Estructuras de datos
Clase teórica 10

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MEDELLÍN

Contenido

- Tablas hash
- Hashing por encadenamiento
- Funciones hash universales

Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Introducción

Supongamos que necesitamos almacenar elementos que tienen una clave única, siendo |S| la cantidad de claves que almacenamos y U el universo de claves posibles ($S \leq U$). Supongamos además, que aparte de la inserción, necesitamos únicamente de las operaciones de búsqueda y borrado.

Por mencionar algunos ejemplos de esta situación imaginemos programas para:

- Manejar una agenda telefónica (10^2 - 10^3) vs. números de teléfono (10^7)
- Llevar registro de los usuarios de un aplicativo (10^3 - 10^4) vs. cédulas (10^{10})
- Almacenar los visitantes de un sitio web (10^4 – 10^7) vs. direcciones IP (2^{32})

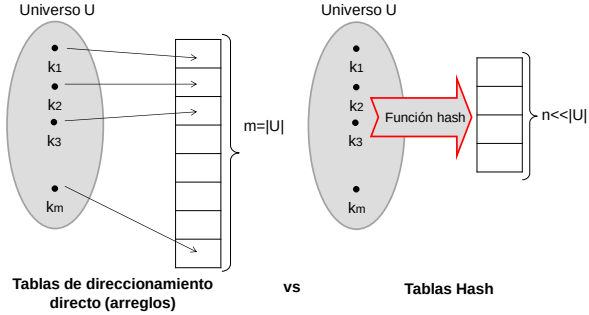
Analicemos las opciones que hemos visto en clase:

- Un **arreglo dinámico** permitiría *O(n)* en la inserción, *O(n)* en el borrado y *O(log(n))* en la búsqueda (y eso si está ordenado).
- Si conocemos U nos podemos ahorrar la inserción y el borrado con un **arreglo estático** que abarque todo U, pero igual gastaríamos *O(log(n))* + *O(n*log(n))* con el problema adicional que requeriríamos muuucha memoria (|U|).
- Una **lista enlazada** nos daría *O(1)* para la inserción, *O(n)* para la búsqueda y *O(n)* para el borrado.
- Un **árbol binario de búsqueda balanceado** también permite almacenar elementos con claves únicas (no repetidos) con *O(log(n))* para la inserción, búsqueda y borrado.

¿Qué tal si les digo que una tabla hash permite *O(1)* en promedio para las tres operaciones?

Tablas hash

Tablas hash



Tablas de direccionamiento directo (arreglos) **vs** **Tablas Hash**

Tablas hash

Principios generales de una tabla hash:

- Utiliza un arreglo *A* de tamaño *n*, siendo *n* proporcional a |S|, por ejemplo *n* = 2*|S|
  Aunque S puede variar, la idea es que *n* permanezca relativamente constante
- Tiene una función *h*, cuyo rango es U y cuyo dominio es el intervalo cerrado [0,*n*-1]
  La idea es que *h* "reparta" las claves lo más homogéneamente posible, prácticamente de manera aleatoria
- Almacena el elemento de clave *k* en *A*[*h*(*k*)]
- Posee un mecanismo para manejar "colisiones"
  Por más *h* sea homogénea las colisiones son inevitables. Un ejemplo de esto es el "paradigma del cumpleaños"

Paradigma del cumpleaños

Supongamos que queremos guardar en una tabla hash cierta información de un grupo de personas. Supongamos adicionalmente que usamos como función hash la fecha de cumpleaños (transformada en un valor entero 1-365). ¿Cuál es la cantidad mínima de personas que debe haber para que exista una probabilidad del 50% de que dos de ellas tengan la misma fecha de cumpleaños?

Increíblemente es sólo 20 personas

Veamos por qué …

Paradigma del cumpleaños

Sea *P* la probabilidad que dos personas tengan la misma fecha de cumpleaños

Sea *Pd* la probabilidad de que una persona cumpla años en un día *d* específico = *1/365*

Por tanto, la probabilidad que dos personas cumplan años en el mismo día *d* es *(1/365)*(1/365)*

La probabilidad que dos personas cumplan años en cualquier día es *365*(1/365)*(1/365) = 1/365*

