Tablas HASH

Agustín J. González

ELO320: Estructura de Datos y Algoritmos

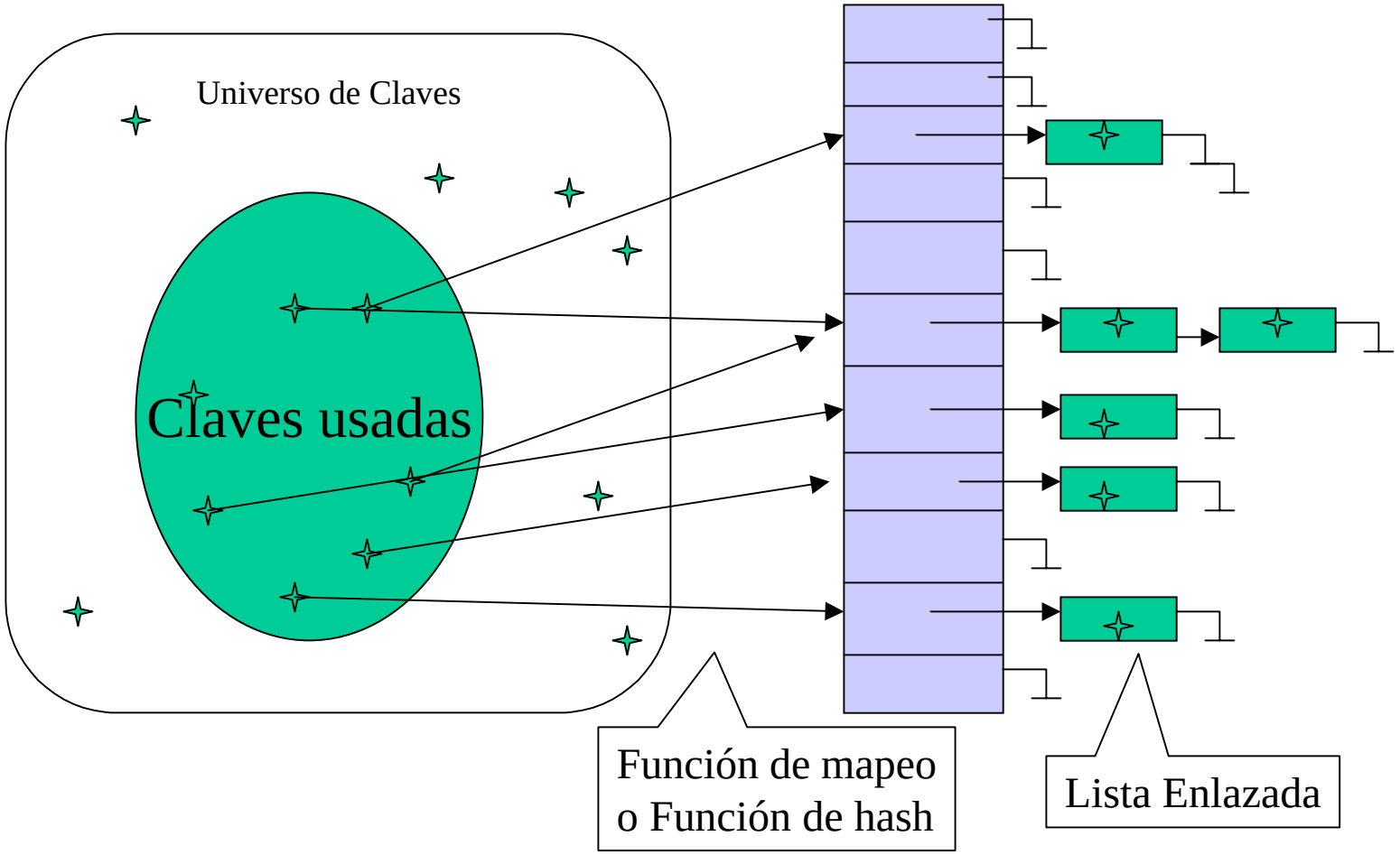
1er. Sem. 2002

Introducción

- Muchas aplicaciones requieren un conjunto dinámico que soporte las operaciones de un diccionario: Insert, Search, Delete. Por ejemplo el compilador cuando guarda los identificadores de un programa.
- Es posible hacer uso de una lista enlazada con un tiempo O(n) ; sin embargo, este tiempo se puede reducir notablemente a orden O(1) en la mayoría de los casos usando una tabla hash.
- La idea surge de los arreglos que nos permiten acceso a sus elementos en orden O(1).
- Una opción sería usar un arreglo tan grande como el rango de posibles claves. La desventaja es el espacio de memoria requerido en tal estrategia.
