



Tablas Hash

Profesor: Julio César López

jlopez@eisc.univalle.edu.co

October 29, 2002

Contents



Page 1 of 21

Full Screen

Quit



Contenido

1. Tablas de Direccionamiento Directo
2. Tablas Hash
 - (a) Solución de colisiones con encadenamiento
 - (b) Análisis de Hashing con Encadenamiento
3. Funciones Hash
 - (a) El Método de la División
 - (b) El Método de Multiplicación

Contents



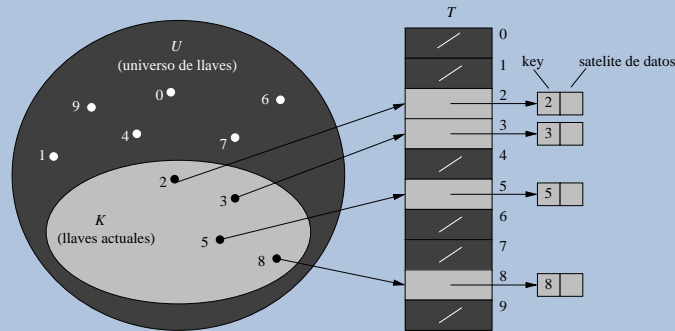
Page 2 of 21

Full Screen

Quit

Tablas de Direccinamiento Directo

Para representar conjuntos dinámicos, se usa un arreglo o un **tabla de direccionamiento directo** $T[0..m - 1]$, en la cual cada posición, o **slot**, corresponde a una llave en el universo de llaves U .



En la figura se implementa una tabla de direccionamiento directo T . Cada llave en el universo $U = \{0, 1, \dots, 9\}$ corresponde a un índice en la tabla. El conjunto $K = \{2, 3, 5, 8\}$ de llaves actuales determina los *slots* en la tabla que contienen apuntadores a elementos. Los otros *slots* contienen NIL.

Tablas de Direccionamiento Directo (cont.)

Las operaciones son triviales de implementar:

DIRECT-ADDRESS-SEARCH(T, k)
return $T[k]$

DIRECT-ADDRESS-INSERT(T, x)
 $T[key[x]] \leftarrow x$

DIRECT-ADDRESS-DELETE(T, x)
 $T[key[x]] \leftarrow \text{NIL}$

Cada una de la operaciones es rápida: solo $O(1)$ es requerido.

Contents



Page 4 of 21

Full Screen

Quit



Tablas Hash

- Cuando se trabaja con tablas de direccionamiento directo, si el universo es grande, almacenar una tabla T de tamaño $|U|$ puede llegar a ser difícil dada la capacidad de memoria que tenga el computador. Además, el conjunto K de llaves actuales puede ser tan pequeño comparado con U que el resto de memoria se desperdicia.
- Una **tabla hash** requiere mucho menos espacio de almacenamiento si el conjunto K es mucho mas pequeño que el universo. Específicamente, los requerimientos pueden reducirse a $\Theta(|K|)$.

Contents



Page 5 of 21

Full Screen

Quit

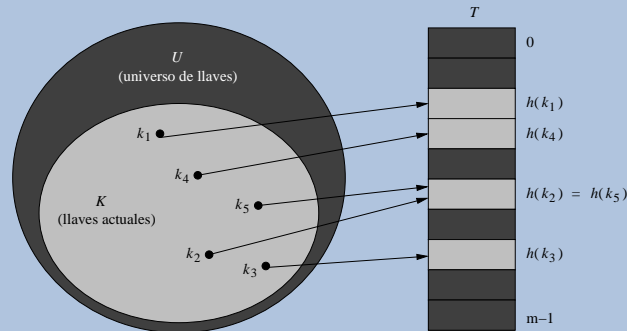
Tablas Hash (cont.)

Con direccionamiento directo, un elemento con llave k es almacenado en el *slot* k . Con *hashing*, el elemento es almacenado en el *slot* $h(k)$; esto es, una **función hash** h es usada para encontrar el *slot* de la llave k .

Aquí h asigna las llaves del universo a los *slots* de una tabla hash $T[0..m-1]$:

$$h : U \rightarrow \{0, 1, \dots, m-1\}$$

Se dice que $h(k)$ es el **valor hash** de la llave k .





Tablas Hash (cont.)

- La idea de la función hash es reducir el rango de índices que necesitan ser manejados. En vez de $|U|$, se necesitan solo m valores.
- El problema de esta idea es que dos llaves podrían ser asignadas al mismo *slot* – una **colisión**. Sin embargo, para evitar este problema existen técnicas efectivas.
- Por ejemplo, una solución es hacer que h sea aparentemente “al azar” (por supuesto, una función hash h debe ser determinística para que, dada una entrada k , siempre produzca la misma salida $h(k)$). No obstante, como $|U| > m$, puede haber más de dos llaves que tengan el mismo valor hash; como consecuencia, evitar colisiones es imposible!.

