# 7. Tablas

Tablas son un organización secuencial de celdas. Una tabla de símbolos es una tabla donde se almacena un símbolo y su descripción o valor (ejemplo, tipo o valor). Por ejemplo, variables ocupadas por un proceso pueden ser almacenadas en un tabla de símbolos, donde cada variable tiene un símbolo y un tipo.

**structure** *TABLE_SYMBOL(NAME,VALUE)*
**declare**
$NEW() \rightarrow tableS$
$ADD(tableS, name, value) \rightarrow tableS$
$DELETE(tableS, name) \rightarrow tableS$
$IS\_IN(tableS, name) \rightarrow boolean$
$VAL(tableS, name) \rightarrow value$
$IS\_EMPTY(tableS) \rightarrow boolean$
$SIZE(tableS) \rightarrow int$
**for all** $t \in TABLE\_SYMBOL, i \in VALUE, s, s_1 \in NAME$ **let**
$VAL(NEW(), s) ::= error$
$VAL(ADD(t, s, i), s_1) ::=$ if $s = s_1$ then $i$ else $VAL(t, s_1)$
$IS\_IN(NEW(), s) ::= false$
$IS\_IN(ADD(t, s_1, i), s) ::= (s = s_1) | IS\_IN(t, s)$
$SIZE(NEW()) ::= 0$
$SIZE(ADD(t, s_1, i)) ::=$1+SIZE(t)
$IS\_EMPTY(t) ::= (SIZE(t) = 0)$
$DELETE(NEW(), s) ::= NEW()$
$DELETE(ADD(t, s_1, i), s) ::=$ if $s = s_1$ then t
else $ADD(DELETE(t, s), s_1, i)$

Ejemplo:

Considere la siguiente declaración de variables en un programa:

int i,j;
float x;

La tabla de símbolos que representa esta declaración es:

ADD(ADD(ADD(NEW(),i,int),j,int),x,float)

## 7.1. Tablas de direccionamiento directo

Direccionamiento directo es una técnica simple que funciona bien cuando el Universo $U$ de claves a almacenar es pequeño (donde claves son el campo de la tabla por el que se busca información, es

decir, símbolo, id, etc.). Suponga que una aplicación necesita un conjunto dinámico en el cual cada elemento tiene una clave sacada de un universo $U = \{0, \ldots, m-1\}$, donde $m$ es no muy grande. En este universo, no hay dos claves repetidas.

Para representar el conjunto dinámico, usamos un arreglo o tabla de direccionamiento directo. Sea $T[0, \ldots, m-1]$ una tabla en la cual cada posición o celda corresponde a una clave del universo $U$. Si el conjunto contiene no elemento con clave $k$, entonces $T[k] = nil$.

El direccionamiento es trivial:

Direct-AddressSearch$(T, k)$ return $T[k]$

Direct-AddressInsert$(T, k)$ $T[k] \leftarrow x$

Direct-AddressDelete$(T, k)$ $T[k] \leftarrow nil$

Cada una de estas operaciones es de orden $O(1)$.

# 8. Tablas Hash

El problema del direccionamiento directo es obvio: si el universo $U$ es grande, almacenamiento en una tabla $T$ de tamaño $|U|$ puede ser impráctico o incluso imposible. Más aún, el conjunto $K$ de claves almacenadas en un momento dado puede ser tan pequeño que mucho del espacio en $T$ sería desperdiciado.

En un direccionamiento directo, un elemento con clave $k$ es almacenado en celda o slot $k$. Con *hash*, este elemento es almacenado en el slot $h(k)$, es decir, una función hash es dada para calcular el slot de la clave $k$. Así, $h$ mapea el universo de claves $U$ a los slots de la tabla $T[0, \ldots, m-1]$:
$h : U \rightarrow \{0, 1, \ldots, m-1\}$

El problema de esta gran idea es que dos claves pueden tener el mismo valos $h$, lo que se llama colisión. Dado esto, una propiedad deseada de este tipo de función es que sea fácil de calcular y que no genere muchas colisiones. Se desea una función hash que dependa de todos los caracteres de la clave y que para una clave elegida aleatoriamente se tenga una probabilidad homogenea de distribución entre los slots de la tabla. A este tipo de función se le llama una función hash uniforme.