- Otra opción es usar un arreglo menor, al cual podemos mapear las claves en uso. Esta función de mapeo es la ***función hash***. La tabla así organizada es la ***tabla hash***.
- Como es posible que dos claves conduzcan al mismo mapeo (lo cual se conoce como una **colisión**), es necesario buscar formas para resolver esta situación.
- Una forma, conocida como hashing abierto (hay otros términos dependiendo del texto), crear una lista asociada a cada entrada del arreglo.
- Otra forma, conocida como hashing cerrado (el término depende del libro), almacena las claves en las mismas entradas del arreglo o tabla hash.

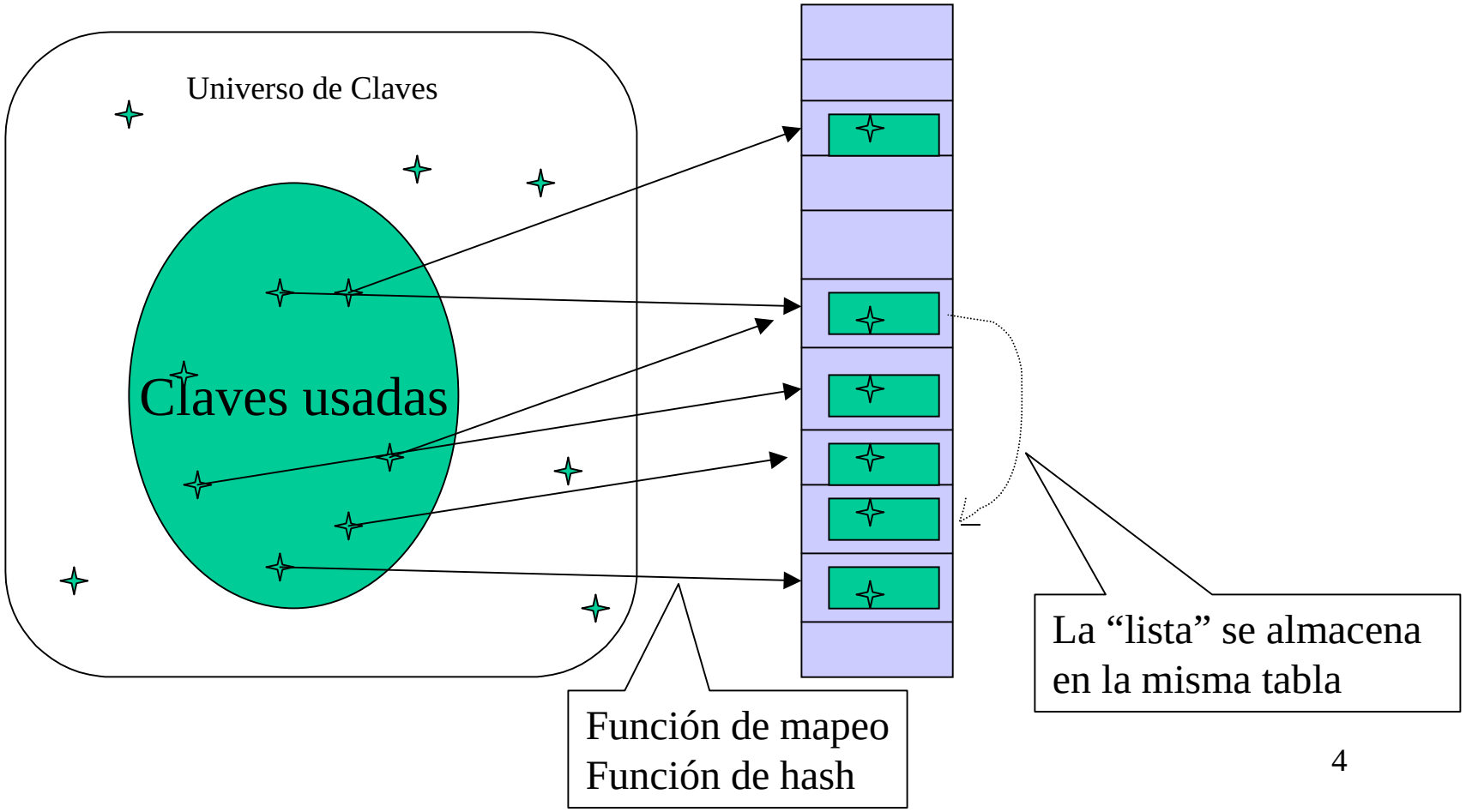
Visión gráfica (hashing abierto)

- Desde un “gran” Universo sólo un número reducido de claves serán consideradas.



