Busqueda Mediante Hashing

Eclapajo Team

Claudio Acuña
Jose Acuña
Sebastian Esparza
Patricio Espinoza

Universidad tecnologica Metropolitana
Departameto Ingeneria
Escuela de Informatica

10 de Mayo 2013

Indice

Introduccion

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.

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.
Existen dos formas de Hashing distintas. El primero, llamado Hashing Cerrado, es una estructura de datos estática por lo cual usa un tamaño fijo para el almacenamiento, por lo que limita el tamaño de los conjuntos. El segundo, llamado Hashing Abierto, es una estructura de datos dinámica por lo cual no impone un límite al tamaño del conjunto.

Función de Hash

Una función de Hash es una caja negra que tiene como entrada una llave y como salida una dirección
$F(K) = andress$

Función de Hash

Una función de Hash es una caja negra que tiene como entrada una llave y como salida una dirección
$F(K) = andress$
Ejemplo: h(LOWELL)=4

Función de Hash

Una función de Hash es una caja negra que tiene como entrada una llave y como salida una dirección
$F(K) = andress$
Ejemplo: h(LOWELL)=4

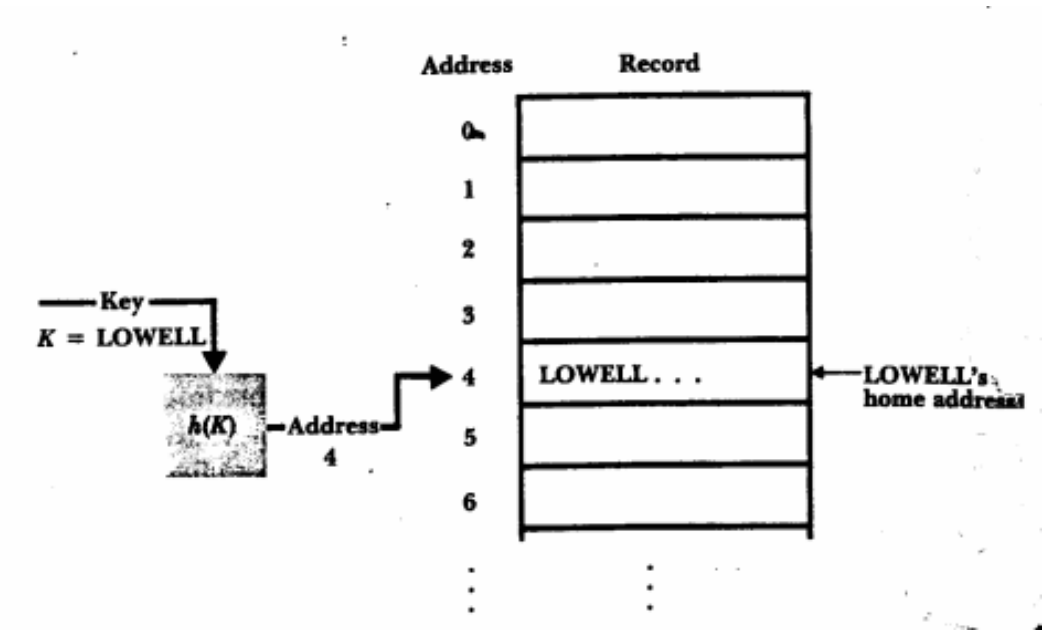

Figure:

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". Existen distintos grados de colisiones

