada (15,D) se inserta al final de la lista A[5]. Hubo colisión.

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

```
M ← new MapeoConHash();
M.put( 5, 'A' )
M.put( 6, 'B' )
M.put( 3, 'C' )
M.put( 15, 'D' )
M.put( 25, 'E' )
```



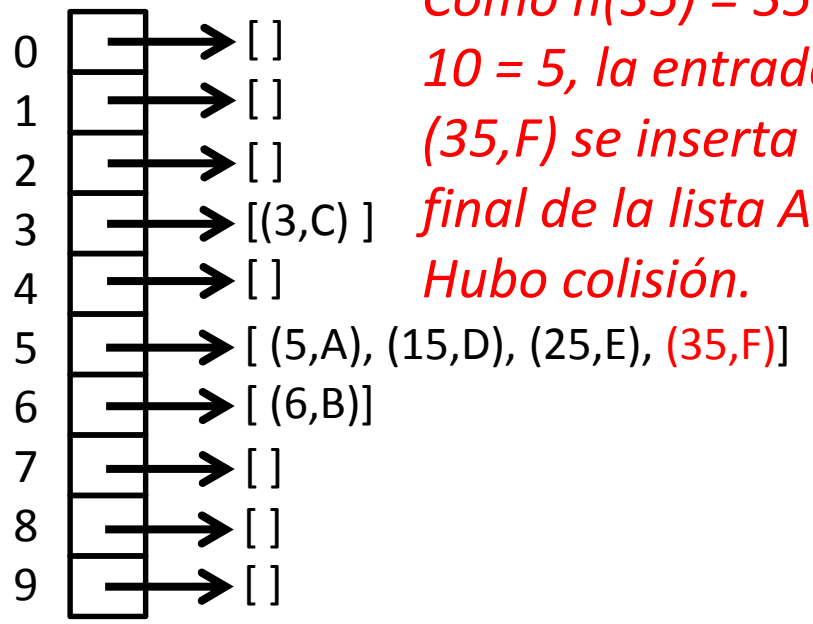
Como h(25) = 25 mod 10 = 5, la entrada (25,E) se inserta al final de la lista A[5]. Hubo colisión.

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

```
M ← new MapeoConHash();
M.put( 5, 'A' )
M.put( 6, 'B' )
M.put( 3, 'C' )
M.put( 15, 'D' )
M.put( 25, 'E' )
M.put( 35, 'F' )
```



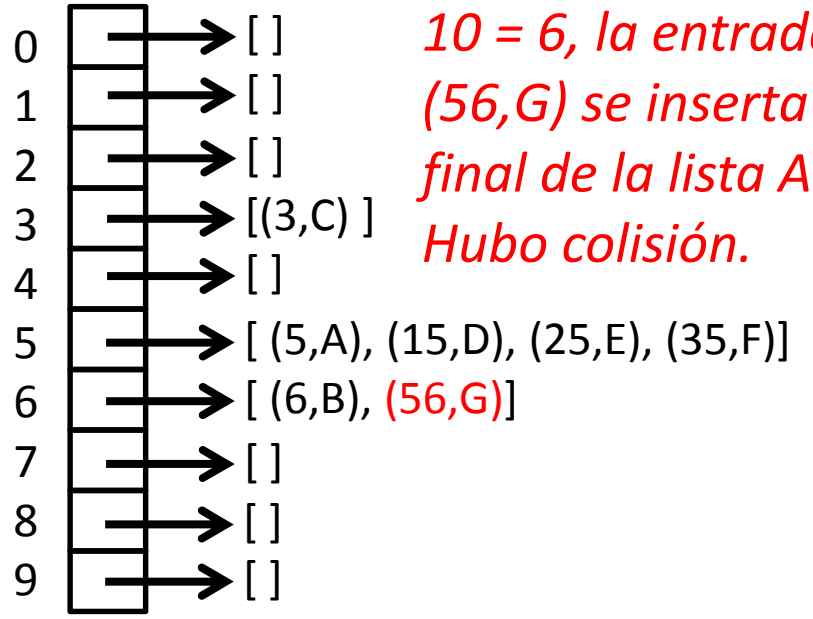
Como h(35) = 35 mod 10 = 5, la entrada (35,F) se inserta al final de la lista A[5]. Hubo colisión.

