TRANSFORMACIONES DE LLAVES (Hashing)

Se trata del problema de intentar encontrar un elemento de entre un conjunto ordenado de elementos . Cuando el número de elementos del conjunto ordenado es muy grande , el tiempo empleado para buscarlo puede hacerse demasiado largo , y para esto surge la idea de situar cada elemento de forma que si sabemos cuál es , esto nos lleve muy rápidamente a encontrarlo de entre el conjunto de todos los elementos . Esto tiene su utilidad cuando los citados elementos son registros con varios campos y lo que queremos es encontrar es el registro completo que en uno de sus campos contiene un cierto valor (ya sea numérico o de cadena de letras) .

El problema sería muy sencillo si por ejemplo tenemos elementos que sólo pueden constar de una letra , sólo tendríamos que poner la letra "a" en el lugar 1 de un vector , la "b" en el lugar 2 y así sucesivamente hasta la "z" que estaría situada en el lugar 26 . De esta forma , si queremos buscar la letra "j" , sabemos que sólo puede estar en la casilla 10 del vector . Pero imaginemos que en vez de una letra , cada elemento contiene dos letras ; en este caso , la "aa" la pondríamos en la celda 1 del vector , la "ab" en la celda 2 , la "ac" en la celda 3 y así sucesivamente hasta la "zz" que estaría situada en la celda 26x26=676 del vector ; en este caso , serían necesarias 676 celdas de vector para poder obtener de una forma rápida la localización de cualquier cadena ; por ejemplo , supongamos que queremos encontrar la celda que contiene "cj" , para ello sólo tenemos que multiplicar el lugar que corresponde a la "c" (de entre todo el abecedario , que sería el lugar 3º) por el lugar que corresponde a la "j" (de entre todo el abecedario , que sería el 10º) , es decir 3x10=30 , la "cj" estaría con toda seguridad en la celda 30 del vector .

Ahora vamos a suponer , que se admiten elementos con 10 letras posibles , el número de casillas del vector necesarias para poder resolver la situación de una forma unívoca sería 26x26x26x26x26x26x26x26x26x26=26e10 , algo imposible de contener en cualquier ordenador .

Por lo tanto , la función que asigna el lugar para un elemento cualquiera (llave) de entre todos los posibles H:K ===> A , es una función de muchos probables a pocos posibles (porque las memorias son muy limitadas).

Por lo tanto , esta función debe de no ser unívoca sino que debe de asignar más de un posible elemento para cada celda . Por ejemplo , en el tema de asignar palabras de 2 letras , se encontró que serían necesarias 26x26=676 celdas , pero si tenemos por convenio que todas las palabras que empiecen por "a" vayan a la celda 1 , las que empiecen por "b" a la 2 ,etc... sólo en el caso de que en la base de datos hubiera más de una palabra que empezara por "a" ,encontraríamos un conflicto llamado colisión y tendríamos que volver a redistribuir del misma forma todas las que empezaran por"a" . Con el resto se haría lo mismo .

Es decir , es necesario por una parte encontrar una buena función H y por otra parte una buena función H' que nos redireccionara en caso de conflicto en la primera localización .

El requisito básico de una buena función H es que distribuya los elementos de una forma lo más uniforme posible para que se de un mínimo de colisiones . Una función muy usada (la que hemos utilizado en los ejemplos) es la función ORD (k) , que nos devuelve el lugar del letra de entre todo el abecedario . Por lo tanto , si tenemos un vector con N celdas , la función más sencilla de direccionamiento sería H(k)=(ORD (k)) MOD N .

Pero si los elementos utilizados son palabras , como todas las letras no tienen la misma probabilidad de salir , hay que elegir un N que sea primo para que la redistribución sea uniforme lo más posible .

Una de las formas de tratar las colisiones (una colisión se produce cuando 2 o más elementos deben de ser localizados en la misma celda del vector) es utilizando una lista de punteros de

forma que cada vez que se archive un elemento que deba corresponder a esa celda , se añada a la lista . Cuando se desee buscar algún elemento al que la función H nos señale , lo buscaremos de entre toda la lista de los elementos asignables a dicha celda del vector y si recorremos toda la lista sin encontrarlo , se devolverá que no está presente (a este método de tratar las colisiones se le llama *encadenamiento directo*) .

También hay otra forma de guardar los elementos que tengan asignada la misma celda , estos elementos se puede ir guardando en una tabla que se llama *área de desbordamiento* , este método se llama de *dirección abierta* .
Otro método muy eficaz para resolver las colisiones es utilizar una segunda función para redistribuir los elementos en colisión en el mismo vector original .

Esta función puede ser

h0=H(k)
hi=(h0 + i) MOD N

A este método se le llama de *exploración lineal* y tiene la desventaja de que las entradas tienden a agruparse alrededor de las llaves que no han tenido ninguna colisión .

Hay otro método que aunque requiere más cálculos , no tiene la desventaja de la agrupación alrededor de las llaves no colisionadas . Se trata del método de la *exploración cuadrática* , que consiste en aplicar la siguiente función

h0=H(k)
hi=(h0 + (i x i)) MOD N

Este segundo método tiene la ventaja de que no agolpa los elementos alrededor de los no colisionados pero tiene el inconveniente de que se dejan algunos huecos vacíos en los que no es posible guardar elementos en colisión .

****

Un esquema sería :

- ¿Por qué utilizar las transformaciones de llaves ?
- Elección de una función H primaria : por ejemplo H(k)=ORD (k) MOD N
- Elección de un N primo para evitar agrupamientos
- Tratar las colisiones
- Métodos para tratar colisiones :
    1 . Encadenamiento directo (listas enlazadas)
    2 . Dirección abierta (mirando en una tabla adicional)
    3 . Redireccionamiento :
        A . Exploración lineal (agrupamiento)
        B . Exploración cuadrática (ranuras vacías)