Contents



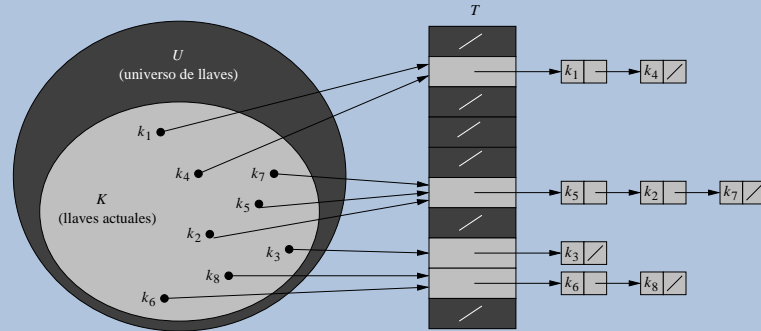
Page 7 of 21

Full Screen

Quit

Solución de colisiones con encadenamiento

Una forma de resolver las colisiones es el llamado **encadenamiento**.



En la figura, cada $slot\ T[j]$ contiene una lista encadenada de todas las llaves cuyo valor hash es j .



Encadenamiento (cont.)

Las operaciones en una tabla hash T son fáciles de implementar cuando las colisiones son resueltas con encadenamiento.

CHAINED-HASH-INSERT(T, x)

inserte x en la cabeza de la lista $T[h(key[x])]$

CHAINED-HASH-SEARCH(T, k)

busque un elemento con llave k en la lista $T[h(k)]$

CHAINED-HASH-DELETE(T, x)

borre x de la lista $T[h(key[x])]$

El peor de los casos para la inserción es $O(1)$. Para la búsqueda, el peor de los casos es proporcional al tamaño de la lista. La eliminación de un elemento x puede ser realizado en $O(1)$ si la lista es doblemente encadenada.

Contents



Page 9 of 21

Full Screen

Quit



Análisis de Hashing con Encadenamiento

- Dada una tabla T con m slots que almacenan n elementos, se define el **factor de carga** α para T como n/m , esto es, el número promedio de elementos almacenados en una cadena.
- El comportamiento, en el peor de los casos, de *hashing* con encadenamiento es terrible: todas las n llaves se asignan al mismo *slot*, creando una lista de tamaño n .
- El comportamiento promedio depende de qué tan bien la función hash distribuya el conjunto de llaves a ser almacenados en los m slots. Se asume que cualquier elemento dado es igualmente asignado a cualquiera de los m slots, independientemente de donde se han asignado los otros elementos. Eso es llamado la suposición de **hashing uniforme simple**.

Contents



Page 10 of 21

Full Screen

Quit



Análisis (cont.)

Teorema 1

En una tabla hash en la cual las colisiones son resueltas con encadenamiento, una búsqueda sin éxito toma un tiempo $\Theta(1 + \alpha)$, en promedio, bajo la suposición de *hashing* uniforme simple.

Contents



Page 11 of 21

Full Screen

Quit



Análisis (cont.)

Teorema 2

En una tabla hash en la cual las colisiones son resueltas con encadenamiento, una búsqueda exitosa toma un tiempo $\Theta(1 + \alpha)$, en promedio, bajo la suposición de *hashing* uniforme simple.

Contents



Page 12 of 21

Full Screen

Quit

Análisis (cont.)

- Todo el análisis anterior significa que si el número de *slots*, en una tabla hash, es proporcional al número de elementos en la tabla, se tiene $n = O(m)$ y de esta forma, $\alpha = n/m = O(m)/m = O(1)$.
- Por lo tanto, la búsqueda toma tiempo constante en promedio.
- Como la inserción toma $O(1)$ en el peor de los casos, y el borrado toma $O(1)$ en el peor de los casos cuando las listas están doblemente encadenadas, todas las operaciones pueden tomar tiempo constante en promedio.

Contents



Page 13 of 21

Full Screen

Quit

Funciones Hash

- Una buena función hash satisface (aproximadamente) la suposición de *hashing* uniforme simple.
- Formalmente, se asume que cada llave es asignada independientemente de U de acuerdo a una distribución de probabilidad P ; esto es, $P(k)$ es la probabilidad de que k sea asignada.
- Entonces se tiene que

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m} \quad \text{para } j = 0, 1, \dots, m - 1 \quad (1)$$

Contents



Page 14 of 21

Full Screen

Quit

Funciones Hash (cont.)