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();
M.put(5, ‘A’)
M.put(6, ‘B’)
M.put(3, ‘C’)
M.put(15, ‘D’)
M.put(25, ‘E’)
M.put(35, ‘F’)
M.put(56, ‘G’)



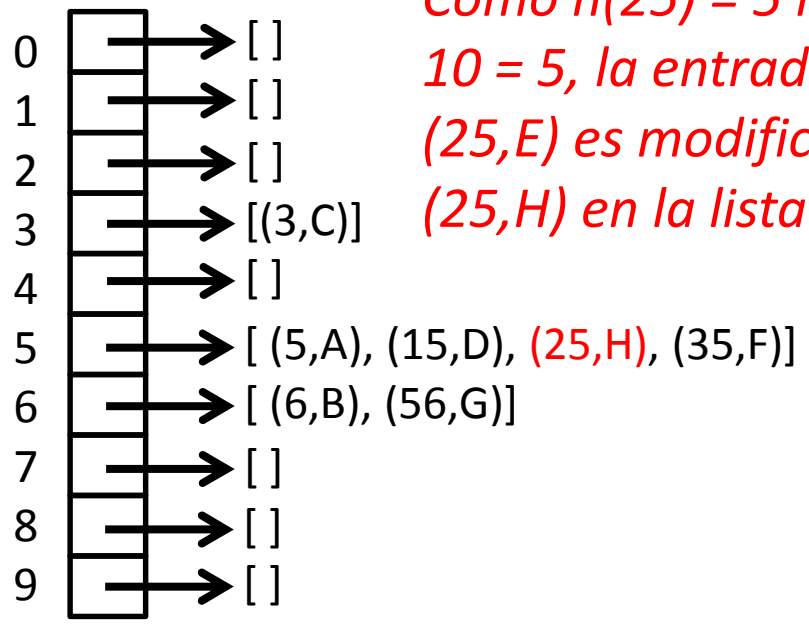
Como h(56) = 56 mod 10 = 6, la entrada (56,G) se inserta al final de la lista A[6]. Hubo colisión.

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

```
M ← new MapeoConHash();
M.put( 5, ‘A’ )
M.put( 6, ‘B’ )
M.put( 3, ‘C’ )
M.put( 15, ‘D’ )
M.put( 25, ‘E’ )
M.put( 35, ‘F’ )
M.put( 56, ‘G’ )
M.put( 25, ‘H’ )
```



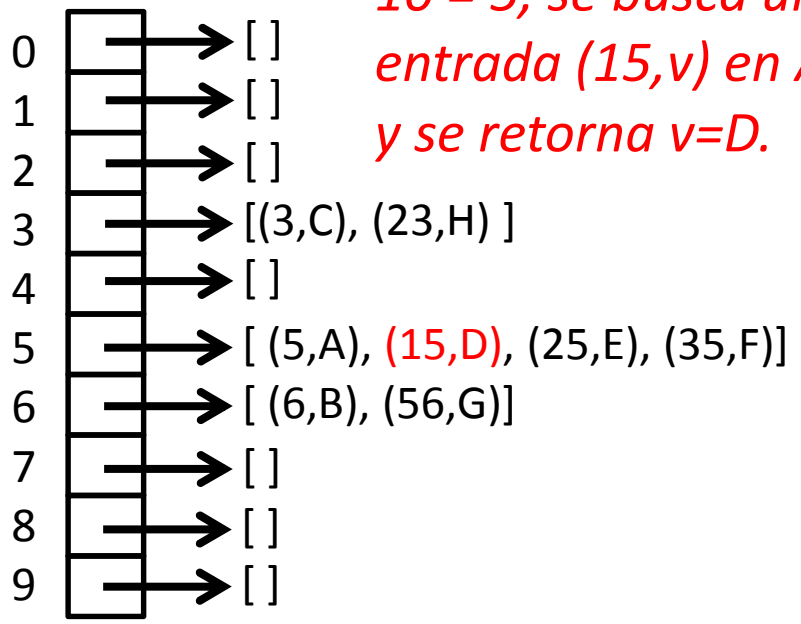
Como h(25) = 5 mod 10 = 5, la entrada (25,E) es modificada a (25,H) en la lista A[5].

