Secciones TQ - 1er Semestre
Marzo/2018

Tablas de Dispersión

Hash Tables

Prof. Cristian Cappo

¿Qué veremos en esta clase?

- Motivación con ejemplos
- Función de dispersión
- Tablas de dispersión
- Resolución de colisiones
- Ejercicios

- Suponga que cada ítem de dato tiene asociado una única clave en un rango en particular.

  Ejemplos:

  - Dirección IP (binario)
  - Cédula de identidad (decimal)

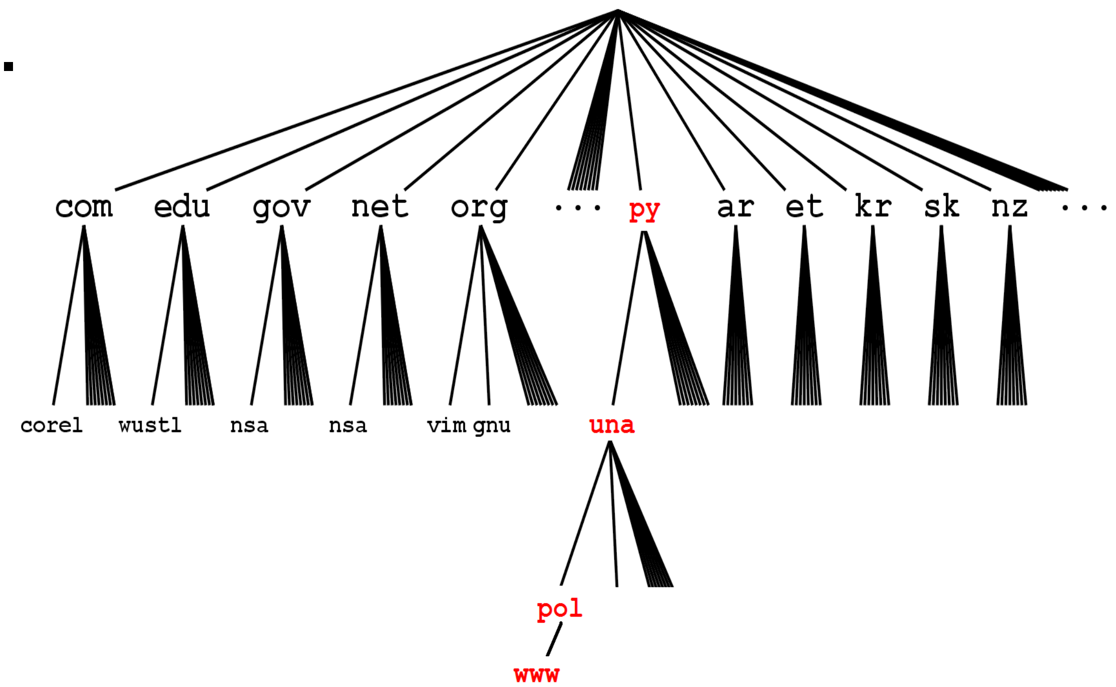
Analizaremos ambos casos

Ejemplo 1: Dirección IP

- Cada computadora en una red que utiliza IPv4 tiene una única dirección IP.
  - 32 bits/4 bytes que permite un poco más de 4 billones de direcciones $2^{32} = 4.294.967.296$
- El mismo esta representado como un conjunto de 4 bytes, de forma que sea legible para nosotros. Así
  - 200.10.229.161 es el web server de Politécnica
  - http://200.10.229.161 Es una dirección válida
- *No es fácil de recordar los números*

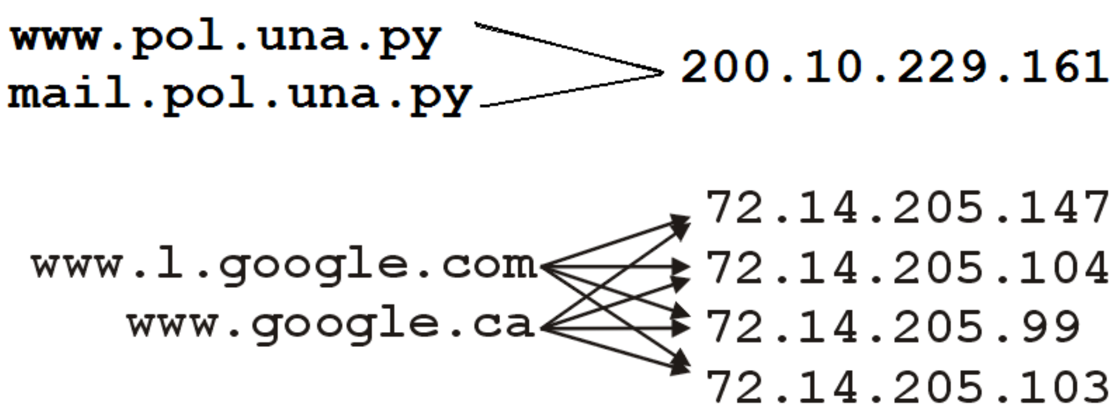
Dirección IP

- Los nombres de dominio fueron introducidos para poder fácilmente asociar el número con alguno nombre.
- DNS (*Domain Name System*) es el sistema que permite tal conversión. El mismo es jerárquico y distribuido.



Dirección IP

- El mapeo no suele ser uno a uno
  - Un IP puede corresponder a múltiples nombres de dominio.
  - Algún dominio puede tener varios IP



Dirección IP

- En el DNS cada rama tiene responsabilidad de sus propios nombres.
- Así la UNA tiene asignado un conjunto de números.
  - 200.10.228/22 , es decir, unos 2^{10} números equivalente a 1024 números.

Dirección IP

- Un posible mapeo sencillo de IP a nombres de dominios:
  - Un arreglo de tamaño 1024
  - Trasladar **200.10.229.161** a una única clave **$229*2^8$ + 161**.

	Indice (clave)	Dirección	Nombre de dominio
0	58369	200.10.228.1	sbd.cnc.una.py
1	58370	200.10.228.2	dev.cnc.una.py
..	...	..	..
131	58500	200.10.228.132	ns.cnc.una.py
..	..	..	..
416	58785	200.10.229.161	server.pol.una.py
..	..	..	..
1023	..	..	..

Dirección IP

- En este ejemplo la solución es clara:
  - Un arreglo fijo en tamaño
  - El arreglo esta casi lleno (denso)
  - El traslado de IP a nombre esta en *O(1)*
- ¿Cuáles son los problemas de este modelo?
  - ¿Qué pasa si el arreglo es muy esparcido? Desperdicio de espacio.
  - ¿*Podremos hacerlo de otra forma*?

Dirección IP

- Problemas:
  - No se utiliza todo el arreglo (no todas las direcciones tienen nombre).
  - ¿Que pasa si la cantidad utilizada es muy pequeña?
    - Por ejemplo, solo 100 máquinas tienen nombre
- Suponga ahora que vamos a utilizar direcciones IPv6 , que usa direcciones de 128 bits.
  - 2^{128} .. un número indecible (340 sextillones)

    340.282.366.920.938.463.463.374.607.431.768.211.456
- La UNA tiene una asignación de IPv6 de 2^{32} direcciones (2001:1320/32).. ¿podremos crear un arreglo de 4 billones de elementos?

Dirección IP

- ¿Podremos lograr algo mejor que O(log n)?
- ¿Podremos bajar a O(1)?
- Problema:
  - Si requerimos que las entradas este ordenadas, no podemos bajar de O(log n)
- ¿Necesitamos mantener orden en las entradas?
  - ¿Es importante conocer la dirección IP que viene alfabéticamente luego de *www.pol.una.py*?

Ejemplo 2: Nro. de cédula

- Dado el siguiente ejemplo:
  - Cada estudiante en POL se le identifica por su cédula.
  - Asignar un arreglo de más de seis millones no es razonable, verdad?
  - POL solo tiene unos 3000 estudiantes
  - Hay un poco más de 50 alumnos en esta clase
- Suponga que quiera guardar la nota asociada con cada estudiante en esta clase

Nro. de cédula

- **Solución**
  - Tener un arreglo de 1000 posiciones
  - Identificado por 000, 001, .., 999
  - Guardar la nota del estudiante con cedula 4562781 en la posición 781
    4562781 % 1000 = 781
- **Beneficio**
  - El módulo toma O(1)
  - Acceder al arreglo es O(1)
  - Solo 50 estudiantes: 1 de cada 20 cubetas esta ocupada.