Si hay *n* personas, la cantidad de parejas diferentes es *n*(*n*-1)/2

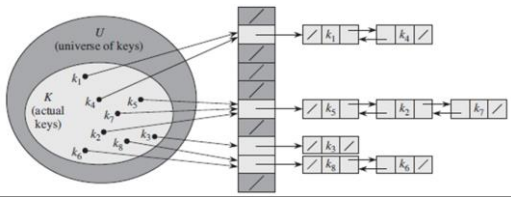
Luego $P = \frac{n(n-1)}{2}\frac{1}{365}$, si P = 0.5 entonces $n^2 - n - 365 = 0$

De donde *n = 19.61*

Hashing mediante encadenamiento

Ya que las colisiones son inevitables, es decir, para *x*, *y* en U, *h(x) = h(y)* se hace necesario algún mecanismo para manejarlos, siendo el más común el encadenamiento, el cual consiste en:

- Almacenar, no un elemento en el arreglo, si no una lista enlazada
- Dada la clave *k* de un elemento, las operaciones de inserción, búsqueda y eliminación se realizan en la lista que se encuentra en *A[h(k)]*



Funciones hash

En una tabla hash con encadenamiento la inserción es *O(1)* siempre y cuando se realice siempre al final de la lista.
Por su parte la búsqueda y la eliminación son *O(longitud de la lista)* siempre y cuando la lista sea doblemente enlazada.
Dado que dicha longitud puede estar entre 0 y |S|, la eficiencia total de la tabla hash depende de la definición de la función hash.

Propiedades de una "buena" función hash:

1. Debe distribuir los elementos de manera homogénea, es decir, de forma igualmente probable en cada espacio del arreglo. Prácticamente con una distribución aleatoria uniforme; pero
2. Debe ser debe ser determinística para poder hacer las operaciones de búsqueda y eliminación; y
3. Debe ser fácil de evaluar para no agregarle complejidad

Un ejemplo de lo que NO se debe hacer a la hora de definir una función hash es el siguiente:

Retomando el caso de la agenda telefónica tenemos |U| = 10^7 si suponemos que |S| = 10^3, *h(x)* podría ser INT(*x**10^{-4}) ¿Qué tiene de malo esta función? Pues que usualmente los prefijos de los números de teléfono, es decir, los tres primeros dígitos son muchos menos que 999, lo cual implicaría una distribución muy heterogénea: posiciones del arreglo muy pobladas y otras completamente vacías.

Otra solución para un U de enteros es *h(x) = x % |S|*. Esta alternativa podría ser buena siempre y cuando la distribución de valores de U no comparta un factor común con |S|. Por ejemplo, si todos los valores de U son pares y |S| es par, de entrada la mitad de las posiciones del arreglo estarían vacías.

Funciones hash

Una regla general para definir una función hash puede ser:

1. Si las claves de U no corresponden a un valor entero *x*, convertirlas a uno. Por ejemplo, si son strings, convertir cada carácter al valor ASCII correspondiente y luego realizar una operación de agregación.

2. Realizar un proceso de "compresión" mediante *x%n* para llevar los valores numéricos al intervalo [*0*,*n-1*] pero considerando que:
   - *a una variable aleatoria entera*
   - *n* debe ser proporcional a |S|, usualmente se usa *n* ≈ 2*|S|
   - Para evitar factores comunes entre *x* y *n*, la mejor alternativa es usar un *n* primo, usualmente se usa el más cercano a 2*|S|

La función anterior garantiza que la probabilidad de que haya una colisión es a lo sumo $1/n$. En otras palabras:

$P_{h \in H}[h(x) = h(y), x \neq y] \leq \frac{1}{n}$

Prueba:
h(x) = h(y) → x % n = y % n → ***(x-y) % n = 0***

En esta ecuación ocurren tres cosas:
- $x \neq y$, por tanto $x-y \neq 0$
- *n* es primo, lo cual elimina la existencia de patrones en la aritmética modular

En otras palabras, el resultado del lado izquierdo es un valor entre 0 y *n-1*, todos con la misma probabilidad de salir.

Es decir, la probabilidad de que sea 0 es *1/n*

Ejemplo

Supongamos que existe una tabla hash mediante un arreglo de *n* = 7 posiciones. ¿Cómo quedaría la tabla al insertar los elementos (en ese orden estricto): 8, 14, 19, 3, 10, 9, 12, 1, 6, 15

En la próxima clase …

Comprobación (mediante análisis amortizado) de la eficiencia *O(1)* de las tablas hash y …

Problemas candidatos a resolver mediante tablas hash