Universidad Nacional de Ingeniería
Ciencias de la Computación
Hashtables

Yuri Nuñez Medrano *
ynunezm@gmail.com

Resumen

Las tablas hash se basan en una tabla T, donde cada registro sea rápidamente ubicable.

1. Direct Address Table

Se asume que tenemos un Universo de keys razonablemente pequeño y se tiene un universo $U = \{0, 1, ..., m-1\}$, donde m no es grande.
Se asume que los key son distintos.
Se tiene una tabla $T[0, ..., m-1]$.
$T[k] = \left\{ \begin{array}{l} x \in k(keys) \wedge x.key = k \\ nil\ en\ otro\ caso \end{array} \right\}$ Con las siguiente operaciones.

Algorithm 1: DIRECT_ADDRESS_SEARCH(T,k)

```
return T[k]
```

Algorithm 2: DIRECT_ADDRESS_INSERT(T,x)

```
T[x.key]=x
```

Algorithm 3: DIRECT_ADDRESS_DELETE(T,x)

```
T[x.key]=nil
```

Direct Address tiene inconvenientes, si el Universo U es grande, el almacenamiento de una tabla T de tamaño $|U|$ puede ser impracticable.

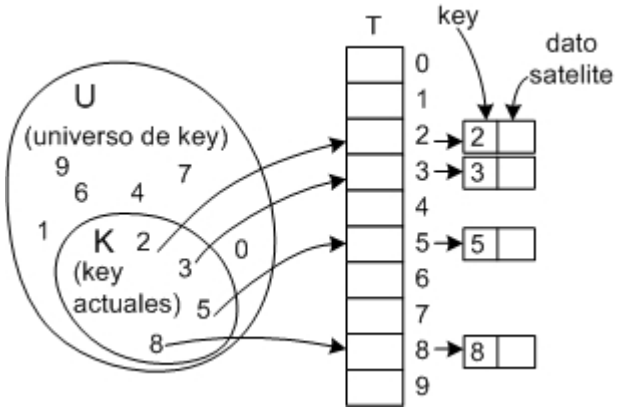

Figura 1: Direct Address

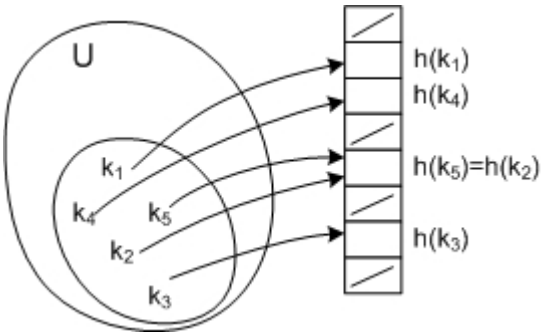

Figura 2: Tabla Hash

2. Función Hash

La función hash mapea los keys "aleatoriamente " en los registros de la tabla T.
h mapea el universo U de keys en los registros de la tabla hash $T[0, 1, ..., m-1]$.
$h : U \to \{0, 1, ..., m-1\}$
Donde m es menor que $|U|$
La función hash reduce el rango del indice del tamaño del array en vez del tamaño $|U|$, el array tiene tamaño m en la figura 2.

*Escuela de Ciencias de la Computación, 27-08-15

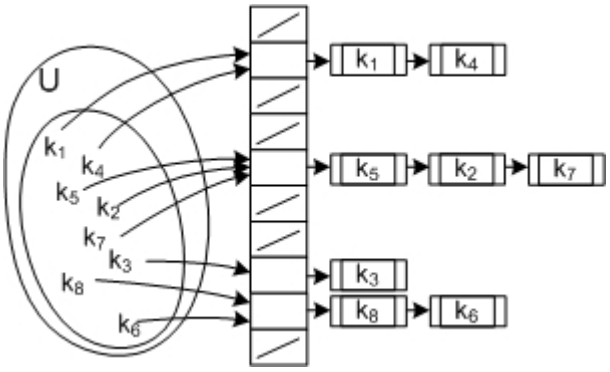

Figura 3: Chaining Hash

3. Chaining Hash

Cuando se inserta en un registro ya ocupado ocurre una colisión. Para la resolución de colisiones mediante chaining, enlazar el registro en una lista como en la figura 3

con las siguientes colisiones en listas
$h(k_1) = h(k_4)$
$h(k_5) = h(k_2) = h(k_7)$
$h(k_8) = h(k_6)$

Con las siguiente operaciones.