...	CEDULA
..	...
781	4562781
782	
783	
...	
999	

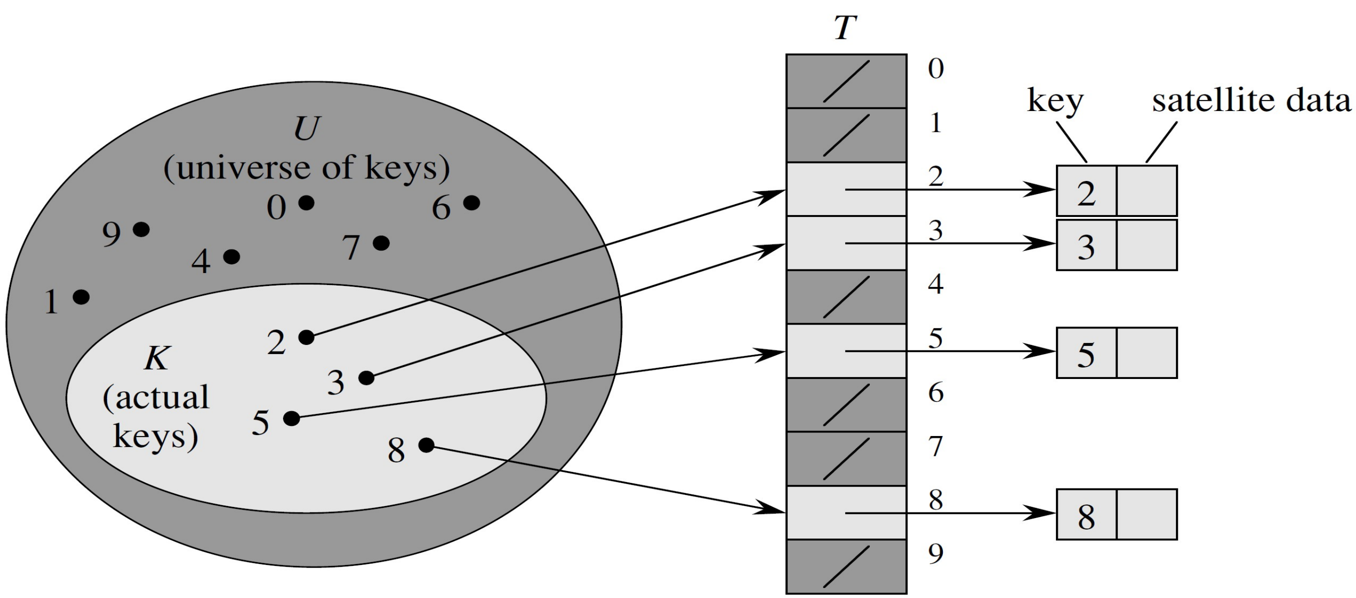
Nro. de cédula

- Problema:
  - Múltiples estudiantes puede tener los mismos últimos tres dígitos

    Asumiendo que la distribución de los últimos tres números es uniforme
    - Ejemplo: La probabilidad es 0,01% de que dos estudiantes de 1000 tengan el mismo número de cédula.

Utilizando una tabla de direccionamiento directo

- Es una técnica simple aplicable en un universo U pequeño de claves. Supongamos $U=\{0,1,..,m-1\}$ donde m no es grande.
- La solución es tener un arreglo T[0..m-1], en el que cada posición corresponde a una clave de U.



Direccionamiento directo

- Si |*U*| es grande, tener una tabla T de tamaño |*U*| puede ser impráctico y hasta imposible (como vimos en el ejemplo de Ipv6).
- Por otra parte el conjunto de claves guardadas actualmente puede ser mucho menor que |*U*| y el espacio ocupado por T podría ser mal utilizado.
- En este caso, cuando se requiere un tiempo O(1) para las operaciones de búsqueda, inserción y borrado, la estructura de datos que me conviene es una ***Tabla de Dispersión***

Dispersión y tabla de dispersión

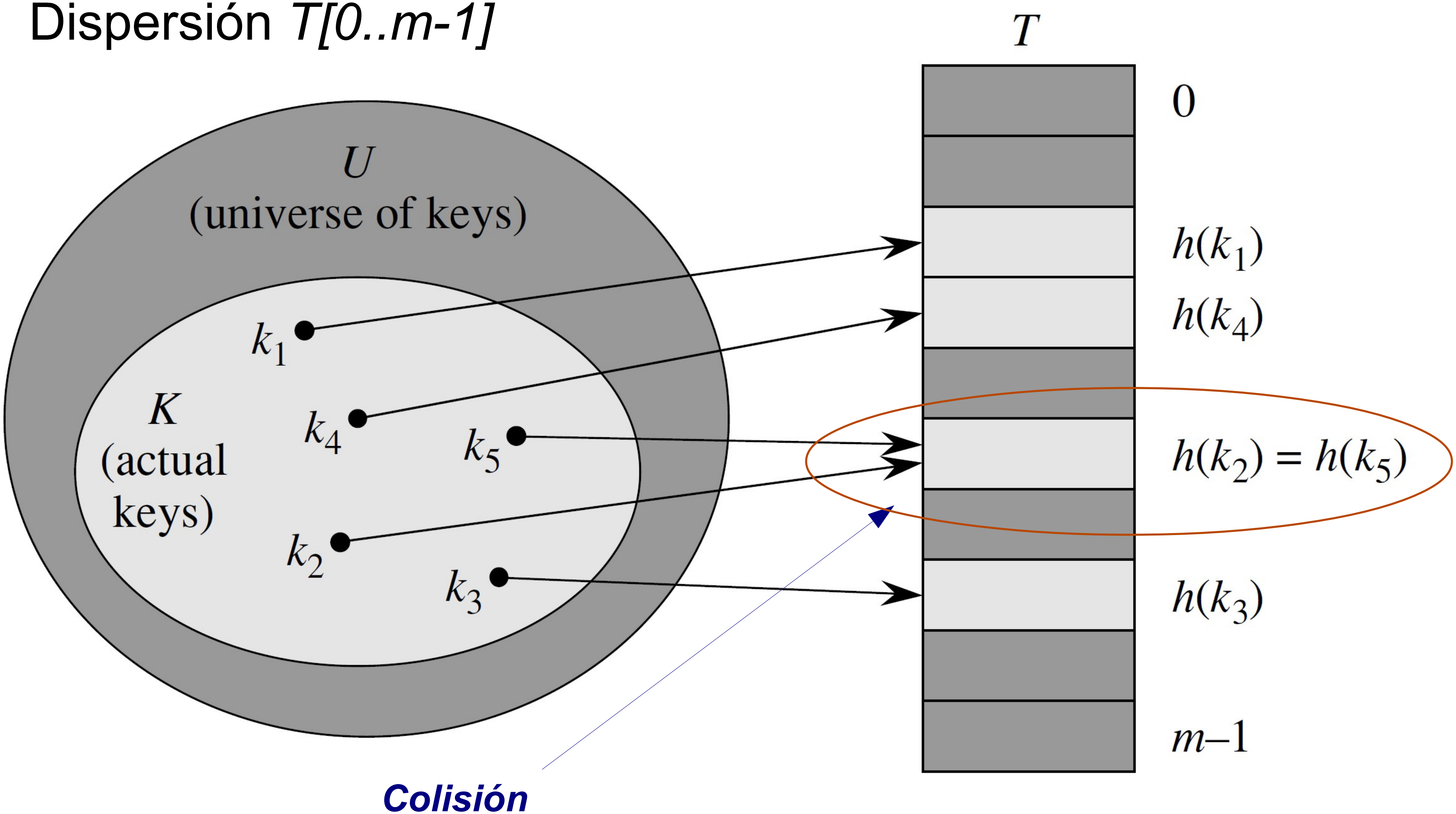
- Con el direccionamiento directo un elemento con clave *k* es guardada en la posición *k*. Con la tabla de dispersión es guardada en la posición ***h(k)***, esto es, utilizamos una función de dispersión o *función hash* ***h*** que calcula la posición a partir de la clave k.
- El proceso de mapear un número en un rango pequeño se denomina **Dispersión o Hashing.**
- Cuando ocurre que dos o más objetos pueden tener el mismo valor o clave Hash o de dispersión entonces se da una ***colisión***
- El arreglo que guarda los registros es denominada **Tabla de Dispersión**. Una posición en la tabla se llama **cubeta** o *slot*. Utiliza una función de dispersión (*función hash*) y un mecanismo para resolver las colisiones.
- Se lo puede ver como una generalización de la noción de un arreglo ordinario.

¿Cuándo usar Tablas de dispersión?

