Busqueda Mediante Hashing

Daniel Abrilot - John Lopez[1]

8 de noviembre de 2013

[1]Universidad Tecnologica Metropolitana

Índice general

Capítulo 1

Introducción

1.1. Definición Hash:

Se refiere a una función o método para generar claves o llaves que representen de manera casi unívoca a un documento, registro, archivo, etc., resumir o identificar un dato a través de la probabilidad, utilizando una función hash o algoritmo hash. Un hash es el resultado de dicha función o algoritmo.

1.2. Problematica

El tiempo ocupado en ordenar un arreglo y buscar un elemento mediante la búsqueda binaria es similar al de buscar secuencialmente en un arreglo desordenado, por lo cual se creó un método alternativo para trabajar con las operaciones básicas para los datos (ingresar, eliminar y buscar), el Hashing. Su principal característica es que ingresa cada elemento en un lugar específico determinado por el módulo de éste, por lo cual cada vez que se necesite buscar un elemento sólo bastará calcular su posición por el mismo método.

Capítulo 2

Tabla Hash

2.1. Definición Tabla Hash

Una tabla Hash es una estructura de datos que pretende la inserción, búsqueda y borrado de elementos en un tiempo constante.

2.1.1. Como Funciona

Utilizando un función de dispersión (hash(x)) distribuirá los objetos de forma uniforme a lo largo de la tabla.

- Función de dispersion:$valorHash = hash(x)$. es una caja negra que tiene como entrada una llave y como salida una dirección.
- Uso del operador modulo. evita valor hash exceda el tamaño del array. $indiceHash = valorHash$ es un entero entre [0,length(array)-1].

2.1.2. Desventajas de la Tabla Hash

La Tabla Hash permite realizar las operaciones de búsqueda, inserción y borrado pero existen algunas desventajas:

- Almacenar N datos requiere de una Tabla Hash de tamaño M >> N, para reducir el numero de colisiones.
- No es posible obtener una secuencia ordenada de sus elementos sin que se genere un coste de tiempo no constante.

Capítulo 3

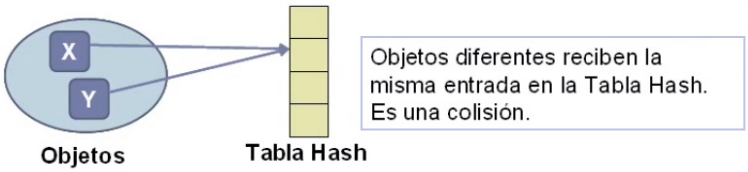
Colisiones

3.1. Definición de Colisión

El hashing es similar al indexamiento en el sentido de asociación entre llaves y direcciones relativas de registros Pero difiere de los índices en 2 cosas: La dirección generada por Hash suele ser aleatoria (random). No hay una relación aparente entre la llave y la localización del registro correspondiente El Hash permite que 2 llaves puedan producir la misma salida > direcciones iguales, a esto se le conoce como "colisión".

- Una colisión se produce cuando objetos diferentes dan lugar al mismo indice hash.

- Objetos o1 y o2, donde $o1 != o2$ && $hash(o1) = hash(o2)$



- Implica un sobrecoste controlar las casillas ocupadas/libes.

- Es complicado evitar las colisiones

3.2. Formas de Resolver las Colisiones

3.2.1. Resolución Mediante Exploración

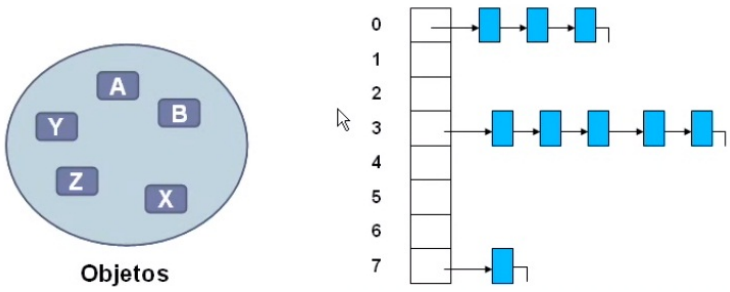
Trata de buscar otra posición libre en la tabla para albergar la inserción del elemento.

- Exploración Lineal: visita la siguiente casilla, si esta libre realiza la inserción sino visita la siguiente.
- Exploración Cuadrática: visita la casilla i 2 posiciones.si esta libre realiza la inserción sino visita la siguiente.

3.2.2. Resolucion Mediante Encadenamiento Enlazado

En cada posición de la tabla se mantiene una lista enlazada en la que se van insertando los elementos cuyo valor hash les asigna la misma posición.

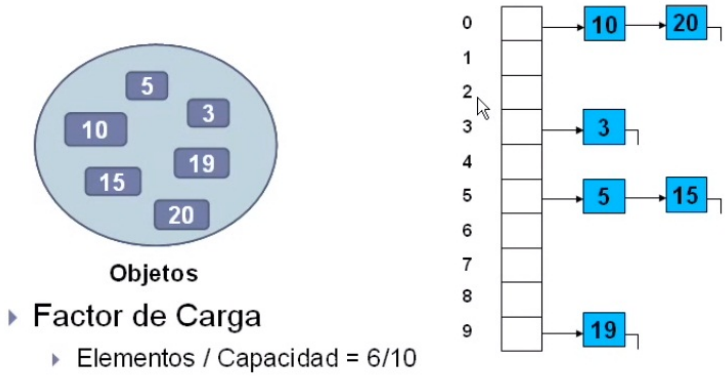
- El Hashing Enlazado usa un vector de Listas Enlazadas.
- Aquellos objetos que reciban un determinado valor de Hash, se insertaran en la lista enlazada correspondiente.