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();
M.put(5, ‘A’)
M.put(6, ‘B’)
M.put(3, ‘C’)
M.put(15, ‘D’)
M.put(25, ‘E’)
M.put(35, ‘F’)
M.put(56, ‘G’)
M.put(23, ‘H’)
C ← M.get(15)



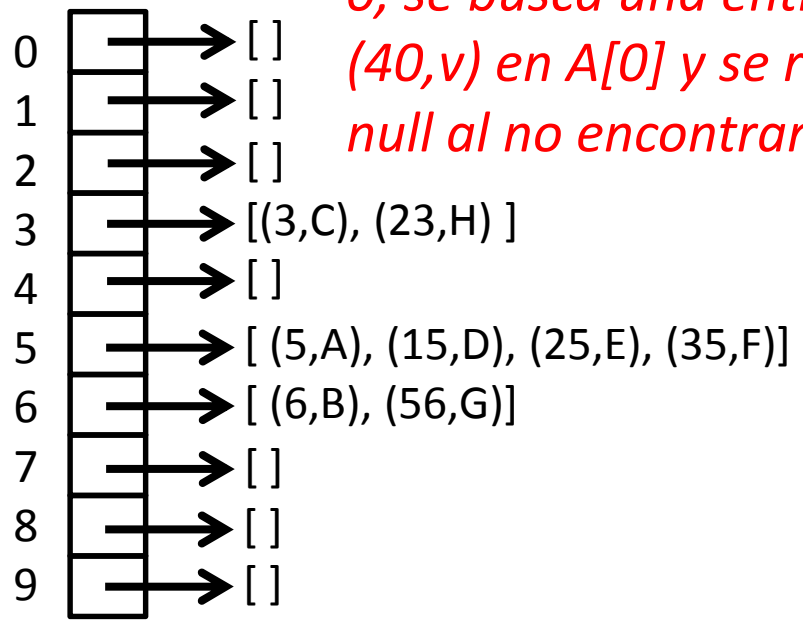
Como h(15) = 15 mod 10 = 5, se busca una entrada (15,v) en A[5] y se retorna v=D.

Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();
M.put(5, ‘A’)
M.put(6, ‘B’)
M.put(3, ‘C’)
M.put(15, ‘D’)
M.put(25, ‘E’)
M.put(35, ‘F’)
M.put(56, ‘G’)
M.put(23, ‘H’)
C ← M.get(15) ; C ← M.get(40)



Como h(40) = 40 mod 10 = 0, se busca una entrada (40,v) en A[0] y se retorna null al no encontrarla.

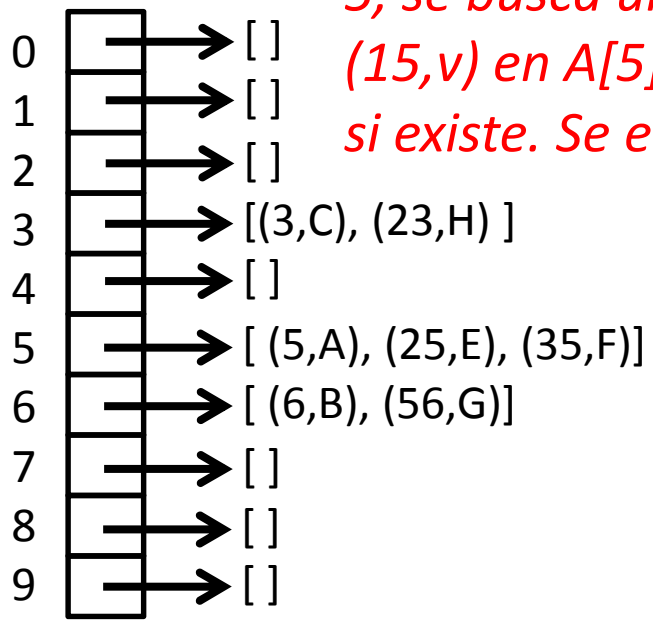
Ejemplo preliminar

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

Como h(15) = 15 mod 10 = 5, se busca una entrada (15,v) en A[5] y se elimina si existe. Se elimina (15,D)

M ← new MapeoConHash();
M.put(5, ‘A’)
M.put(6, ‘B’)
M.put(3, ‘C’)
M.put(15, ‘D’)
M.put(25, ‘E’)
M.put(35, ‘F’)
M.put(56, ‘G’)
M.put(23, ‘H’)



C ← M.get(15) ; C ← M.get(40); M.remove(15)

Problemas pendientes

- Hashing con claves genéricas:
  - ¿Cómo implementar h(k) cuando k no es entero?
- Distribución uniforme de claves:
  - ¿Qué significa?
  - ¿Cómo lo implemento?

Hashing con claves genéricas

Paso 1 (hash code): Dada una clave k, obtener un número entero llamado *código de hash* a partir de k.

Paso 2 (función de compresión): A partir del código de hash obtener un valor entero entre 0 y N-1.

Códigos de hash

- La forma más sencilla es usar el método:

  public int hashCode();

  heredado de Object.
- Cuidado: En la implementación más simple *hashCode()* retorna la dirección de memoria del objeto con lo que dos objetos iguales (en el sentido de *equals*) tendrían distinto hashCode.
- Por ello la clase String redefine el método *hashCode()*.