- Apropiado solo para conjuntos (no hay clave duplicada).
- No existe necesidad de:
  - Búsqueda por rango.
  - Visitar los datos en orden de clave.
- Eficiente para responder la pregunta "*¿Cual registro, si existe, tiene la clave K?*"

Tabla de dispersión

Así, $h\!:\!U \rightarrow \{0,1,...,m-1\}$ la función *h* mapea el universo U de claves en las posiciones de una tabla Hash o de Dispersión *T[0..m-1]*



Dirección IP a nombre

- ¿Cómo hacemos con la dirección IP de tipo IPv6?
  - Asumir que ahora usamos el 50% de Ipv4, es decir solo 512 números (recordar que tenemos 2^{10} asignables)
  - Tener un arreglo de 512 elementos
  - Definir una función que mapee la dirección de 128 bits a uno de los valores 0..511
  - Definir un mecanismo de colisión

Arreglos asociativos en PHP / Perl

Ejemplo en PHP

```
<?php
   $edad = array("Pedro"=>36, "Juan" => 40, "Maria" => 35);
   $edad['Pedro'] = 50;
   foreach ($edad as $e=>$valor) {
      echo "Nombre = "  . $e . " Edad = " . $valor;
      echo "\n";
   }
   $edad['Juan'] = 85;
   echo "Edad de Juan es ". $edad['Juan'] . "\n";
?>
```

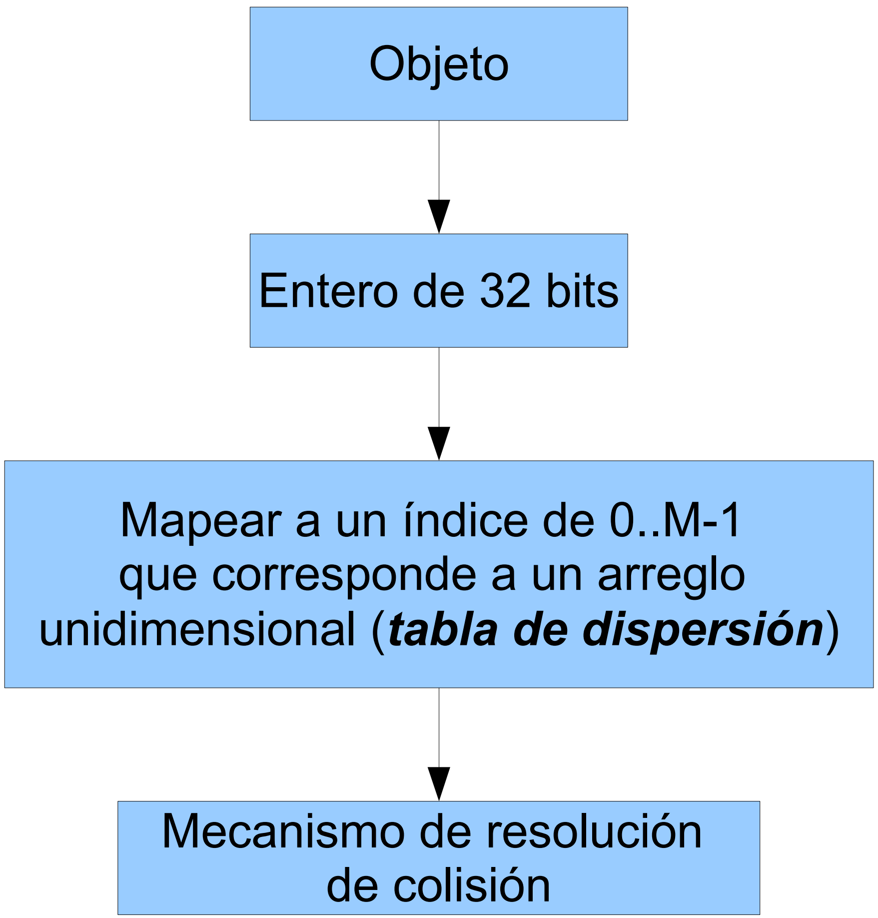
¿Cómo funciona?

Ejemplo en **Perl**

```
%edad = ('Pedro'=>36, 'Juan'=>40, 'Maria'=>35);
$edad{'Pedro'} = 50;
while(($nombre,$e) = each(%edad) ) {
   print "Nombre = ". $nombre . " Edad = " . $e;
   print "\n";
}
$edad{'Juan'} = 85;
print "Edad de Juan es " . $edad{'Juan'}. "\n";
```

Tablas de Dispersión

- Finalmente la idea es:



Técnicas varias
(representación numérica del objeto)

Mapear el entero calculado arriba, eficientemente a uno de los M valores. (*Función Hash*). Ejemplos: *método de división (módulo), método multiplicativo, hashing universal. Nosotros utilizamos el método de división.*

Dispersión Abierta

Dispersión Cerrada → Lineal
→ Cuadrática
→ Doble Hash

Función HASH

- ¿Qué es el HASH de un objeto?
  - HASH: *a restatement of something that is already known* (un *re-expresión* de algo que es conocido)

    *En realidad es una representación **corta** de un objeto*
- Finalmente lo que se quiere es mapearlo en un rango de valores discreto

  $0, 1, 2, .. , M-1$
- La función que hace este mapeo se denomina función HASH o función de dispersión.

Función Hash

- **Propiedades**
  - ***Es determinística***: Objetos iguales tienen el mismo valor HASH. La función h(k) siempre retorna el mismo valor para la misma *k*.
  - Objetos diferentes tienen diferente valor de HASH y son iguales con una probabilidad muy baja.
  - Rápida: O(1)
  - Distribución uniforme en 0.. M-1 (sin *agrupación o clustering*)
  - Objetos similares no deben tener el mismo valor de HASH (por ejemplo en una tabla de símbolos(?), la variable ***i*** y la variable ***j*** deben dar valores diferentes)

Función HASH

- Función de dispersión para Cadenas
  - Dos cadenas son iguales si los caracteres son iguales y están en el mismo orden
  - Una cadena es simplemente un arreglo de bytes. Cada byte tiene un valor de 0..255.
- Una función hash sobre una cadena debe ser una función sobre estos bytes

Función HASH

- Ejemplo de función HASH para cadena:
  - Sumar el valor ASCII de cada carácter de la cadena
  - ¿Que sucede? NO ES BUENA
    - Es lenta, ¿porqué?
    - No dispersa bien los valores, ¿porqué?

Función HASH

- Función HASH para una cadena:
  - Otra opción: considerar los primeros cuatros bytes como un número de 4 bytes.
  - Más rápida pero tiene un problema de agrupación, ¿porqué?

Función HASH

- Otra función podría ser,
  - Definir la cadena como un polinomio de grado *n-1*

$$p(x) = a_0 x^{n-1} + a_1 x^{n-2} + \cdots + a_{n-3} x^2 + a_{n-2} x + a_{n-1}$$

- Usamos la *regla de Horner* para evaluar en un número primo *x* = 37 (considero un número de base 37)

$$p(x) = a_0 + x\,(a_1 + x\,(a_2 + \cdots x\,(a_{n-2} + a_{n-1} x) \cdots))$$

Por ejemplo: considere el número en hexadecimal `7CE`.
Pasamos a decimal usando su forma de polinomio: $7.16^2 + 12.16^1 + 14.16^0 = 1998$
Usando la regla de Horner: $14 + 16\ (7.\ 16 + 12) = 1998$

```
for ( int k=0; k < s.length(); k++ )
      hash_value = hash_value * 37 + (s.charAt(k))
```

Función HASH

De nuevo es O(n)

- Como resolver: solo tomando algunos caracteres, por ejemplo, en la posición 2^k-1

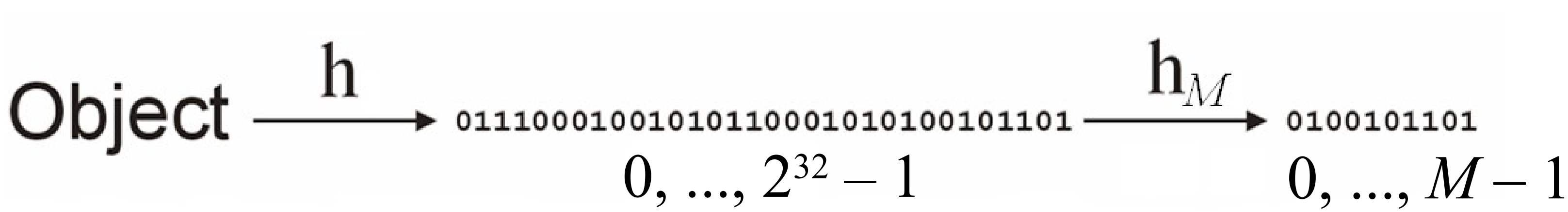
```
hash_value = 0;

for ( int k=1; k < s.length(); k*=2 )
     hash_value = hash_value * 37 + (s.charAt(k-1))
```