## 8.1. Funciones Hash

Una buena función hash satisface la hipótesis de hashing uniforme simple: cada clave tiene la misma probabilidad de ser asignada a cualquiera de los $m$ slots, independientemente de donde haya sido asignada otra clave. Lamentablemente, no se sabe a priori la distribución probabilística deacuerdo a la cual las claves son asignadas. Más aún, las claves pueden no ser generadas en forma independiente.

Algunas funciones hash son las siguientes:

- *División*: Usando el operado $k$ mod $m$. Cuando se usa la función *mod*, se evita usualmente ciertos valores de $m$. Por ejemplo, $m$ no deberiera ser una potencia de 2, ya que si $m = 2^p$, entonces $h(k)$ es justo los $p$ bits más pequeños de $k$. Así, un número primo para $m$ suele ser una buena opción.

- Multiplicación:

$$h(k) = \lfloor m(kA \bmod 1 \rfloor$$

donde $0 < A < 1$ y $kA$ mod 1 significa la parte fraccionaria de kA, es decir $kA - \lfloor kA \rfloor$.

La ventaja de la función multiplicación es que el valor de $m$ no es crítico.

- *Universal hash*: Es una función hash escogida aleatoriamente de un conjunto de funciones hash que garantiza un buen rendimiento en promedio.

Sea cual sea la función hash, se define como *factor de carga* $\alpha$ a el número de claves almacenadas por número de slots.

## 8.2. Manejo de colisiones por encadenamiento

Por cada slot en la tabla existe una lista encadenada que contiene las claves cuya función hash da ese valor de slot.

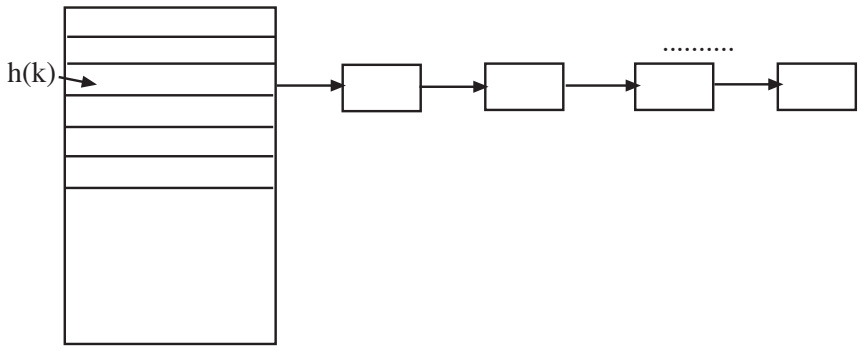


Figura 18: Tabla hash con resolución de colisiones por encadenamiento

En una tabla hash con resolución de colisiones por encadenamiento, un búsqueda sin éxito toma el tiempo $O(1 + \alpha)$, en promedio, asumiendo una distribución uniforme. Este teorema se basa en la idea que para que una búsquena sea infructosa, uno debe encontrar el slots donde debiera estar y recorrer toda la lista encadena. Asumiendo que el hashing es uniforme simple, cualquier clave tiene la misma probabilidad de ser asignada a cualquier slot. Por lo que en promedio, se tiene que la lista encadena de cualquier slot es $\alpha$ y la búsqueda infructosa es $O(1 + \alpha)$.