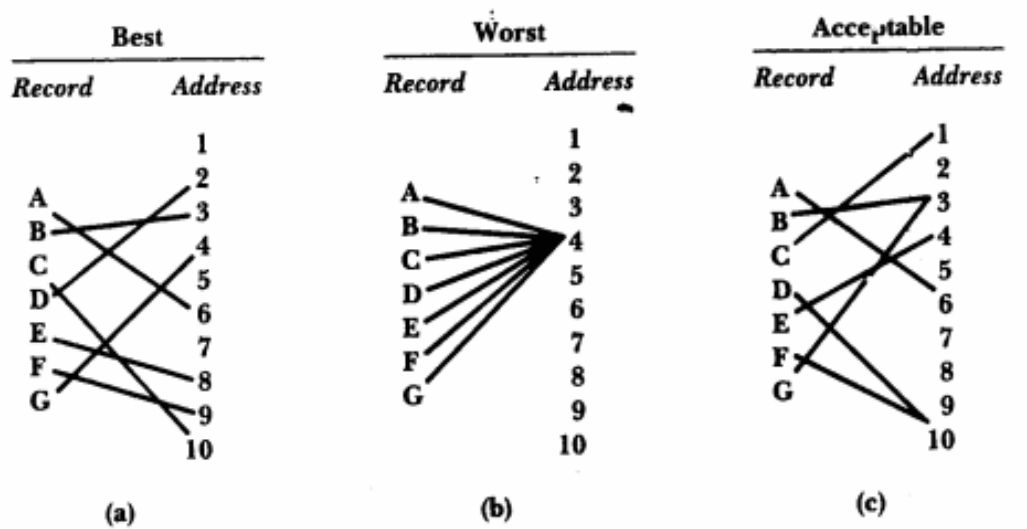

Figure:

Colisiones

En el ejemplo que mostramos en la sección anterior la función de Hash se basa en lo siguiente:

- Se obtiene el código ASCII de las 2 primeras letras de la llave
- Se multiplican dichos números
- La dirección serán los 3 últimos digitos de dicho resultado (esto para limitar nuestro archivo únicamente a 1000 direcciones)

LLave	ASCII	Producto	Dirección
BALL	66 65	66x65= 4290	290
LOWELL	76 79	76X79= 6004	004
TREE	84 82	84x82= 6888	888

A simple vista notamos que nuestra función de hash es pobre ya que fácilmente habrán llaves que generen la misma dirección ej (**OL**IVIER), a éstas llaves que generan las mismas direcciones se les llama "sinónimas".
Idealmente lo que se buscaría es tener un "algoritmo de hash perfecto" en el cual no ocurran colisiones y siempre nos garantice direcciones diferentes. Pero desafortunadamente esto es casi imposible, se dice que 1 de 10^120000 algoritmos evitarían dichas colisiones, así que hay que acostumbrarse a la idea de trabajar y lidiar con colisiones.
Para reducir el número de colisiones se tienen algunas soluciones:

A simple vista notamos que nuestra función de hash es pobre ya que fácilmente habrán llaves que generen la misma dirección ej (**OL**IVIER), a éstas llaves que generan las mismas direcciones se les llama "sinónimas".
Idealmente lo que se buscaría es tener un "algoritmo de hash perfecto" en el cual no ocurran colisiones y siempre nos garantice direcciones diferentes. Pero desafortunadamente esto es casi imposible, se dice que 1 de 10^120000 algoritmos evitarían dichas colisiones, así que hay que acostumbrarse a la idea de trabajar y lidiar con colisiones.
Para reducir el número de colisiones se tienen algunas soluciones:

- **Propagar los registros**: Buscar funciones que distribuyan muy aleatoriamente los registros podemos evitar "agrupaciones" de llaves que produzcan las mismas direcciones

A simple vista notamos que nuestra función de hash es pobre ya que fácilmente habrán llaves que generen la misma dirección ej (**OL**IVIER), a éstas llaves que generan las mismas direcciones se les llama "sinónimas".
Idealmente lo que se buscaría es tener un "algoritmo de hash perfecto" en el cual no ocurran colisiones y siempre nos garantice direcciones diferentes. Pero desafortunadamente esto es casi imposible, se dice que 1 de 10^120000 algoritmos evitarían dichas colisiones, así que hay que acostumbrarse a la idea de trabajar y lidiar con colisiones.
Para reducir el número de colisiones se tienen algunas soluciones:

- **Propagar los registros**: Buscar funciones que distribuyan muy aleatoriamente los registros podemos evitar "agrupaciones" de llaves que produzcan las mismas direcciones
- **Usar memoria extra**: En el ejemplo anterior planteamos tener una dirección de entre 1000 posibles, el uso de memoria extra se basa en proponer un espacio de direcciones posibles mucho más grande que el número de registros a usar, de modo que si vamos a insertar 100 registros un espacio de 500 direcciones nos una mejor opción de esparcir mejor.