Ejemplos con los dos abordajes

Cadena	2^k-1	n
hola	3959	5423964
olla	**4215**	**5774428**
ola	**4215**	**156052**
cola	3774	5170699
oso	4222	156325
oca	4206	155719

Dispersión

- Una vez que tenemos el valor HASH del objeto debemos asignarle una posición en la tabla.
- Requerimos entonces tener un valor de 0..M-1 (donde M es el tamaño de la tabla)
- Normalmente se utiliza el módulo que es el método por división *h(k) = k mod M*



Dispersión

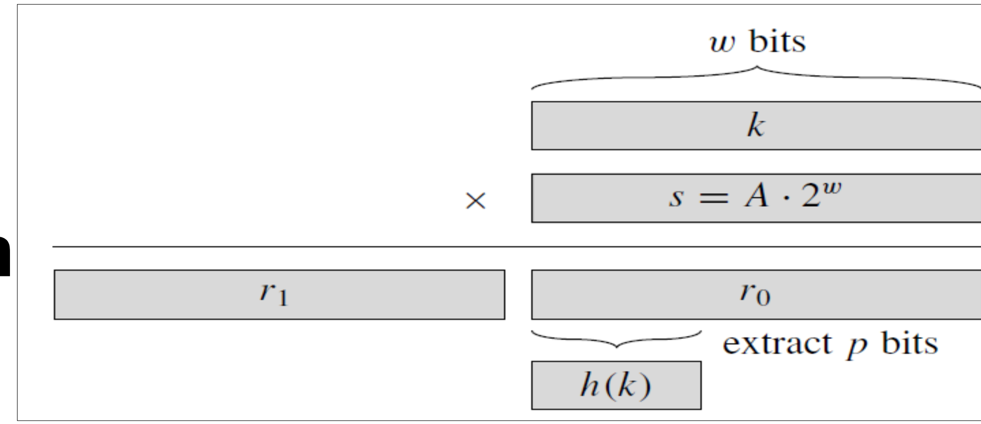
Método por división

$h(k) = k \bmod M$

- Evitamos ciertos valores de M
  - No debe ser una potencia de 2
    - Ya que si $m = 2^p$, entonces h(k) es solo los *p* bits más bajos de *k*
    - Es mejor hacer depender de todos los bits.
  - Es recomendable utilizar un número primo no tan cercano a una potencia de 2.
  - Es el que usaremos normalmente.

Dispersión

- Otros métodos
  - **Método por multiplicación**



$$h(k) = \lfloor M (k A \bmod 1) \rfloor$$

- Donde A es una constante $0 < A < 1$
  - Según Knuth $A \approx (\sqrt{5}-1)/2 = 0.6180339887..$
- Ventaja: la elección de M no es crítica
- A es un valor $s / 2^w$ para $0 < s < 2^w$ y w es el tamaño de bits suficiente para representar k.
- Por ejemplo si $M=2^{14} = 16384$, $k = 123456$.

$$h(123456) = \lfloor 2^{14} (123456 \cdot 0.6180339887 \bmod 1) \rfloor = 67$$

Dispersión

- Otros métodos
  - **Hashing universal**
    - Podría ocurrir que si se tiene una función hash fija entonces un adversario podría elegir *n* claves para los que la dispersión sea siempre la misma, haciendo que el tiempo sea $\Theta(n)$.
    - **Idea**: elegir una nueva función hash en cada ejecución de forma aleatoria de un conjunto de funciones.
    - Ejemplo: *p* es un primo tal que $0 < k <= p$

$$h_{a,b}(k) = ((ak + b) \bmod p) \bmod m)$$

$$H_{p,m} = \{ h_{a,b} : a \in Z^*_p \text{ y } b \in Z_p \}$$

$$Z_p = \{0,1,..,p-1\}$$

$$Z^*_p = \{1,2,...,p-1\}$$

Ya que existen *p-1* posibilidades para ***a*** y *p* para ***b*** entonces hay *p(p-1)* funciones hash en $H_{p,m}$

Tablas de dispersión

Resolución de colisiones

- **Abierta (encadenamiento separado)**

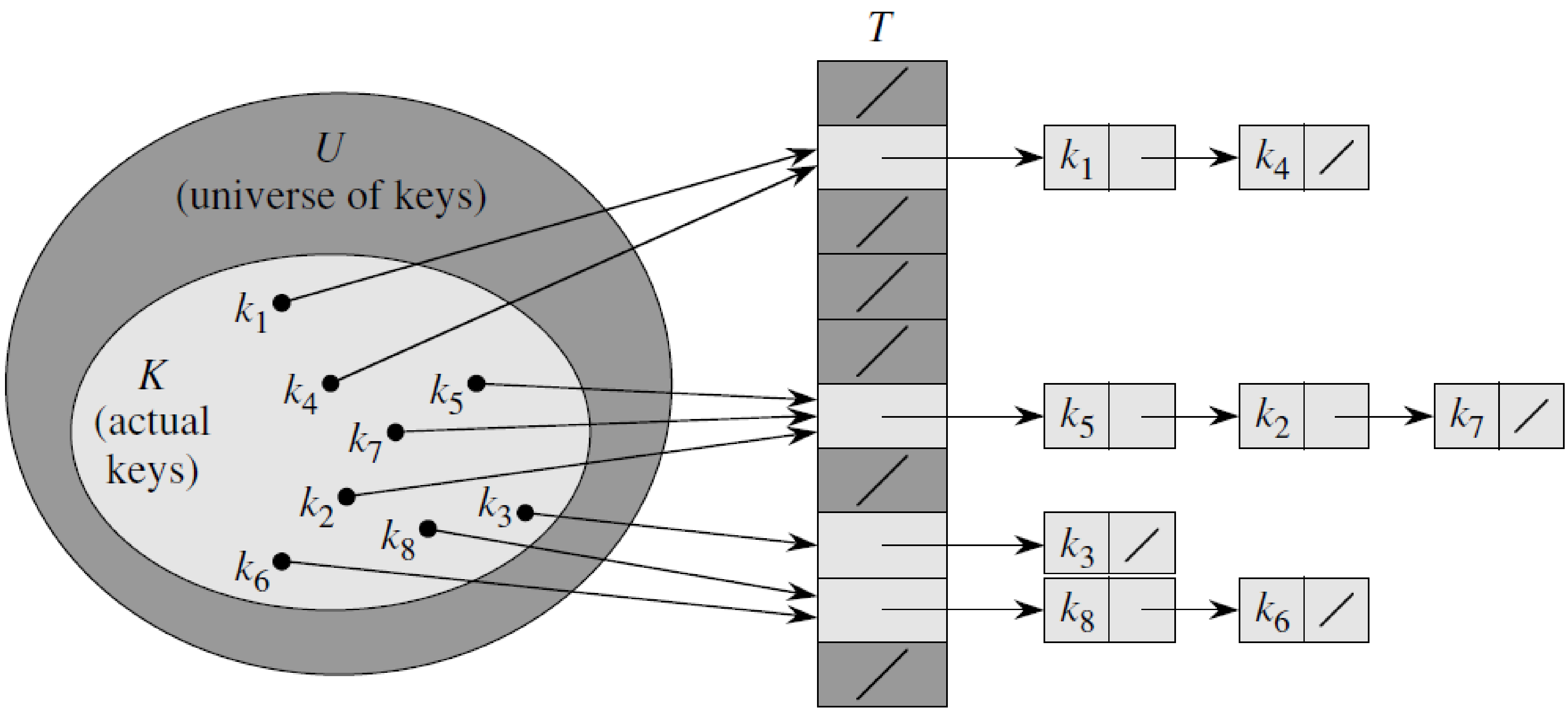
  Cada lugar de la tabla es asociada con una estructura dinámica como por ejemplo una lista enlazada

- **Cerrada**

  La resolución se hace en la propia tabla

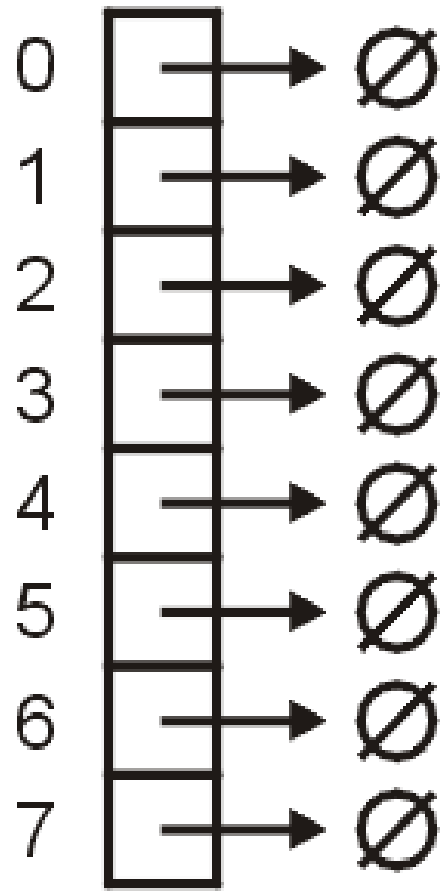
  - Exploración lineal
  - Exploración cuadrática
  - Doble hash