Algunas veces se desconoce la distribución P .

Por ejemplo, suponiendo que las llaves son números reales k “al azar” independientes y uniformemente distribuidos en el rango $0 \leq k < 1$, la función hash

$$h(k) = \lfloor km \rfloor$$

puede satisfacer la ecuación 1.



Funciones Hash (cont.)

- En la práctica, algunas heurísticas pueden ser usadas para crear una función hash con un buen desempeño.
- La información cualitativa de P a veces es útil en el proceso de diseño. Por ejemplo, en la tabla de símbolos de un compilador, las llaves son cadenas de caracteres que representan identificadores en un programa. Es común que símbolos muy parecidos, como `pt` y `pts`, aparezcan en el mismo programa. Una buena función hash minimizará la posibilidad que dichos símbolos sean asignadas al mismo *slot*.

Contents



Page 16 of 21

Full Screen

Quit



El Método de la División

En el **método de la división** para crear funciones hash, se asigna una llave k a uno de los m slots tomando el residuo de k dividido por m . La función hash es

$$h(k) = k \bmod m.$$

Por ejemplo, si la tabla hash tiene tamaño $m = 12$ y la llave es $k = 100$, entonces $h(k) = 100 \bmod 12 = 4$.

Contents



Page 17 of 21

Full Screen

Quit



El Método de la División (cont.)

- Cuando se usa el método de la división, usualmente se evitan ciertos valores de m . Por ejemplo, m no debe ser una potencia de 2, ya que si $m = 2^p$, entonces $h(k)$ es igual a los p bits de más bajo orden de k .
- Buenos valores para m son números primos no cercanos a potencias de 2 exactas. Por ejemplo, suponiendo que se desea llenar una tabla hash (con las colisiones solucionadas por encadenamiento) con $n = 2000$ cadenas de caracteres, donde un caracter tiene 8 bits, el número 701 puede ser escogido ya que es un primo cercano a $\alpha = 2000/3$ y lejano a alguna potencia de 2.
- Tratando cada k como un entero, la función hash será

$$h(k) = k \bmod 701.$$

Contents



Page 18 of 21

Full Screen

Quit



El Método de la Multiplicación

El **método de la multiplicación** para crear funciones hash opera en dos pasos.

1. Se multiplica la llave k por una constante A en el rango $0 < A < 1$ y se extrae la parte fraccionaria de kA .
2. Se multiplica este valor por m y se toma el piso del resultado.

La función hash es

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

donde “ $k A \bmod 1$ ” significa la parte fraccionaria de kA , que es, $kA - \lfloor kA \rfloor$.

Contents



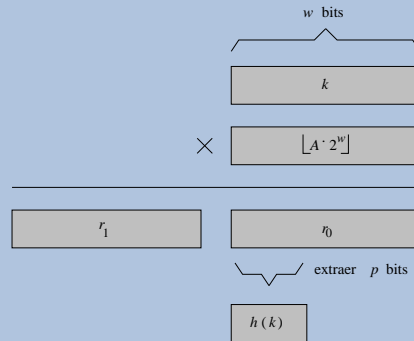
Page 19 of 21

Full Screen

Quit

El Método de la Multiplicación (cont.)

- La ventaja del método de la multiplicación es que el valor de m no es crítico. Comúnmente se escoge como una potencia de 2, $m = 2^p$ para un entero p .
- Este método se puede implementar fácilmente en los computadores:



- Suponiendo que el tamaño de palabra en la máquina es w bits y que k cabe en una palabra, se multiplica primero k por el entero $[A \cdot 2^w]$. El resultado es un valor $r_1 2^w + r_0$, donde r_1 es la palabra de más alto orden del producto y r_0 es la palabra de más bajo orden del producto.

El Método de la Multiplicación (cont.)

- Aunque este método funciona para cualquier valor de A , se ha encontrado que funciona mejor con algunos valores.
- Donald E. Knuth sugiere que

$$A \approx (\sqrt{5} - 1)/2 = 0.6180339887... \quad (2)$$

- Como un ejemplo, si se tiene $k = 123456$, $m = 10000$, y A es como en la ecuación 2, entonces

$$\begin{aligned} h(k) &= \lfloor 10000 \cdot (123456 \cdot 0.61803... \bmod 1) \rfloor \\ &= \lfloor 10000 \cdot (76300.0041151... \bmod 1) \rfloor \\ &= \lfloor 10000 \cdot 0.0041151... \rfloor \\ &= \lfloor 41.151... \rfloor \\ &= 41. \end{aligned}$$

Contents



Page 21 of 21

Full Screen

Quit