A simple vista notamos que nuestra función de hash es pobre ya que fácilmente habrán llaves que generen la misma dirección ej (**OL**IVIER), a éstas llaves que generan las mismas direcciones se les llama "sinónimas".
Idealmente lo que se buscaría es tener un "algoritmo de hash perfecto" en el cual no ocurran colisiones y siempre nos garantice direcciones diferentes. Pero desafortunadamente esto es casi imposible, se dice que 1 de 10^120000 algoritmos evitarían dichas colisiones, así que hay que acostumbrarse a la idea de trabajar y lidiar con colisiones.
Para reducir el número de colisiones se tienen algunas soluciones:

- **Propagar los registros**: Buscar funciones que distribuyan muy aleatoriamente los registros podemos evitar "agrupaciones" de llaves que produzcan las mismas direcciones
- **Usar memoria extra**: En el ejemplo anterior planteamos tener una dirección de entre 1000 posibles, el uso de memoria extra se basa en proponer un espacio de direcciones posibles mucho más grande que el número de registros a usar, de modo que si vamos a insertar 100 registros un espacio de 500 direcciones nos una mejor opción de esparcir mejor.
- **Colocar más de un registro en una dirección**: A diferencia de los casos anteriores donde cada dirección almacena únicamente un registro, este concepto se basa en "buckets" o cubetas de datos en cada dirección, ahí se colocan algunos (casi todos) los registros que colisionan de manera que al hacer una búsqueda debemos recuperar la cubeta entera y ahi buscar por el registro deseado.

Hash Cerrado

Existen distintos tipos de Hashing cerrados. Los más comunes son el lineal, el doble y el directo, y se diferencian unos de otros por la forma de resolver las colisiones:

Hash Cerrado

Existen distintos tipos de Hashing cerrados. Los más comunes son el lineal, el doble y el directo, y se diferencian unos de otros por la forma de resolver las colisiones:

- **El Hashing lineal** resuelve el problema de las colisiones asignando el primer lugar disponible recorriendo circularmente la tabla. Cuando elimina recorre toda la tabla para asegurarse de que lo eliminó. Sufre de agrupamiento primario, dado que cuando existen colisiones la tabla crece por su corrimiento y luego cualquier inserción tendrá colisión.

Hash Cerrado

Existen distintos tipos de Hashing cerrados. Los más comunes son el lineal, el doble y el directo, y se diferencian unos de otros por la forma de resolver las colisiones:

- **El Hashing lineal** resuelve el problema de las colisiones asignando el primer lugar disponible recorriendo circularmente la tabla. Cuando elimina recorre toda la tabla para asegurarse de que lo eliminó. Sufre de agrupamiento primario, dado que cuando existen colisiones la tabla crece por su corrimiento y luego cualquier inserción tendrá colisión.

- **El Hashing doble** aplica una segunda función Hashing cuando se produce una colisión, para determinar el incremento con el cual buscar la próxima posición. Esta alternativa requiere tablas con números primos de elementos para asegurar el recorrido de todos los lugares posibles.

Hash Cerrado

Existen distintos tipos de Hashing cerrados. Los más comunes son el lineal, el doble y el directo, y se diferencian unos de otros por la forma de resolver las colisiones:

- **El Hashing lineal** resuelve el problema de las colisiones asignando el primer lugar disponible recorriendo circularmente la tabla. Cuando elimina recorre toda la tabla para asegurarse de que lo eliminó. Sufre de agrupamiento primario, dado que cuando existen colisiones la tabla crece por su corrimiento y luego cualquier inserción tendrá colisión.