Tabla de dispersión
con resolución abierta



Tablas de dispersión

Resolución abierta (ejemplo)



Tablas de dispersión

Resolución abierta. Ejemplo

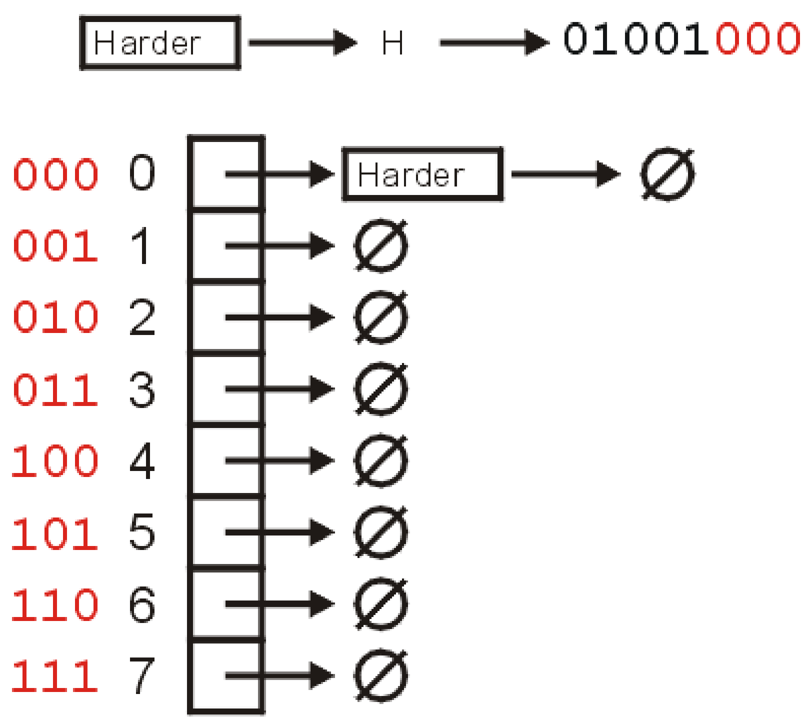
- Queremos guardar una palabra en una tabla HASH.
- La siguiente es una lista de la representación binaria de cada letra inicial

A	01000001	N	01001110
B	01000010	O	01001111
C	01000011	P	01010000
D	01000100	Q	01010001
E	01000101	R	01010010
F	01000110	S	01010011
G	01000111	T	01010100
H	01001000	U	01010101
I	01001001	V	01010110
J	01001010	W	01010111
K	01001011	X	01011000
L	01001100	Y	01011001
M	01001101	Z	01011010

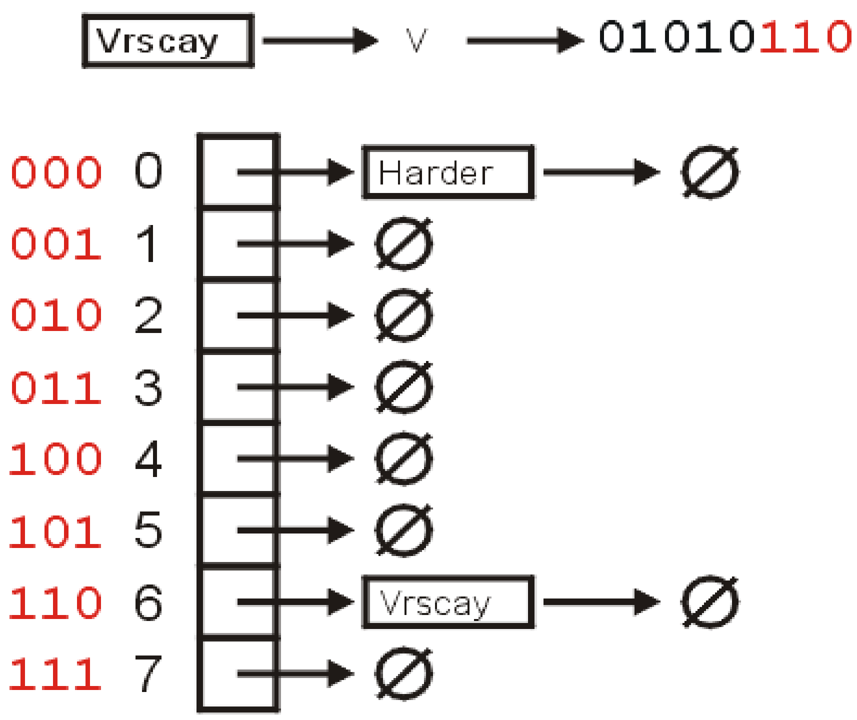
Tablas de dispersión

Resolución Abierta. Los últimos tres bits puede leerse entre 0 y 7.

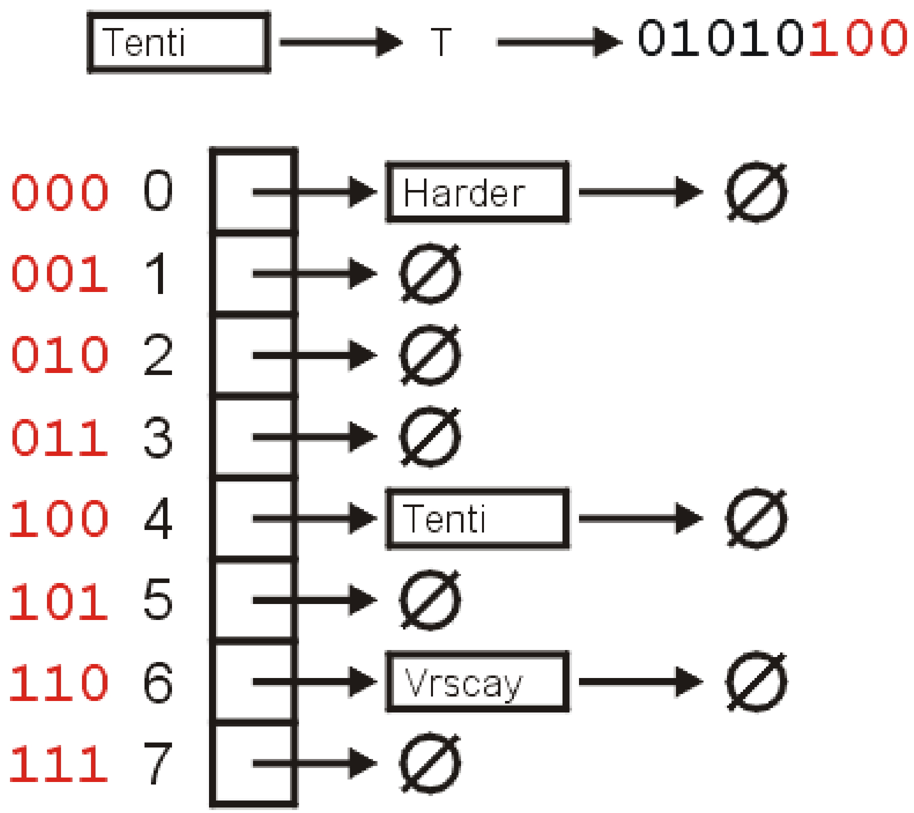
- Entonces consideramos la tabla de 0 a 7
- Insertamos “Harder” en la tabla