3.2.3. Ejemplo de uso de Hashing Enlazado

Insertar los elementos {5,10,3,15,20,19} en una Tabla Hash de tamaño 10 y con funcion hash:

- hash (x) = x % 10



3.3. Rehashing en una Tabla Hash

Si el Factor de Carga (grado de ocupación) de una Tabla Hash es alto (>75 %), el numero de colisiones aumenta significativamente. Para ello la solución es el Rehashing que consiste en aumentar el tamaño de la tabla para reducir el factor de carga.

- Ventajas: Permite mantener un reducido factor de carga para que las principales operaciones (insertar, buscar, eliminar) se realicen en tiempo constante.
- Inconvenientes: Requiere construir un nuevo array e insertar nuevamente todos los datos.

Capítulo 4

Algoritmos de Hash mas Comunes

4.1. SHA-1

SHA-1: Secure Hash Algorithm, Algoritmo de Hash seguro de síntesis que genera un hash de 160 bits. Se utiliza, por ejemplo, como algoritmo para la rama digital.

4.2. MD2 y MD4

MD2 y MD4: Message-Disgest Algorithm, Algoritmo de Resumen del Mensaje de hash que generan un valor de 128 bits.

4.3. MD5:

MD5: Esquema de hash de hash de 128 bits muy utilizado para autenticación cifrada, Gracias al MD5 se consigue, por ejemplo, que un usuario demuestre que conoce una contraseña sin necesidad de enviar la contraseña a través de la red.

4.4. Ejemplos de Funciones Hash mas Comunes

Residuo de la Division

- Cantidad de direcciones 996; Modulo = 997 H(245643)=2445643 mod 997=381;Dirección:381.

Centro del Cuadrado

- H(123)=123š=15129;Dirección:51 H(730)=730š=532900;Dirección:29

Pliegue

- H(12345678)=123+456+78=657;Dirección:657

Capítulo 5

Ventajas y Desventajas Búsqueda por Hashing

5.1. Ventajas

- Se pueden usar los valores Naturales como llave, puesto que se traducen internamente a direcciones fáciles de localizar
- Se logra independencia lógica y física, debido a que los valores de las llaves son independientes del espacio de direcciones.
- No se requiere almacenamiento adicional para los índices.

5.2. Desventajas

- No se pueden usarse registros de longitud variable.
- El archivo no esta calificado.
- No permite llaves repetidas.
- Solo permite acceso por una sola llave.

Capítulo 6

Complejidad

- Si la función de hash es de buena, habrán pocas colisiones; si hay c colisiones en promedio las operaciones resultaran de complejidad O(c), es decir constante independiente del tamaño de la tabla de hash.
- Ahora si todas las claves que se buscan en la tabla generan colisiones , es decir el peor caso, la complejidad de las operaciones serán de orden O(B) siendo B el tamaño de la tabla.

6.1. Caso Promedio

En el caso promedio de los casos al efectuar la búsqueda de hash secuencial es de orden O(1+N/B). El evaluar la función de hash tiene complejidad 1

6.2. Caso Mejor

En el mejor de los casos si B (cantidad de contenedores) es proporcional a N (cantidad de nodos) de la lista las operaciones resultan de costo constante. O(c);

6.3. Caso Peor

En el peor de los casos llenar una tabla con n items tiene complejidad O(n* (1+n/B)).

Capítulo 7

Codigo en Ruby y Pruebas

7.1. Codigo

```
class BusquedaHash

    def initialize()
    end
    def BH()
        busqueda = Hash.new
        #busqueda es una nueva tabla hash
        clave = 700
        #Se almacena una clave aleatoria
        for i in(0..99)
            hash = clave%100
            #Se aplica funcion hash mod10
            busqueda[hash] = clave
            #Se guarda el valor de la clave en la tabla
            hash con el indice de la funcion
            clave = clave + 1
        end
        #busqueda.each do |key,value|
        #Con esto se pueden visualizar los elementos de la tabla hash
        # puts "#{key} es #{value}"
        #end
        puts "Su elemento es #{busqueda[76]}"
        #Busqueda en el universo de la tabla hash de la clave 76
```

```
    end
end
beginning_time = Time.now
(1..10000).each { |i| i }
tabla = BusquedaHash.new
tabla.BH()
end_time = Time.now
puts "Time elapsed #{(end_time - beginning_time)*1000} millisegundos"
gets()
```

7.2. Pruebas

```
Medicion de tiempos:
----------
Caso N=1000
Clave: 7000
Elemento Buscado: 999
Tiempo transcurrido: 4 milisegundos
----------------------
Caso N=10000
Clave: 70000
Elemento buscado: 7635
Tiempo Transcurrido: 7 milisegundos
----------------------
Caso N = 100000
Clave = 700000
Elemento buscado: 87435
Tiempo Transcurrido: 53 milisegundos
----------------------
Caso N = 1000000
Clave = 7000000
Elemento buscado: 567234
Tiempo transcurrido: 893.051 milisegundos
----------------------
Caso N=10000000
Clave = 70000000
Elemento buscado: 7654389
```

```
Tiempo transcurrido: 12180.697 milisegundos
(12 segundos)
----------------------
```

Capítulo 8

Conclusiones

8.1. Acerca de Eficiencia

La Eficiecia de una Funcion de Hash depende de:

- La distribución de los valores de llave que realmente usan.
- El numero de valores de llave que realmente están en uso con respecto al tamaño del espacio de direcciones.
- El numero de registros que pueden almacenarse en una dirección dada sin causar una colisión.
- La técnica usada para resolver el problema de las colisiones.

8.2. Acerca de Tabla Hash

Hay que elegir correctamente la funcione de hash

- Debe ser facilmente calculable, sin costosas operaciones.
- Debe tener una buena distribucion de valores entre todas las componentes de la tabla.