En una tabla hash con resolución de colisiones por encadenamiento, un búsqueda exitosa toma el tiempo $O(1+\alpha)$, en promedio, asumiendo una distribución uniforme. Para probar el último teorema, asuma una tabla hash donde cualquier elemento tiene la misma probabilidad de ser buscado. Para obtener el valor esperado del número de elementos examinados para encontrar uno determinado $x$, se toma el promedio, sobre los $n$ elementos en la tabla, de una 1 más el número de elementos ingresados antes de agregar $x$ a la lista,

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{i-1}{m}\right) &= 1+\frac{1}{nm}\sum_{i=1}^{n} i-1 \\ &= 1+\frac{1}{nm}\left(\frac{(n-1)n}{2}\right) = 1+\frac{\alpha}{2}-\frac{1}{2m} \\ &= O(1+\alpha)\end{aligned}$$

## 8.3. Manejo de colisiones por direccionamiento abierto

En direccionamiento abierto, todos los elementos son almecenados en la tabla misma. Esto significa que cada entrada en la tabla tiene un elemento o ninguno ($\alpha \leq 1$). Así no más claves que el número de slots en la tabla pueden ser localizadas. La ventajas en que no se manejan punteros de ningún modo.

Para ingresar una clave en direccionamiento abierto, se examina sucesivamente la tabla hasta que se encuentra un slot vacío. La secuencia de posiciones revisadas depende de la clave a ser insertada. Para determinar el slot a revisar, se extiende la función hash para incluir un número de revisión (comenzando con 0) como un segundo input. Así la función hash es:

$$\begin{aligned} h &: U \times \{0,1,\ldots,m-1\} \rightarrow \{0,1,\ldots,m-1\} \\ \text{con} & \\ & \{h(k,0), h(k,1), \ldots, h(k,m-1)\}\end{aligned}$$

**Procedure** $HASH\_INSERT$(`var` $T, k, v$)
    i := 0
    repeat $j := h(k,i)$[
        if $T(j).key = nil$ do [
            $T(j).key := k$
            $T(j).value := v$]
        else $i := i+1$
    until $i = m$]
    error "hash overflow"
**end** $HASH\_INSERT$

**Procedure** $HASH\_SEARCH(T,k)$
    i := 0
    repeat $j := h(k,i)$[
        if $T(j).key = k$ do return j
        else $i := i+1$
    until $i = m$ or $T(j) = nil$ ]
    return nil
**end** $HASH\_SEARCH$

Algunas funciones para direccionamiento abierto son:

- $h(k,i) = (h(k)+i) \text{mod} m$

- $h(k,i) = (h(k) + c_1 i + c_2 i^2) \text{mod} m$
- $h(k,i) = (h_1(k) + i h_2(k)) \text{mod} m$

Dada una tabla hash con direccionamiento abierto y factor de carga $\alpha = \lfloor \frac{n}{m} \rfloor < 1$, el número esperado de revisiones en un búsqueda sin éxito es a lo más de $\frac{1}{(1-\alpha)}$, asumiendo hashing uniforme.

Sea $X$ una variable aletaoria representando el número de intentos o revisiones en una búsqueda sin éxito, y sea $q_i$ la probabilidad de que al menos $i$ slots ocupados sean revisados, el número esperado de revisiones infructosas en una tabla es:

$$\begin{aligned} E[X] &= \sum_{i=0}^{\infty} i Pr\{X = i\} \\ &= \sum_{i=0}^{\infty} i(Pr\{X \geq i\} - Pr\{X \geq i+1\}) \\ &= \sum_{i=1}^{\infty} Pr\{X \geq i\} \\ &= \sum_{i=1}^{\infty} q_i \end{aligned}$$

La probabilidad que el primer intento resulte en un slot ocupado es $q_1 = \frac{n}{m}$ y sucesivamente:

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

lo que nos queda que

$$\sum_{i=1}^{\infty} q_i \leq 1 + \alpha + \alpha^2 \ldots = \frac{1}{1-\alpha}$$

De lo anterior, insertar un elemento en una tabla hash con direccionamiento abierto con factor de carga $\alpha$ requiere de a lo más $1/(1-\alpha)$ intentos en promedio, usando un hashing uniforme. Insertar una clave requiere una búsqueda sin éxito seguida por la localización de la clave en el slot vacío. Thus, el número esperado de intentos es a lo más de $1/(1-\alpha)$.