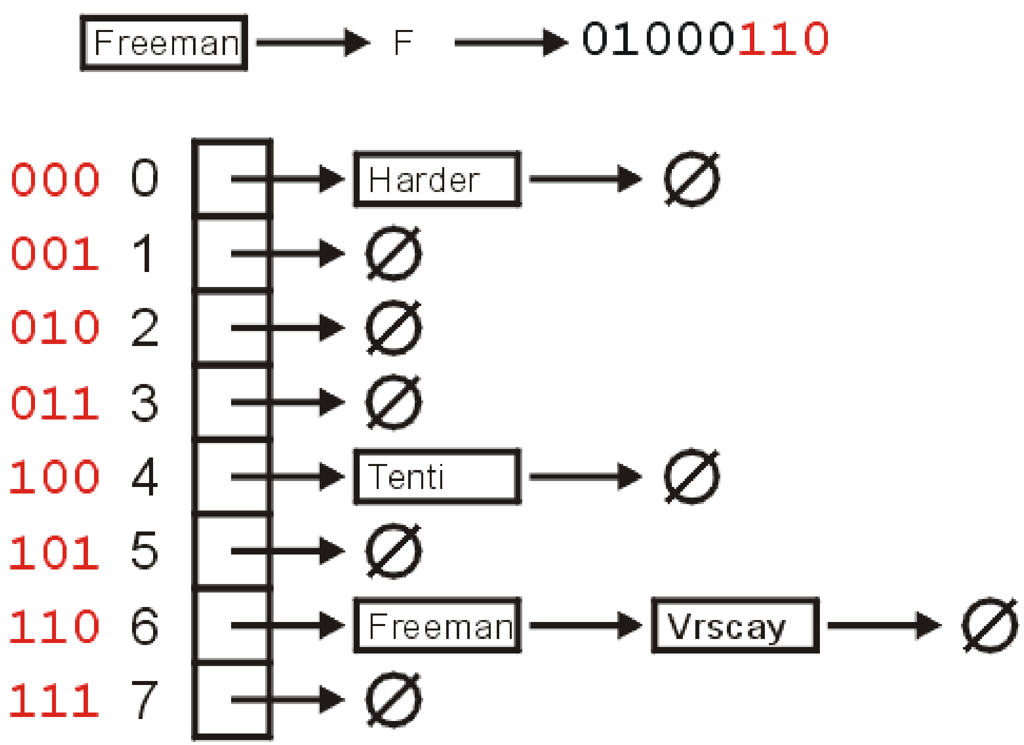
- Insertamos `"Vrscay"`



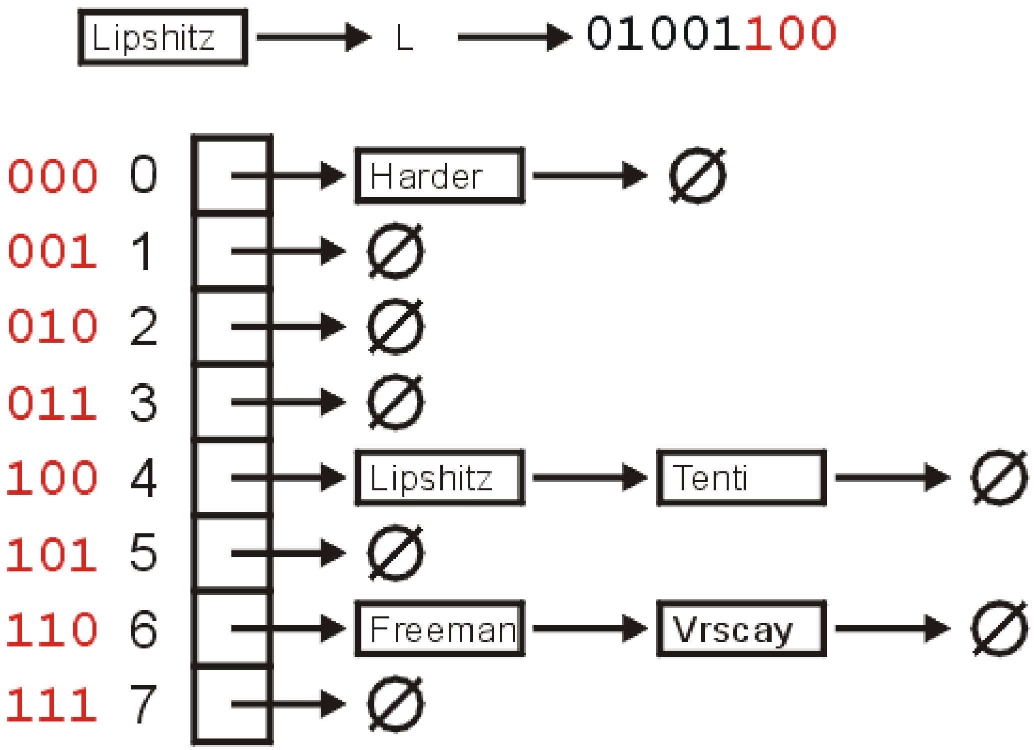
- Insertamos "`Tenti`"



- Insertamos "`Freeman`"



- Insertamos “`Lipshitz`”



- Insertamos "`Bishop`"

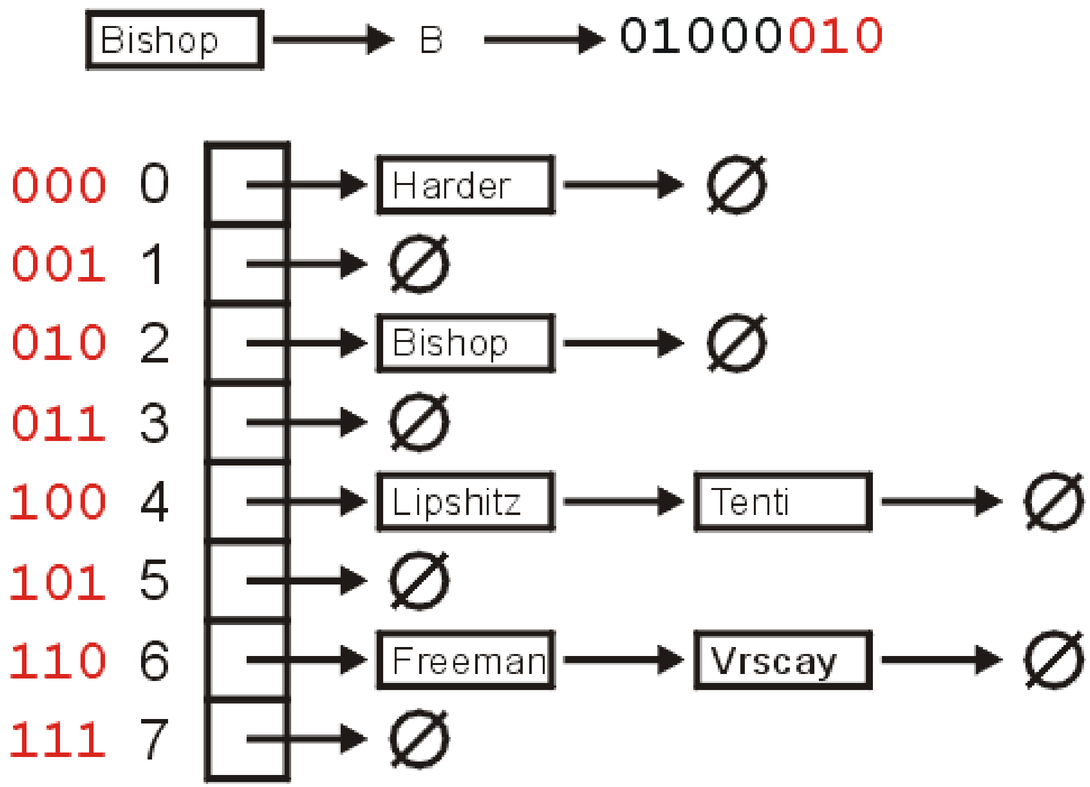
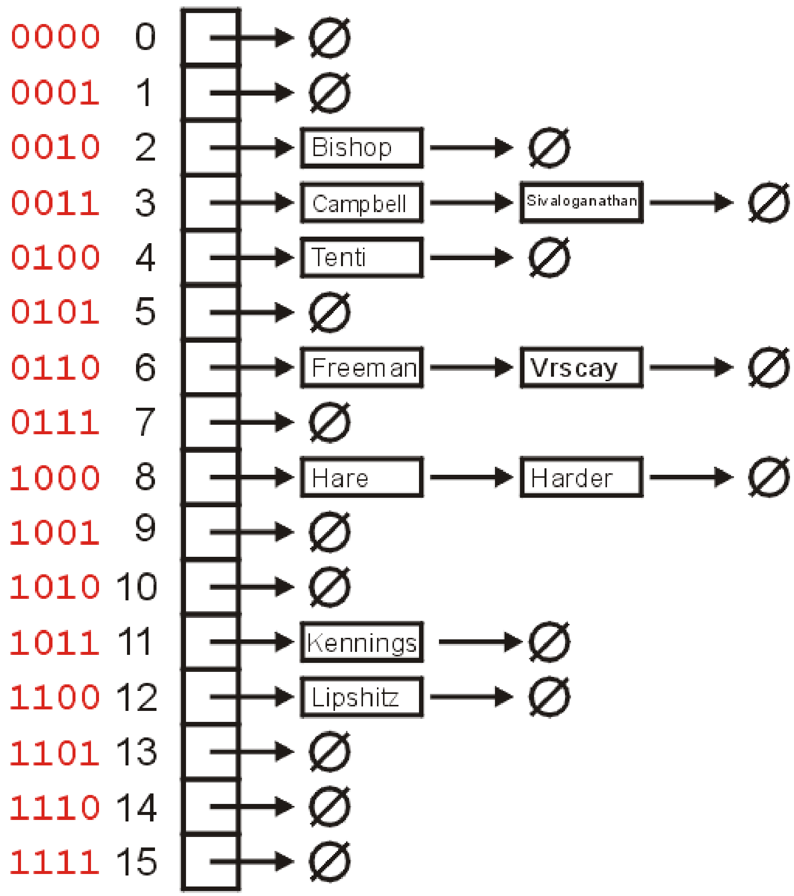