Códigos de hash para strings

- La forma más sencilla de implementar hashcode para strings consiste de sumar los códigos Ascii (o Unicode) de los caracteres del string.
- Ejemplo: El hash code de “ABBA” sería: 65+66+66+65 = 262
- Sin embargo, el código de “BABA” sería el mismo, con lo cual habría una colisión.

Códigos de hash para strings

- Una aproximación mejor consiste en utilizar la posición del carácter considerado en un código polinomial.
- Si el string s es $(x_0, x_1, \ldots, x_{k-1})$, dado un entero $a \neq 1$, se puede usar el hash code:

$$x_0 a^{k-1} + x_1 a^{k-2} + \ldots + x_{k-2} a + x_{k-1}$$

Nota: El valor de "a" se debe encontrar experimentalmente hasta lograr una "buena" distribución de claves.

Códigos de hash para tipos numéricos

- Para el tipo Float:

  Dado un float x existe un método de clase Float.floatToIntBits(x) que retorna un entero a partir de x.

  Ejemplo:

  - Float.floatToIntBits(4.5697f): 1083325179
  - Float.floatToIntBits(1.8e34f): 2019417596
- Para los tipos byte, short, int, char: Castear a int.

Funciones de compresión

Dado i = k.hashCode(), se requiere en la implementación de:

- Put(k,v): obtener el número de cubeta a insertar la entrada (k,v)
- Get(k): obtener el número de cubeta para buscar una entrada con clave k.

- Implementaciones:
  - Método de la división: i mod *N* (*N* debe ser primo)
  - Método MAD (loco) (por multiply, add & divide):

$$[(ai+b) \bmod p] \bmod N,$$

donde a y b son enteros al azar y p es un primo mayor a N. Nota: “a”, “b” y “p” deben buscarse experimentalmente hasta lograr una distribución uniforme de claves.

Implementación de separate chaining

Clase Mapeo<K,V>

Atributos:

```
Map<K,V> [] A
int  n          { n es size, la cantidad de entradas del mapeo }
int  N ← 13  { o cualquier otro número primo }
```

Constructor Mapeo()

```
n ← 0
A ← (Map<K,V> []) new MapeoConLista[N]
for i← 0 to N-1 do
        A[i] ← new MapeoConLista<K,V>()
```

Implementación de separate chaining

Algoritmo get(k)
{ retorna el valor asociado con la clave k en el mapeo, o null si no hay una entrada con clave k en el mapeo }
return A[h(k)].get(k) { Delega en el get del mapeo implementado con lista. }

Algoritmo put(k, v)
{ Si hay una entrada en el mapeo con clave k, reemplaza el valor con v y retorna el viejo valor. Sino retorna agrega la entrada (k,v) y retorna null }
t ← A[h(k)].put(k,v) { Delega en el put del mapeo implementado con lista. }
if t = null **then** { k es una nueva clave }
    n ← n+1
return t

Algoritmo remove(k)
{ Retorna el valor asociado con la clave k del mapeo o null si no hay entrada con clave k }
t ← A[h(k)].remove(k) { Delega en el mapeo implementado con lista. }
if t ≠ null **then** { k fue encontrado }
    n ← n – 1
return t

¿Distribución uniforme de claves?

- Fenómeno aleatorio: Fenómeno que bajos mismas condiciones puede producir resultados diferentes.
- Evento: Resultado posible de un fenómeno aleatorio.
- Espacio muestral: Conjunto de eventos posibles.
- Ejemplo: Al arrojar un dado, los eventos posibles son que salga la cara 1, que salga la cara 2, …, ó que salga la cara 6.
- Probabilidad: La probabilidad de un evento E, notada p(E), es un número real entre 0 y 1.
- Ejemplo: Dado perfecto: p(salga la cara i al arrojar el dado) = 1/6
- Frecuencia relativa: La probabilidad de un evento E se estima con su frecuencia relativa = cantidad de ocurrencias del evento E / cantidad de eventos ocurridos
- Ejemplo: Dado del casino: ¿Cuál es la probabilidad de obtener un 2 al tirar el dado? p(salga la cara 2 al arrojar el dado) =

  cuántas veces salió el 2 / la cantidad de tiradas.

¿Distribución uniforme de claves?