- **El Hashing doble** aplica una segunda función Hashing cuando se produce una colisión, para determinar el incremento con el cual buscar la próxima posición. Esta alternativa requiere tablas con números primos de elementos para asegurar el recorrido de todos los lugares posibles.

- **El Hashing directo** trabaja con un almacén auxiliar, también llamado OVERFLOW, en el cual se almacenan las colisiones.

Hash Cerrado

Existen distintos tipos de Hashing cerrados. Los más comunes son el lineal, el doble y el directo, y se diferencian unos de otros por la forma de resolver las colisiones:

- **El Hashing lineal** resuelve el problema de las colisiones asignando el primer lugar disponible recorriendo circularmente la tabla. Cuando elimina recorre toda la tabla para asegurarse de que lo eliminó. Sufre de agrupamiento primario, dado que cuando existen colisiones la tabla crece por su corrimiento y luego cualquier inserción tendrá colisión.

- **El Hashing doble** aplica una segunda función Hashing cuando se produce una colisión, para determinar el incremento con el cual buscar la próxima posición. Esta alternativa requiere tablas con números primos de elementos para asegurar el recorrido de todos los lugares posibles.

- **El Hashing directo** trabaja con un almacén auxiliar, también llamado OVERFLOW, en el cual se almacenan las colisiones.

Hash Abierto

El Hashing abierto maneja las colisiones generando una lista enlazada en cada una de las posiciones de la tabla, es decir, si dos valores poseen un mismo valor para Hashing estarán dentro de una misma lista enlazada.

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, lass complejidad de las operaciones serán de orden O(B) siendo B el tamaño de la tabla.
- En el caso de la búsqueda implementada en el ejemplo de hash abierto el algoritmo de búsqueda es el siguiente.

```
pcelda buscar(char *s)
{
pcelda cp; /* current pointer */
for (cp = hashtabla[h(s)]; cp!= NULL; cp = cp->next)
if (strcmp(s, cp->nombre ) == 0)
return (cp); /* lo encontró */
return (NULL);
}
```

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

Algoritmos de Hash Mas Comunes

- SHA-1:**(SECURE HASH ALGORITHM,ALGORITMO DE HASH SEGURO)** Algoritmo de síntesis que genera un hash de 160 bits. Se utiliza, por ejemplo, como algoritmo para la firma digital

Algoritmos de Hash Mas Comunes

- SHA-1:**(SECURE HASH ALGORITHM,ALGORITMO DE HASH SEGURO)** Algoritmo de síntesis que genera un hash de 160 bits. Se utiliza, por ejemplo, como algoritmo para la firma digital
- MD2 y MD4: **(Message-Disgest Algorithm, Algoritmo de Resumen del Mensaje)**Algoritmos de hash que generan un valor de 128 bits.

Algoritmos de Hash Mas Comunes

- SHA-1:**(SECURE HASH ALGORITHM,ALGORITMO DE HASH SEGURO)** Algoritmo de síntesis que genera un hash de 160 bits. Se utiliza, por ejemplo, como algoritmo para la firma digital
- MD2 y MD4: **(Message-Disgest Algorithm, Algoritmo de Resumen del Mensaje)**Algoritmos de hash que generan un valor de 128 bits.
- 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 traves de la red.

Ejemplos de Funciones Hash

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

Ejemplos de Funciones Hash

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

- Centro del Cuadrado
  H(123)=123^2=15129;Dirección:51
  H(730)=730^2=532900;Dirección:29

Ejemplos de Funciones Hash

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

- Centro del Cuadrado
  H(123)=123^2=1**51**29;Dirección:51
  H(730)=730^2=53**29**00;Dirección:29

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

Ventajas de la Busqueda Por Hashing

- 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 direccioes
- No se requiere almacenamiento adicional para los índices

Desventajas de la Búsqueda por Hashing

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

La eficencia de una Función 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