Tabla de dispersión

Factor de carga

- Se define como $\lambda = n/M$ (n = número de elementos, M tamaño de la tabla).
- En el caso de una tabla de dispersión abierta el factor de carga es la cantidad promedio de nodos por lista enlazada o estructura auxiliar.
- Tiempo promedio de búsqueda en una lista es λ y el tiempo requerido para una búsqueda no exitosa es $\Theta(1+\lambda)$. Una búsqueda exitosa también es $\Theta(1+\lambda)$
- Si el factor de carga es grande,
  - ¿Que significa?
  - ¿Que le parece que se puede hacer?

Redispersión del ejemplo



B	01000010
C	01000011
F	01000110
H	01001000
K	01001011
L	01001100
S	01010011
T	01010100
V	01010110

Tablas de dispersión

Resolución abierta

- Requiere memoria extra
- Requiere tiempo para asignar/liberar objetos de memoria.

Tablas de dispersión

Resolución Cerrada: todas las claves se guardan en la misma tabla hash, sin estructuras auxiliares

- Exploración lineal
  - $p(K,i) = i + h(K)$
- Exploración cuadrática
  - $p(K,i) = i^2 + h(K)$
- Doble Hashing
  - Basada en la exploración lineal pero utilizando como desplazamiento inicial una segunda función de dispersión. De esta forma se prueba la secuencia

    $p(K,i) = i * h_2(K)$

Tabla de dispersión

Ejemplo de exploración lineal

- Insertar los siguientes números en una tabla de dispersión de exploración lineal:

81,70,97,60,51,38,89,68,24

0	1	2	3	4	5	6	7	8	9

Exploración lineal, inserción

- Es fácil insertar el 81, 70 y 97 en su lugar correspondiente

0	1	2	3	4	5	6	7	8	9
70	81						97		

Exploración lineal, inserción

- La inserción del 60 causa una colisión en la cubeta 0 y 1
  - Cubeta 0 (llena)
  - Cubeta 1 (llena)
  - Cubeta 2 (disponible)

0	1	2	3	4	5	6	7	8	9
70	81	60					97		

Exploración lineal, inserción

- Insertar el 51 también causa colisión

0	1	2	3	4	5	6	7	8	9
70	81	60	51				97		

- 38 y 39 sin colisiones

0	1	2	3	4	5	6	7	8	9
70	81	60	51				97	38	89

Exploración lineal, inserción

- Al insertar 68 tenemos colisión en la cubeta 8 y chequeamos las cubetas
  - 9, 0, 1, 2, 3 y encontramos lugar en el 4.

0	1	2	3	4	5	6	7	8	9
70	81	60	51	68			97	38	89

- Finalmente insertamos el 24

0	1	2	3	4	5	6	7	8	9
70	81	60	51	68	24		97	38	89

Exploración lineal : búsqueda

- Similar a la inserción
- Iniciamos en la cubeta apropiada y continuamos la búsqueda hasta que:
  - Encontramos el ítem buscado
  - Encontramos una cubeta vacía
  - Hemos atravesado el arreglo completo
- Lo último ocurre solo cuando la tabla esta llena

Exploración lineal: búsqueda

- Para buscar el *68*, examinamos la cubeta 8, 9, 0, 1, 2, 3 y encontramos en 4.
- Para buscar el *23*, examinamos la cubeta 3, 4, 5 y la cubeta 6 esta vacía, así el 23 no esta en la tabla.

0	1	2	3	4	5	6	7	8	9
70	81	60	51	68	24		97	38	89

Exploración lineal: eliminación

- No se puede simplemente remover el elemento de la tabla de dispersión
- Ejemplo: si removemos el 89, luego no podríamos encontrar el 68, ¿porqué?

0	1	2	3	4	5	6	7	8	9
70	81	60	51	68	24		97	38	89

Exploración lineal: eliminación

- ¿Cómo solucionar?
  - Mover los elementos que no han sido movidos antes de su cubeta inicial
  - Por ejemplo, si removemos el 89

0	1	2	3	4	5	6	7	8	9
70	81	60	51	68	24		97	38	89

Exploración lineal: eliminación

- Probamos hacia adelante hasta encontrar una entrada que puede ser movida a la cubeta 9.
- No podemos mover el 70, 81, 60, 51 pero si el 68.

0	1	2	3	4	5	6	7	8	9
70	81	60	51		24		97	38	68

Exploración lineal : eliminación

- A continuación, volvemos a buscar hacia adelante y notamos que el 24 puede ser movido hacia atrás.
- La siguiente celda esta vacía por tanto hemos finalizado.

0	1	2	3	4	5	6	7	8	9
70	81	60	51	24			97	38	68

¿Cómo eliminamos el 60?

Exploración lineal: inconveniente

- Observar el siguiente fenómeno:
  - A medida que aumentamos la cantidad de elementos en la tabla de dispersión, las regiones contiguas(o *cluster*) son mayores.
  - Esto resulta en mayor tiempo de búsqueda.
  - Este hecho se denomina *agrupamiento primario*

Exploración lineal:
Agrupamiento primario

- A medida que el factor de carga aumenta, la probabilidad de colisión también aumenta
- Justificación:
  - Suponga una región (o cadena) de longitud *m*
  - Una inserción puede ocurrir en cualquier región ocupada, o inmediatamente al principio o al final lo que causará el incremento de la región.

...	28	29	30	31	32	33	34	35	...
			230	531	730	432			

Exploración lineal:
agrupamiento primario

- Consecuentemente, si la cadena es de tamaño m, entonces la probabilidad de que su longitud se incremente es (m+2)/M, donde M es el tamaño de la tabla.
- Esto significa que la longitud de estas regiones puede afectar al rendimiento.

Exploración lineal: rendimiento

Es posible estimar el número promedio de pruebas para una búsqueda exitosa, donde λ es el factor de carga:

$$\tfrac{1}{2}\left(1+\frac{1}{1-\lambda}\right)$$

Por ejemplo si $\lambda = 0.5\ (50\%)$ entonces tenemos 1.5 pruebas

Exploración lineal

- El número de pruebas para una búsqueda no exitosa o para una inserción es mayor:

$$\tfrac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right)$$

- Para $\lambda = 0.5$ entonces se requiere 2.5 pruebas

λ	**Búsqueda exitosa**	**Inserción o búsqueda no exitosa**
50%	1.5	2.5
75%	2.5	8.5
90%	5.5	50.5

Exploración lineal

- Nuestro objetivo es tener un tiempo O(1)
- Sin embargo dependemos del factor de carga, cuando este crece, también lo hace el tiempo de respuesta.
- Una solución es mantener este factor en un límite
- Si mantenemos a 2/3 (66%) entonces el número de pruebas para una búsqueda exitosa y no exitosa será de 2 y 5 respectivamente.