- Distribución uniforme: Todos los eventos tienen la misma probabilidad.
- Distribución uniforme de claves: La probabilidad de que una clave termine en un bucket determinado es constante y es igual a 1/N, donde N=cantidad de buckets de la tabla de hash.
- ¿Cómo compruebo que tengo distribución uniforme de claves?:
  1. Tome un conjunto de n claves de muestra k_1, k_2, …, k_n
  2. Realice los n *put(k_1), put(k_2), …, put(k_n)* en su tabla de hash
  3. Cuente cuántas claves (o entradas) hay en cada bucket
  4. Si hay distribución uniforme de claves, entonces en cada bucket debería haber aproximadamente n / N entradas.

Tiempo de ejecución de hash abierto

- Entrada: Tabla de hash de N buckets y n entradas.
- Tamaño de la entrada: n = cantidad de entradas del hash
- Suponemos que no hay distribución uniforme de claves
- Peor caso:
  - Ocurre el peor escenario: Todas las claves terminan en un único bucket
  - En este caso el bucket es una lista de tamaño n
  - $T_{get}(n) = O(1+n)$,
  - $T_{put}(n) = O(1+n)$
  - $T_{remove}(n) = O(1+n)$
  - Nota: El "1" en "1+n" corresponde al tiempo de calcular el hash de la clave, el cual es independiente de n.

Tiempo de ejecución de hash abierto

- Entrada: Tabla de hash de N buckets y n entradas.
- Tamaño de la entrada: n = cantidad de entradas del hash
- Suponemos que sí hay distribución uniforme de claves
- Peor caso:
  - Cada bucket tiene n/N entradas
  - En este caso el bucket es una lista de tamaño n/N
  - $T_{get}(n) = O(1+n/N)$
  - $n/N = \lambda$ = factor de carga < 0,9
  - Luego, $T_{get}(n) = O(1+n/N) = O(1+\lambda) = O(1,9) = O(1)$

Rehash

- Al implementar put, hay que mantener el factor de carga $\lambda=n/N$ controlado.
- Si λ supera 0,9, hay que incrementar el tamaño de N para disminuir n/N.
- Esto requiere crear una tabla más grande (e.g. de tamaño 2*N y buscar el siguiente primo N' tal que N'>2*N); luego insertar todas las entradas de la tabla vieja en la nueva tabla (*a resolver en la práctica*).
- Esta operación inevitablemente tendría O(n) ya que requiere recorrer toda la tabla vieja.

Hash Cerrado

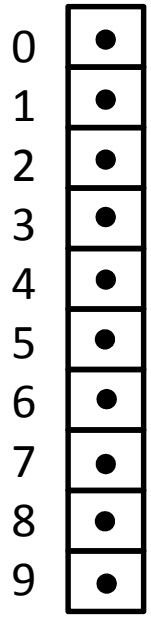
- En hash cerrado se tiene un arreglo de N buckets
- Cada bucket almacena a lo sumo una entrada.
- Dada una clave k y un valor v, la función de hash h indica cuál es la componente h(k) del arreglo en la cual se almacena la entrada (k,v).
- Dadas dos claves k_1 y k_2 con $k_1 \neq k_2$, si $h(k_1)=h(k_2)$ entonces se produce una colisión.
- Políticas para la resolución de colisiones:
  - Lineal (linear probing)
  - Cuadrática (quadratic probing)
  - Hash doble (double hashing)

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash();

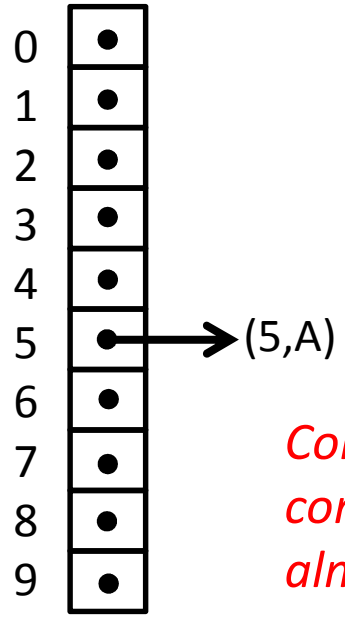


Inicialmente las componentes de la tabla de hash están inicializadas con null indicando que están vacías.

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.
Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')



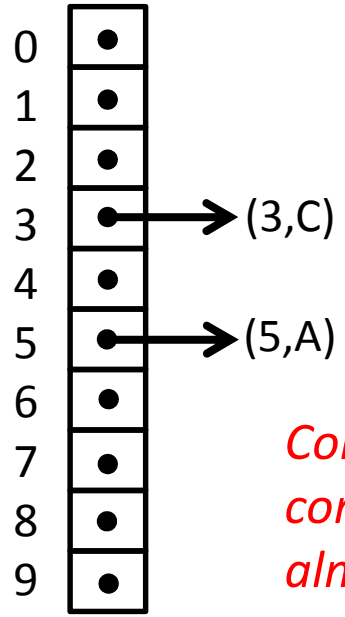
Como h(5) = 5 mod 10 = 5 y la comp. 5 es null, directamente almaceno (5,A) en A[5].