Visión gráfica (hashing cerrado)

- Desde un “gran” Universo sólo un número reducido de claves serán consideradas.



Hashing Abierto

- Suposición de **hashing uniforme**: es cuando cualquier elemento es igualmente probable de caer en cualquiera de las m entradas de la tabla hash, independientemente de cualquier otro elemento.
- Aún con hashing uniforme, el peor caso de hashing abierto nos conduce a una lista con todas las claves en una única lista. El peor caso para búsqueda es así $\Theta(n)$.
- En hashing abierto la búsqueda no exitosa de una clave toma tiempo $\Theta(1+\alpha)$, donde α es el factor de carga =número de claves en la tabla/número de entradas en la tabla hash.
  ¿Por qué esto? El costo de calcular la función hash $\Theta(1)$, más la prueba en cada una de los nodos de la lista asociada a la entrada. En promedio hay n/m nodos en cada lista y hay que probarlos todos 0 ==> $\Theta(\alpha)$. Luego se tiene que el tiempo total es $\Theta(1+\alpha)$.
- Análogamente la búsqueda exitosa de una clave toma un tiempo $\Theta(1+\alpha/2)$
- La inserción de una clave toma $\Theta(1)$
- La eliminación de una clave toma un tiempo $\Theta(1+\alpha)$. Aquí suponemos que la clave debe ser buscada dentro de la lista, para luego ser eliminada.
- En resumen, si la tabla mantiene un limitado número de claves, n/m está acotado por una constante, todas las operaciones toman un tiempo $\Theta(1)$.

Funciones Hash

- Una buena función hash debería satisfacer la suposición de hash uniforme.
- Como el recorrido de la función de hash es un número natural, hay que saber interpretar o transformar a número natural tipo de clave.
- Si se trata de claves enteras, el problema está más o menos resuelto.
- Si se trata de secuencia de caracteres, strings, se puede interpretar cada carácter como un número en base 128 (los números ASCII van del 0 al 127) y el string completo como un número en base 128. Así por ejemplo la clave pt puede ser transformada a (112*128+116)=14452. OBS: ASCII(p)=112 y ASCII(t)=116.
- En adelante supondremos que las claves son números naturales (o ya han sido transformadas a números naturales)

Funciones Hash: Método de División

- **Método de división:**
- Este método consiste en tomar el resto de la división por m, el número de entradas de la tabla. Así
  h(k) = k mod m
  En C sería h(k) = k % m;
- Usar m = una potencia de 2 no es buena idea ya que el valor de hash queda dependiendo de sólo los bits menos significativos de k.
- Una forma de hacer hash(k) dependiente de todos los bits menos significativos es usar número primos no muy cercanos a una potencia de dos.

Funciones Hash: Método de Multiplicación

- **Método de multiplicación**
- Este método opera en dos pasos. Primero, multiplicamos la clave por una constante A en el rango 0 < A < 1 y extraemos la parte fraccionaria de k*A. Segundo, Multiplicamos este valor por el número de entradas de la tabla y tomanos el piso del (o truncamos el) resultado.
- En resumen:
  $h(k) = \lfloor m * (k*A \bmod 1) \rfloor$
  Donde mod 1 debe ser interpretado como $k*A - \lfloor k*A \rfloor$
- ¿Cómo se hace en C? Ver man modf. También es útil man -k fraction
  #include <math.h>
  double modf(double x, double *iptr);
  Description: The modf() function breaks the argument x into an integral part and a fractional part, each of which has the same sign as x. The integral part is stored in iptr. The modf() function returns the fractional part of x.
- Una ventaja de este método es que el valor de m no es crítico.
- El método trabaja bien con cualquier valor de A, pero trabaja mejor con algunos valores que otros, por ejemplo A~(sqrt(5)-1)/2 es recomendado.
  Así para m =10000, $h(123456) = \lfloor 10000 * (123456*0.61803.. \bmod 1) \rfloor = 41$