Exploración lineal

- Por tanto tenemos las siguientes opciones:
  - Cambiar a una M suficientemente grande que no sobrepase el factor de carga
  - Doblar el número de cubetas (o el tamaño de M) cuando llegamos al factor de carga límite
  - Buscar otra estrategia para evitar la agrupación primaria (*exploración cuadrática* o *doble hashing*)

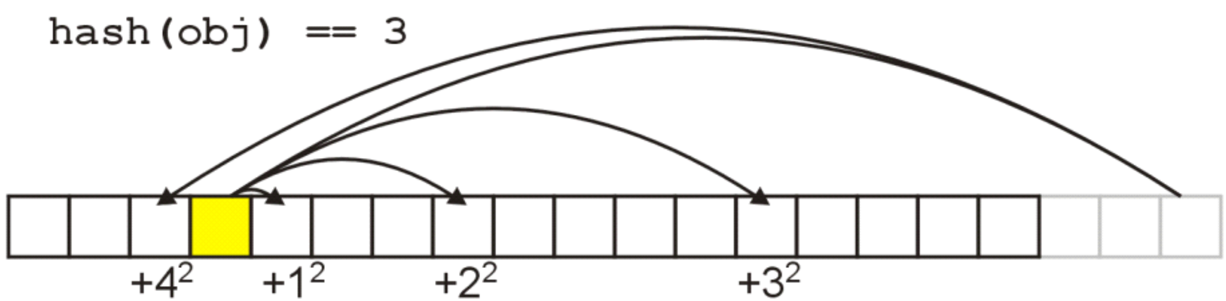
Exploración cuadrática

- Intento de solucionar la agrupación primaria
- En vez de usar una función linear usamos una función cuadrática.
  - Suponga que un elemento debe estar en la cubeta h:
    - Si la misma esta ocupada, chequeamos en la siguiente secuencia

$$h+1^2,\ \ h+2^2,\ \ h+3^2,\ \ h+4^2,\ \ \ h+5^2,\ \ldots$$

$$h+1,\ \ \ h+4,\ \ \ h+9,\ \ \ \ h+16,\ \ h+25,\ \ldots$$

Por ejemplo, con M = 17



Exploración cuadrática

- Resuelve el problema de agrupación primaria
- Existe un problema, suponga que M = 8
  - 1^2 => 1 , 2^2=> 4, 3^2 => 1
- En este caso, hemos chequeado la misma cubeta..
- No hay garantía de que con $h+i^2 \; mod \; M$ tengamos un ciclo *0,1..M-1*
- *Solución: hacer que M sea primo*

Exploración cuadrática
eliminación

- Hemos visto en la exploración lineal un esquema de borrado, y fue relativamente fácil
- Sin embargo en el esquema cuadrático no es fácil hacer esto eficientemente.
- ***Solución***: asociar cada cubeta con un campo que indique : VACIO, OCUPADO, BORRADO

Exploración cuadrática

- Si la función de dispersión no es buena, y tenemos elementos en la misma cubeta, la misma secuencia será utilizada.
- Este es un fenómeno denominado agrupación secundaria (secondary clustering)
- El efecto es menos significativo que la agrupación primaria.
- Soluciones:
  - Aumentar M al siguiente número primo cercano al doble.
  - Uso de una segunda función de dispersión: *doble hashing*

Doble dispersión

- Basado en la exploración lineal pero utilizando como desplazamiento inicial una segunda función de dispersión.

  Se define de esta forma:

  $$h(K,i) = (h_1(K) + i * h_2(K)) \% M \quad para\ i=0,1,2,..$$

  ***Ejemplo**: para una tabla de M=101 y tres valores k1,k2,k3 con $h_1(k1)=30$, $h_1(k2)=28$ y $h_1(k3)=30$ y $h_2(k1)=2$, $h_2(k2)=5$ y $h_2(k3)=5$.*

  La secuencia de prueba para k1 seria 30, 32, 34, 36.

  ¿Cuál sería para k2 y k3?

  K2 = 28, 33, 38, 43
  K3 = 30, 35, 40, 45

Tablas de dispersión - Ejercicio

- Probar exploración lineal y cuadrática con los siguientes valores.
  - Insertar 6 elementos 3, 107, 9, 119, 35, 112 en una tabla de M = 11.
  - $h(k) = k \bmod 11$
- Probar exploración con doble hashing:
  - Secuencia de claves: 79,69,98,72,14,50.
  - $h_1(k) = k \bmod 13$ y $h_2(k) = 1 + (k \bmod 11)$.

Hashing perfecto

- Puede utilizarse para conseguir una performance ideal (O(1) cuando el conjunto de datos es estático.
  - Ejemplos:
    - Conjunto de palabras reservadas en un lenguaje de programación
    - Nombres de archivo en un DVD-ROM
- Es de dispersión abierta pero en vez de utilizar una lista utiliza otra tabla de dispersión.
- Utiliza la dispersión con hashing universal para ambos niveles de hash.

Hashing perfecto

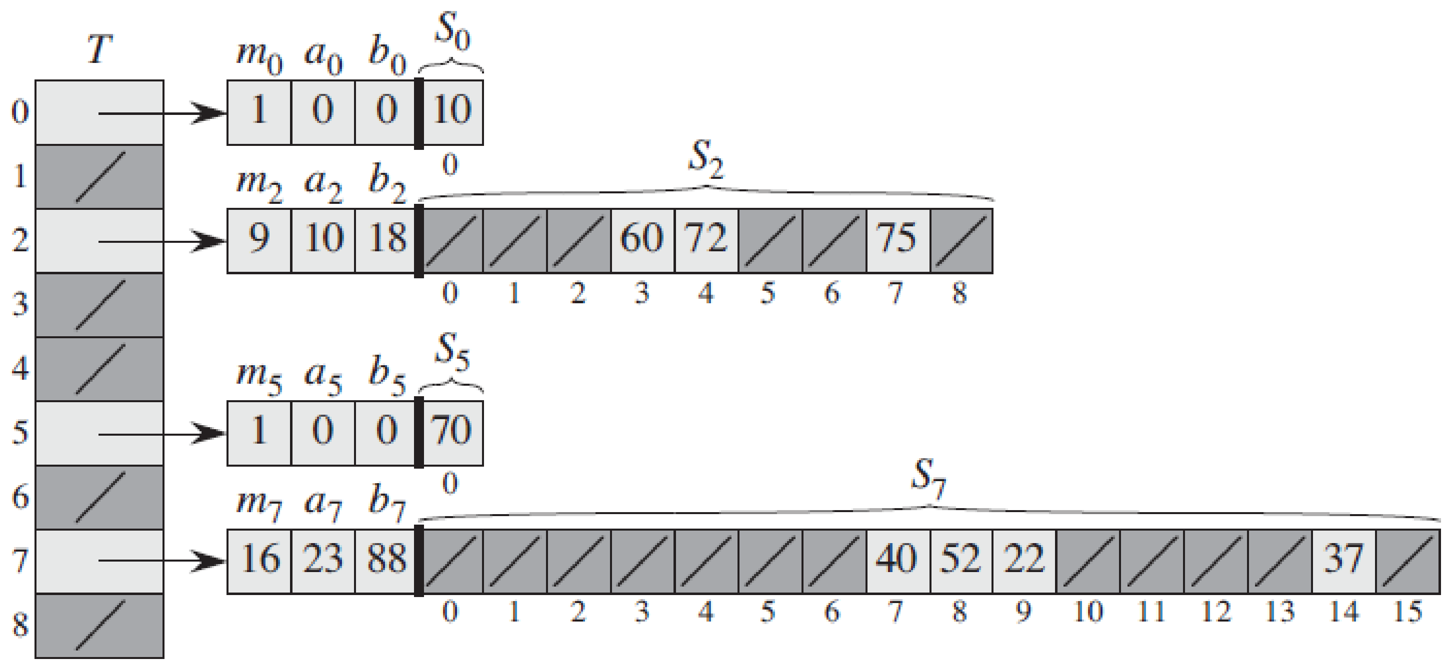


Figure 11.6 Using perfect hashing to store the set $K = \{10, 22, 37, 40, 52, 60, 70, 72, 75\}$. The outer hash function is $h(k) = ((ak + b) \bmod p) \bmod m$, where $a = 3$, $b = 42$, $p = 101$, and $m = 9$. For example, $h(75) = 2$, and so key 75 hashes to slot 2 of table T. A secondary hash table S_j stores all keys hashing to slot j. The size of hash table S_j is $m_j = n_j^2$, and the associated hash function is $h_j(k) = ((a_j k + b_j) \bmod p) \bmod m_j$. Since $h_2(75) = 7$, key 75 is stored in slot 7 of secondary hash table S_2. No collisions occur in any of the secondary hash tables, and so searching takes constant time in the worst case.

Bibliografía

- Mark A. Weiss. *Estructura de datos en Java. Compatible con Java 2*. Pearson. AW. 2000.
- M. Goodrich and R. Tamassia. Data Structures & Algorithms in Java. Fourth Edition. J. Wiley. 2005.
- Levitin, Anany. *Introduction to the Design and Analysis of Algorithms*. 2nd Edition. Pearson. AW. 2007.
- Cormen T., Leiserson C., Rivest R and Stein C. Introduction to algorithms. 2nd. Edition. 2001. Capitulo 11 (Hash Tables).
- Cormen T., Leiserson C., Rivest R and Stein C. Introduction to algorithms. 3nd. Edition. 2009. Capitulo 11 (Hash Tables).