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')



Como h(3) = 3 mod 10 = 3 y la comp. 3 es null, directamente almaceno (3,C) en A[3].

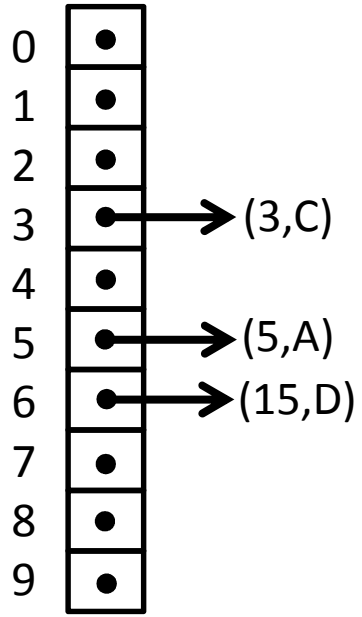
Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')
M.put(15, 'D')

Como h(15) = 15mod 10 = 5 y la comp. 5 está ocupada, se produjo una colisión. Se busca una componente null o disponible en forma lineal y circular. Almaceno (15,D) en A[6].

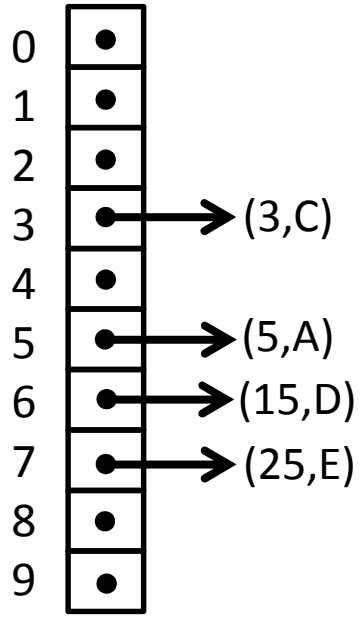


Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')
M.put(15, 'D')
M.put(25, 'E')



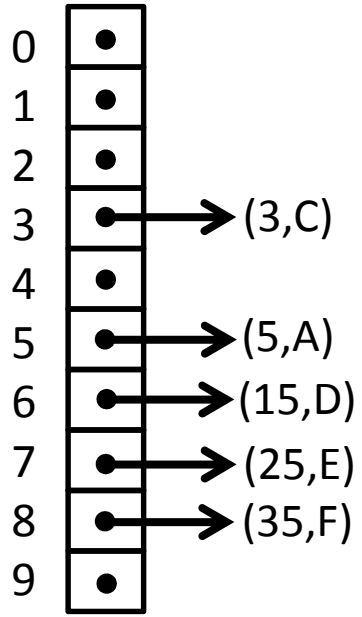
h(25) = 25 mod 10 = 5, y la comp. 5 está ocupada, se produjo una colisión. Se busca una componente null o disponible en forma lineal y circular. Almaceno (25,E) en A[7].

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')
M.put(15, 'D')
M.put(25, 'E')
M.put(35, 'F')



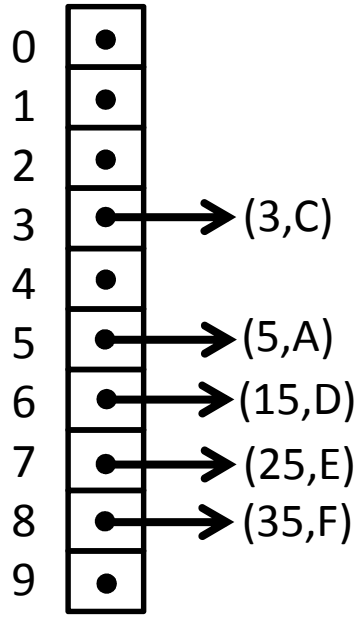
h(35) = 35 mod 10 = 5, y la comp. 5 está ocupada, se produjo una colisión. Se busca una componente null o disponible en forma lineal y circular. Almaceno (35,F) en A[8].