Algorithm 4: CHAINED_ADDRESS_SEARCH(T,k)

1 buscar un elemento con key k en la lista T[h(k)]

Algorithm 5: CHAINED_ADDRESS_INSERT(T,x)

1 insertar x al inicio de la lista T[h(x.key)]

Analisis
Caso peor, si todos los keys esta en el mismo registro, el acceso toma $\Theta(n)$ si $|K| = n$.
Caso promedio, suponemos de una simple función hashing uniforme.
Donde cada $key \in K$ ES equivalente similar para la función hash, para ser distribuida en cualquier registro en T, independientemente de otros key con su función has $h(k)$.
El load facto (α) en una tabla hash con n keys y m registros es $\alpha = n/m$ es el numero de key por registro.
El tiempo esperado de busqueda infructuosa.
$= \Theta(1 + \alpha)$.
El tiempo esperado es $= \Theta(1)$, si $\alpha = O(1)$

Algorithm 6: CHAINED_ADDRESS_DELETE(T,x)

1 eliminar x de la lista T[h(x.key)]

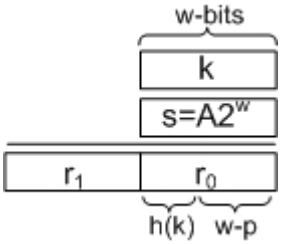

Figura 4: Método de la multiplicación

4. Escogiendo una Función Hash

Debería distribuir los keys uniformemente en los registros.
La distribución de los keys no debería afectar la uniformidad.

4.1. Método de la División

$h(k) = k \ mod \ m$.
Ejm, si m es par y todos los key son pares los registros impares no se usan.
Ejm, si $m = 2^r$.
Escoger un m tan cercano a una potencia de 2 o de 10.

4.2. Método de la multiplicación

Multiplicamos k (keys) por una constante A en el rango $0 < A < 1$ y $m = 2^p$. en la figura 4
la representación de los valores w-bits del key k.
$s = A,2^k$ Ejm, k=89 A=0.8359375.
$s = A,2^w = 106{,}88 \cong 107$
$m = 8$
$p = 3$
$h(k) = \lfloor m(k.A \ mod \ 1) \rfloor$

$$h(k) = \lfloor \overbrace{m(\underbrace{k.A}_{74,398} \ mod \ 1)}^{3,1875} \rfloor \quad \underbrace{\qquad}_{0,398}$$

Ejm, $k = 123456$, $p = 14$, $m = 2^{14}$
$A = 2654435769/2^{32}$.

$$h(k) = \lfloor \overbrace{m(\underbrace{k.A}_{76300,0041} \ mod \ 1)}^{67,1877441} \rfloor \quad \underbrace{\qquad}_{0,0041}$$

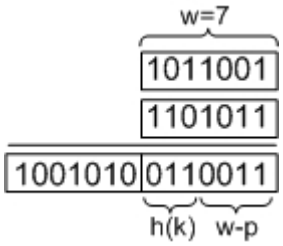

Figura 5: Solución del ejemplo

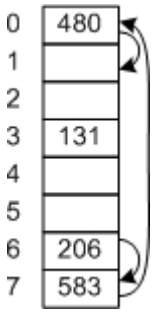

Figura 6: Linear Probing

5. Open Adressing

Resolviendo coliciones por open addressing, evaluando una llamada sistemática a la tabla hasta encontrar un registro vacio.

$h : \underbrace{U}_{Universo\ de\ keys} \quad \underbrace{\{0,1,...,m-1\}}_{prueba\ el\ nro} \rightarrow \underbrace{\{0,1,...m-1\}}_{en\ un\ registro}$

probar la secuencia como una permutación.