Hashing Cerrado

- En Hashing cerrado todos los elementos o claves son almacenadas en la tabla hashing misma. Es decir cada entrada de la tabla contiene un elemento del conjunto dinámico o NULL.
- Cuando se busca, examinamos varias entradas hasta encontrarlo o es claro que no está.
- No hay una lista ni elementos almacenados fuera de la “tabla”.
- La tabla de podría llenar. El factor de carga no puede exceder 1.
- La gran ventaja de hashing cerrado es que elimina totalmente los punteros usados en la lista enlazada. Se libera así espacio de memoria, el que puede ser usado en más entradas de la tabla y menor número de colisiones.
- La inserción se efectúa probando la tabla hasta encontrar un espacio vacío. La función de hash usa un segundo argumento que el el número de la prueba.
  h: U x { 0, 1, 2, .. , m-1} ----> {0, 1, 2, ... m-1}
  Para una clave k se prueban sucesivamente: h(k,0) , h(k,1), .. h(k,m-1)
- Hash_Insert(T, k) { /* pseudo código */

```
int i,j;
for (i = 0; i<m; i++) {
        j=h(k,i);
        if (T[j] == NULL){
                T[j]=k;
                return;
        }
}
printf(“ hash overflow”);
}
```

```
Int Hash_Search(T, k) { /* Pseudo
código*/
int i,j;
for (i = 0; i<m; i++) {
        j=h(k,i);
        if (T[j] == NULL)
                return -1;
        else if  (T[j] == k)
                return j;
}
```

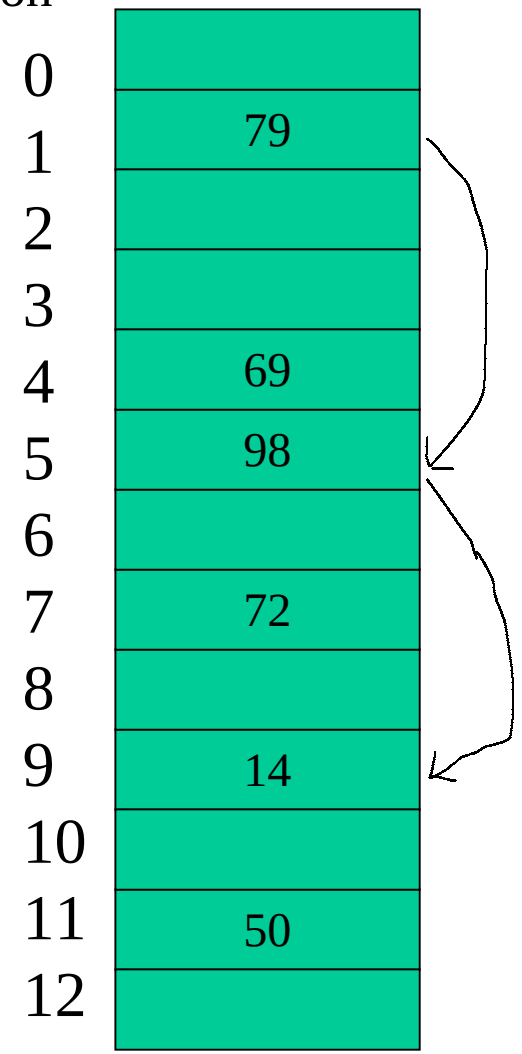
Funciones de Hash h(k,i)

- Existen al menos dos formas para definir esta función: prueba lineal y doble hashing.
- **Prueba lineal**
- La función es:
  h(k,i) = (h'(k) +i) mod m
- Una desventaja de este método es la tendencia a crear largas secuencias de entradas ocupadas, incrementando el tiempo de inserción y búsqueda.
- **Doble hashing**
- La función es:
  $h(k,i) = (h_1(k) + i*h_2(k)) \bmod m$
- Por ejemplo:
  $h_1 = k \bmod m$
  $h_2 = 1 + (k \bmod (m-1))$

Ejemplo de hashing Cerrado

- Sea $h(k,i) = (h_1(k) + i*h_2(k)) \bmod 13$; con
  $h_1 = k \bmod 13$
  $h_2 = 1 + (k \bmod 11)$

- $h(79,0) = 1$
  $h(72,0) = 7$
  $h(98,0) = 7$
  $h(98,1) = (7+11) \bmod 13 = 5$
  $h(14,0) = 1$
  $h(14,1) = (1+4) \bmod 13 = 5$
  $h(14,2) = (1+2*4) \bmod 13 = 9$



Análisis de Hashing Cerrado (caso búsqueda no existosa = inserción)