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')
M.put(15, 'D')
M.put(25, 'E')
M.put(35, 'F')
v ←M.get(25)



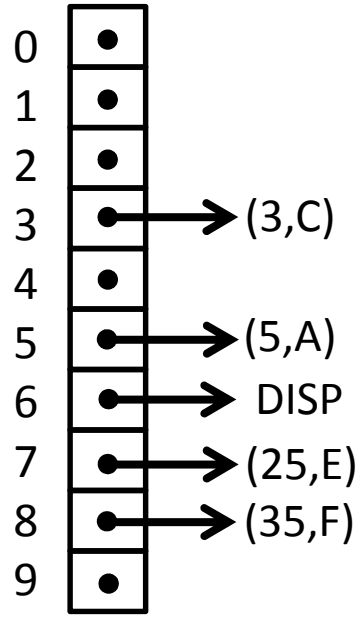
h(25) = 25 mod 10 = 5, y la comp. 5 no tiene la clave 25, se busca linealmente la clave 25 hasta encontrarla o encontrar una componente null en forma lineal y circular. Encuentro la componente 7 y retorno el valor A[7].getValue().

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')
M.put(15, 'D')
M.put(25, 'E')
M.put(35, 'F')
v ←M.get(25)
M.remove(15)



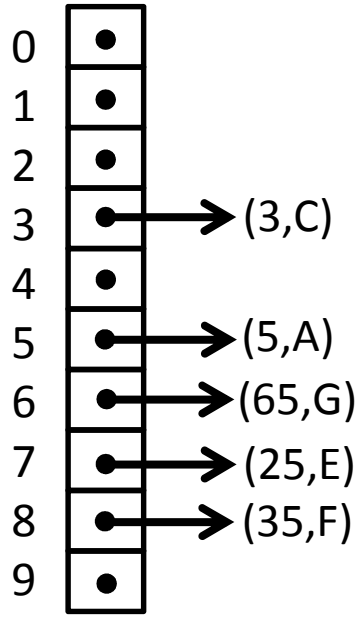
h(15) = 15 mod 10 = 5, y la comp. 5 no tiene la clave 15, se busca linealmente la clave 15 hasta encontrarla o encontrar una componente null en forma lineal y circular. Encuentro la componente 6 y retorno el valor A[6].getValue() pero marcando A[6] como disponible. Si pusiera null no encontraría (25,E) ni (35,F).

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')
M.put(15, 'D')
M.put(25, 'E')
M.put(35, 'F')
v ←M.get(25)
M.remove(15)
M.put(65, 'G')



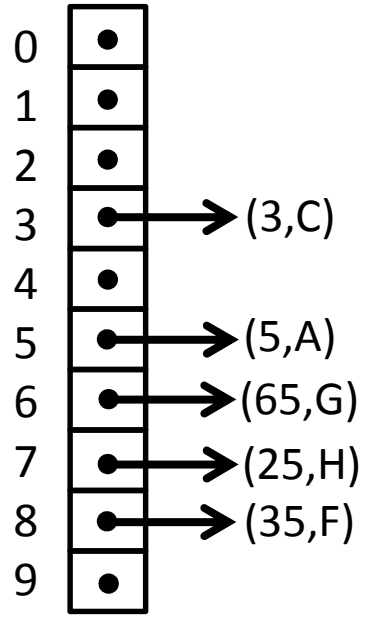
h(65) = 65 mod 10 = 5, y la comp. 5 está ocupada (hubo colisión). Se busca linealmente hasta encontrar un null o un disponible en forma lineal y circular. Encuentro la componente 6 y allí almaceno (65,G).

Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

M ← new MapeoConHash()
M.put(5, ‘A’)
M.put(3, ‘C’)
M.put(15, ‘D’)
M.put(25, ‘E’)
M.put(35, ‘F’)
v ←M.get(25)
M.remove(15)
M.put(65, ‘G’)
M.put(25,’H’)



h(25) = 25 mod 10 = 5, y la comp. 5 no tiene la clave 25, busco linealmente hasta un null o hasta un disponible o hasta hallar el 25, donde reemplazo el valor viejo E por el nuevo H y retorno el viejo E.

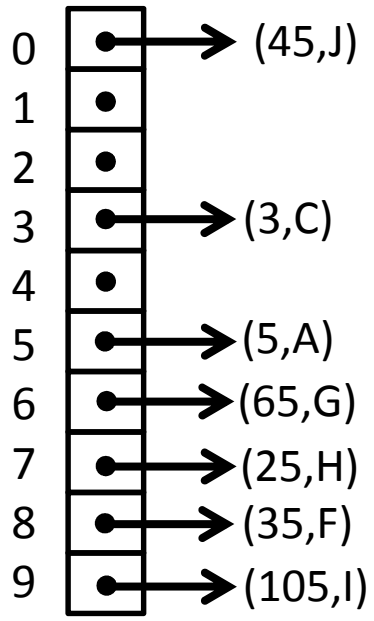
Ejemplo preliminar usando resolución lineal de colisiones

Supongamos que el tamaño del arreglo de bucket N=10 y que la función de hash es h(k) = k MOD N.