Dada una tabla con direccionamiento abierto $\alpha < 1$, el número esperado de revisiones en una búsqueda exitosa en a lo más:

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

Si $k$ fue la $i+1$ clave insertada en la tabla hash, el número esperado de revisiones hechas en la bsqueda por $k$ es a lo más $\frac{1}{(1-i/m)} = \frac{m}{m-i}$. Promediando sobre todas las claves en la tabla hash nos da que el número promedio de revisiones en una búsqueda existosa es:

$$\begin{aligned}\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} &= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} \\ &\leq \frac{1}{\alpha}(H_m - H_{m-n})\end{aligned}$$

donde $H_i = \sum_{j=1}^{i} 1/j$ es el *i-ésimo* número armónico. Usando definiciones de números armónicos, se tiene que:

$$\begin{aligned}\frac{1}{\alpha}(H_m - H_{m-n}) &= \frac{1}{\alpha} ln \frac{m}{m-n} \\ &\frac{1}{\alpha} ln \frac{1}{1-\alpha}\end{aligned}$$

## 8.4. Ejercicios

- Las siguientes operaciones son definidas para una tabla de símbolos la cual maneja un lenguaje con bloques. Defina la especificación algebraica de esta tabla:
  - CREAR: crea una tabla vacía con un número determinado de bloques $m$. Cada bloque tiene una capacidad máxima $p$.
  - AGREGA_CLAVE: localiza una clave y su atributo en la tabla. Las claves sólo se agregan al bloque que se especifique como entrada, siempre y cuando exista el bloque y tenga espacio.
  - BORRAR_CLAVE: borra clave y atributo.
  - BORRAR_BLOQUE: borra todas las claves y atributo de un bloque especificado.
  - RECUPERA: retorna los atributos de la clave especificada
  - ES_IN retorna verdadero si existe la clave en la tabla.
  - SIZE: retorna el número de claves es un bloque particular.
- Suponga que se usa una función hash $h()$ con $n$ claves distintas para una tabla hash $T$ de largo $m$. ¿ Cuál es el número esperado de colisiones?
- Discuta el hecho que el tiempo de ejecución de un algoritmo de búsqueda en una tabla hash con resolución de colisión usando encadenamiento es independiente si es que las claves se insertan al comienzo o al final de la lista.
- ¿ Qué sucede con el tiempo de ejecución si es que el encademiento mantiene las claves ordenadas en la lista? Búsqueda exitosa o sin éxito.
- Muestre que si $|U| > nm$, existe un subconjunto $n$ en $U$ de claves cuya valor hash dan siempre el mismo slot y el peor tiempo de ejecución es $n$.

## 8.5. Hashing Universal y Hashing Perfecto

Dado un función hash fija, uno puede escoger un conjunto de claves $n$ con el mismo valor hash (celda de la tabla), originando que el peor caso para la tabla hash sea $\Theta(n)$. Toda función fija es vulnerable a este caso peor de rendimiento. La única forma efectiva de mejorar esta situación es escoger una función hash en forma aleatoria de manera que sea independiente de las claves que serán almecenadas en la tabla. Este enfoque se llama *universal hashing*, el que ha dado buenos resultados independiente de las claves escogidas.

La idea es seleccionar una función hash en forma aleatoria de un conjunto de funciones cuidadosamente diseñadas.

Sea $H$ una colección finita de funciones hash que asocian un universo $U$ de claves a un rango $\{0, 1, \ldots, m-1\}$. Tal colección de funciones se dice universal si para cada par de claves diferentes $k, l \in U$, el número de funciones hash $h \in H$ que cumplen que $h(k) = h(l)$ es a lo más $|H|/m$.