$\langle h(k,0), h(k,1), ..., h(k,m-1)\rangle$

Algorithm 7: HASH_INSERT(T,k)

```
1 i=0
2 repeat
3 |   j= h(k,i)
4 |   if T[j]=NIL then
5 |   |   T[j]=k
6 |   |   return j
7 |   else i=i+1;
8 until i==m;
9 error "hash table overflow"
```

Algorithm 8: HASH_SEARCH(T,k)

```
1 i=0
2 repeat
3 |   j= h(k,i)
4 |   if T[j]=k then
5 |   |   return j
6 |   i=i+1
7 until T[j]==NIL or i==m;
8 return NIL
```

5.1. Linear Probing

h(k,i)=(h'(k)+i)mod m
donde h' es una funcion hash auxiliar , $i = 0, 1, .., m-1$
$m = 8$ si $h'(k) = k\ mod\ m$
$h(k,i) = (h'(k) + i) mod\ m$

$k = 583$
$h(583, 0) = (h'(583) + 0) mod\ 8 = 7$
$k = 206$
$h(206, 0) = (h'(206) + 0) mod\ 8 = 6$
$k = 480$
$h(480, 0) = (h'(480) + 0) mod\ 8 = 0$
$k = 131$
$h(131, 0) = (h'(131) + 0) mod\ 8 = 3$
$k = 422$
$h(422, 0) = (h'(422) + 0) mod\ 8 = 6$
$h(422, 1) = (h'(422) + 1) mod\ 8 = 7$
$h(422, 2) = (h'(422) + 2) mod\ 8 = 0$
$h(422, 3) = (h'(422) + 3) mod\ 8 = 1$

5.2. Quadratic Probing

$h(k,i) = (h'(k) + c_1 i + c_2 i^2) mod\ m$
donde h' es una funcion hash auxiliar $i = 0, 1, .., m-1$, c_1 y c_2 es una constante auxiliares positivas.

5.3. Doble hashing

$h(k,i) = (h_1(k) + i h_2(k)) mod\ m$
donde h_1 y h_2 son funciones hash auxiliares $i = 0, 1, .., m-1$
$m = 13$ $m' = 11$
si $h_1(k) = k\ mod\ m$
$h_2(k) = 1 + (k\ mod\ m')$
$h(k,i) = (h_1(k) + i h_2(k)) mod\ m$

$k = 14$
$h(14, 0) = (h_1(14) + 0 h_2(14)) mod\ 13 = 1$
$h(14, 1) = (h_1(14) + 1 h_2(14)) mod\ 13 = 5$
$h(14, 2) = (h_1(14) + 2 h_2(14)) mod\ 13 = 9$

5.4. Perfect hashing

Es una buena eleccion para un buen desempeño de un caso promedio y para el caso peor, cuando el conjunto de key es

static, una vez que los key se guardan y no tienen cambios.
$h(k) = ((ak + b) mod\ p) mod\ m$
donde a , b y p son variables constantes.
las iesimas tablas tienen la siguiente formula
$h_j(k) = ((a_j k + b_j) mod\ p) mod\ m$

Ejm: Evaluar los siguientes keys

$K = 10, 22, 37, 40, 52, 60, 70, 72, 75$

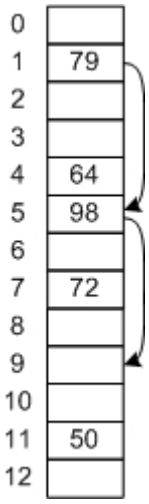

Figura 7: Double Hash

Referencias

[H.Cormen et al., 2009] H.Cormen, T., Leiserson, C., and Riverson, R. L. (2009). *Algorithms*. The MIT Press.

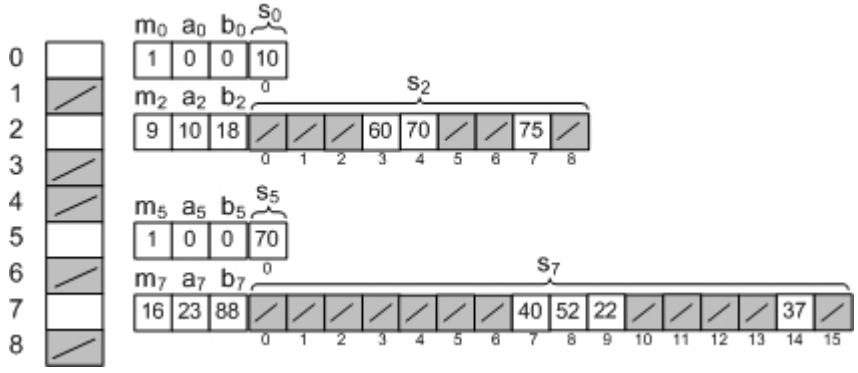

Figura 8: Perfect Hash