Veamos qué sucede con la siguiente secuencia de operaciones sobre un mapeo de enteros en caracteres.

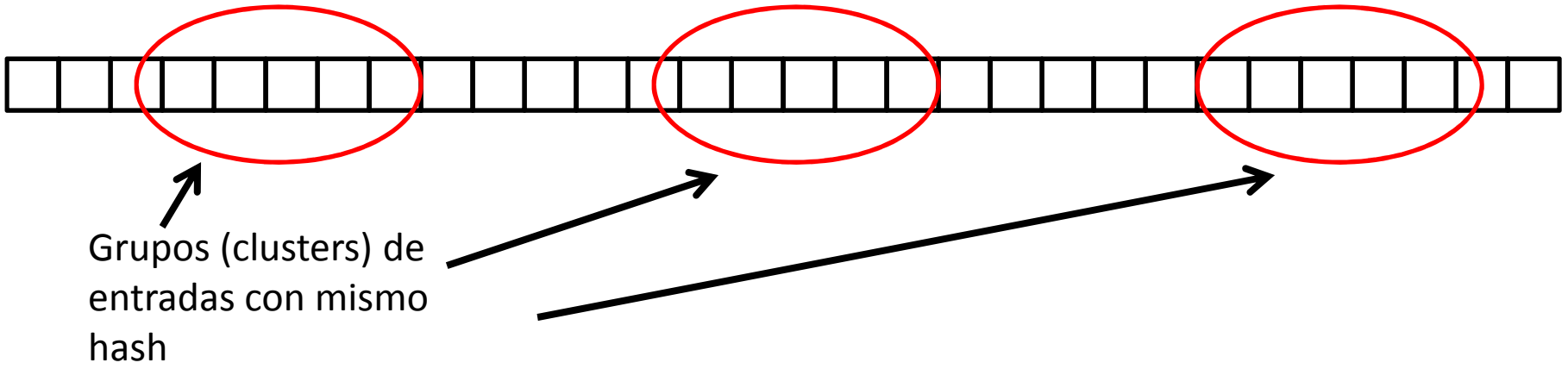
La inserción de (105,I) y (45,J) pueblan el arreglo en forma circular.

M ← new MapeoConHash()
M.put(5, 'A')
M.put(3, 'C')
M.put(15, 'D')
M.put(25, 'E')
M.put(35, 'F')
v ←M.get(25)
M.remove(15)
M.put(65, 'G')
M.put(25,'H'); M.put(105, 'I'); M.put(45, 'J')



Ventajas y desventajas

- El hashing lineal tiende a agrupar entradas con claves con mismo valor de hash.
- Si el arreglo es muy pequeño, se mezclan las claves k_i con distinto hash y la búsqueda se hace muy lenta porque degenera en buscar linealmente en un arreglo.
- Para hacer que esto funcione N >> n con lo que las claves se agrupan por su valor de hash
- GT sugiere que el factor de carga $\lambda = n/N < 0.5$



Otras políticas de resolución de colisiones: Resolución cuadrática (quadratic probing)

- Definición: Si i=h(k), prueba iterativamente los buckets A[(i+f(j)) mod N] para j=0,1,2,… donde $f(j)=j^2$ hasta que encuentra un bucket vacío.
- Ventaja: Evita los patrones de agrupamiento de hash lineal pero crea otro tipo que se llama “agrupamiento secundario”
- Problema: Si N no es primo, puede no encontrar un slot vacío aunque exista
- Problema: Si N es primo y el arreglo está lleno a la mitad o más (i.e. n>N/2), no va a encontrar un slot vacío por más que éste exista.

Otras políticas de resolución de colisiones: Hashing doble (double hashing)

- Definición: Si i=h(k), y A[i] está ocupado, se usa una segunda función de hash h' tal que prueba los buckets A[(i + f(j)) mod N] para j=1,2,3… donde f(j) = j*h'(k).
- La función h'(k) no puede evaluar a 0.
- Se usa h'(k) = q – (k mod q) para un primo q < N.
- N debiera ser primo también.

Comparación entre ambos enfoques

- El hash cerrado ahorra algo de espacio con respecto al hash abierto.
- Los resultados experimentales muestran que el hash abierto es más rápido que los otros métodos.

Bibliografía

- Capítulo 9 de M. Goodrich & R. Tamassia, Data Structures and Algorithms in Java. Fourth Edition, John Wiley & Sons, 2006.