**Teorema.** Suponga que una función hash $h$ es escogida de una colección universal de funciones hash y es usada para asociar $n$ claves en una tabla de tamaño $m$, usando encadenamiento para la resolución de colisiones. Si $k$ no está en la tabla, entonces el largo esperado de la lista que es seleccionada por $h(k)$, $E[n_{h(k)}]$, es a lo más $\alpha$. Si la clave $k$ está en la tabla, entonces el largo esperado $E[n_{h(k)}]$ de la lista conteniendo $k$ es a lo más $1 + \alpha$.

**Prueba.** Dada la definición de una colección universal de funciones hash, la colisión de dos claves diferentes tiene probabilidad de a lo más $1/m$, de manera que, para dos claves $k$ y $l$, $E[X_{kl}] \leq 1/m$. Sea $Y_k$ el número de claves distintas a $k$ que colisionan, entonces:

$$\begin{aligned} Y_k &= \sum_{l \neq k, l \in T} X_{kl} \\ E[Y_k] &= E[\sum_{l \neq k, l \in T} X_{kl}] \\ E[Y_k] &= \sum_{l \neq k, l \in T} E[X_{kl}] \\ E[Y_k] &\leq \sum_{l \neq k, l \in T} 1/m. \end{aligned} \tag{1}$$

- Si $k$ no está en la tabla $T$, entonces $n_{h(k)} = Y_k$ y $|\{l : l \in T \wedge l \neq k\}| = n$. Entonces $E[n_{h(k)}] = E[Y_k] \leq n/m = \alpha$.
- Si $k$ está en $T$, entonces la clave $k$ aparece en la lista $T(h(k)$ y el número $Y_k$ no incluye la clave $k$, entonces tenemos $n_{h(k)} = Y_k + 1$ y $|\{l : l \in T \wedge l \neq k\}| = n - 1$. Entonces, $E[n_{h(k)}] = E[Y_k] + 1 \leq (n-1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha$.

Sea $Z_p$ el conjunto $\{0, 1, \ldots, p-1\}$ y$Z_p^*$ el conjunto $\{1, 2, \ldots, p-1\}$, con $p$ un número primo grande tal que cada clave posible $k$ esta en el rabgo 0 a p-1, se define una función hash como $h_{a,b}(k) = ((ak + b)\ mod\ p)\ mod\ m$. La familia de todas las funciones hash tal que $H_{p,m} = \{h_{a,b} : a \in Z_p^*$ and $b \in Z_p\}$ son una colección universal.

Se llama **hashing perfecto** a la técnia que usa hashing y donde el peor número de accesos requeridos para realizar una búsqueda es de O(1). El perfecto hashing es una idea simple donde se usan

dos niveles de hashing con hashing universal a cada invel. El primer nivel es un esquema hashing simple: $n$ claves son asociados a $m$ celdas de una tabla usando una función hash selecionada de una familia de funciones hash universales. En vez de tener una lista encadena a la celda de primer nivel se tiene una segunda tabla hash $S_j$, la cual tiene asociada una función hash $h_j$.

Para garantizar no colisiones a un segundo nivel, el tamaño de la tabla $S_j$ debe ser el cuadrado del número de claves $n_j$ que son asociadas a la celda $j$. Este número no es muy grande, siempre que se escoja una buena función hash para el primer nivel.

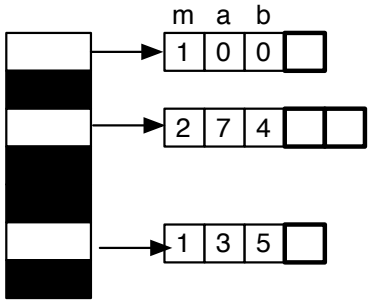


Figura 19: Estructura de un hash perfecto donde cada entrada define una nueva tabla hash con parametros $m$,$a$,$b$