- El número de pruebas promedio en búsqueda no exitosa en hashing cerrado es a lo más $1/(1-\alpha)$. Suponemos hashing uniforme y α= factor de carga = n/m.
- Este tiempo es el mismo del tiempo promedio de inserción del próximo elemento.
- Desarrollo:
  - Recordar que:

$$
\begin{aligned}
E[X] &= \sum_{i=0}^{\infty} iP\{X = i\} \\
&= \sum_{i=0}^{\infty} i\left(P\{X \geq i\} - P\{X \geq i+1\}\right) \\
&= 0 * P\{X \geq 0\} - 0 * P\{X \geq 1\} \\
&+ 1 * P\{X \geq 1\} - 1 * P\{X \geq 2\} \\
&+ 2 * P\{X \geq 2\} - 2 * P\{X \geq 3\} \\
&+ .. + \\
&+ (i-1) * P\{X \geq i-1\} - (i-1) * P\{X \geq i\} \\
&+ i * P\{X \geq i\} - i * P\{X \geq i+1\} + ... \\
&= P\{X \geq 1\} + P\{X \geq 2\} + P\{X \geq 3\} + P\{X \geq 4\} + P\{X \geq 5\} + ... \\
&= \sum_{i=1}^{\infty} P\{X \geq i\}
\end{aligned}
$$

Análisis de Hashing Cerrado (caso búsqueda no existosa = inserción) (CONT...)

- El tiempo de inserción o de búsqueda no exitosa se puede determinar como:

$$I = 1 + \sum_{i=0}^{\infty} i * P\{\text{debamos hacer exactamente i pruebas adntes de insertar}\}$$

$$= 1 + \sum_{i=0}^{\infty} P\{\text{debamos hacer almenos i pruebas adntes de insertar}\}$$

$$= 1 + \sum_{i=0}^{\infty} \mathrm{p_i}$$

$$p_1 = n/m$$

$$p_2 = \left(\frac{n}{m}\right)\left(\frac{n-1}{m-1}\right)$$

$$p_i = \left(\frac{n}{m}\right)\left(\frac{n-1}{m-1}\right)\cdots\left(\frac{n-i+1}{m-i+1}\right) \le \left(\frac{n}{m}\right)^i = \alpha^i$$

$$\therefore I = 1 + \sum_{i=0}^{\infty} \mathrm{p_i} \le 1 + \alpha^1 + \alpha^2 + \alpha^3 + \ldots$$

$$= \frac{1}{1-\alpha}$$

Análisis de Hashing Cerrado (caso búsqueda existosa = promedio de inserción desde 1 a n)

- El tiempo de búsqueda exitosa es $\frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right)+\frac{1}{\alpha}$
- ¿Por qué? La secuencia seguida para buscar la clave k es la misma seguida cuando k fue insertada. Por lo tanto basta calcular el promedio de pruebas para insertar las claves desde la primera hasta la n-ésima.
- Si k es la (i+1)ésima clave, el número esperado de intentos para su inserción es 1/(1-i/m). Por lo tanto en número promedio de intentos será la suma de todos los intentos dividida por el número de claves insertadas.

Análisis de Hashing Cerrado (caso búsqueda existosa = promedio de inserción desde 1 a n) (CONT.)

- Si k es la (i+1)ésima clave, el número esperado de intentos para su inserción es 1/(1-i/m). Por lo tanto en número promedio de intentos será la suma de todos los intentos dividida por el número de claves insertadas.

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{1}{\left(1-\frac{i}{m}\right)}=\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{(m-i)}=\frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{(m-i)}=$$

$$=\frac{1}{\alpha}\left(\frac{1}{m}+\frac{1}{m-1}+\frac{1}{m-2}+\frac{1}{m-3}+...+\frac{1}{m-(n-1)}\right)$$

$$=\frac{1}{\alpha}\left(\frac{1}{m}+\frac{1}{m-1}+\frac{1}{m-2}+\frac{1}{m-3}+...+\frac{1}{m-(n-1)}+\left(\frac{1}{m-n}+..+1\right)-\left(\frac{1}{m-n}+..+1\right)\right)$$

$$=\frac{1}{\alpha}\left(\sum_{j=1}^{m}1/\ j-\sum_{j=1}^{m-n}1/\ j\right)$$

$Pero: \ln i \le \sum_{j=1}^{i} 1/\ j \le \ln i + 1$ Se puede llegar a esto usando la integral de 1/x como aproximación.

$$=\frac{1}{\alpha}\left(\ln m+1-\ln(m-n)\right)=\frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right)+\frac{1}{\alpha}$$