Análisis y Algoritmos

Sergio Juan Diaz Carmona
Universidad De Artes Digitales

Guadalajara, Jalisco

Email: idv17c.sdiaz@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Junio 06, 2019

1) Ordenamientos
   En computación y matemáticas un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación (o reordenamiento) de la entrada que satisfaga la relación de orden dada. Las relaciones de orden más usadas son el orden numérico y el orden lexicográfico. Ordenamientos eficientes son importantes para optimizar el uso de otros algoritmos (como los de búsqueda y fusión) que requieren listas ordenadas para una ejecución rápida. También es útil para poner datos en forma canónica y para generar resultados legibles por humanos.

I. Tabla Hash

Una tabla hash, matriz asociativa, hashing, mapa hash, tabla de dispersión o tabla fragmentada es una estructura de datos que asocia llaves o claves con valores. La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos (teléfono y dirección, por ejemplo) almacenados a partir de una clave generada (usando el nombre o número de cuenta, por ejemplo). Funciona transformando la clave con una función hash en un hash, un número que identifica la posición (casilla o cubeta) donde la tabla hash localiza el valor deseado.

Las tablas hash se suelen implementar sobre vectores de una dimensión, aunque se pueden hacer implementaciones multi-dimensionales basadas en varias claves. Como en el caso de los arrays, las tablas hash proveen tiempo constante de búsqueda promedio O(1),1 sin importar el número de elementos en la tabla. Sin embargo, en casos particularmente malos el tiempo de búsqueda puede llegar a O(n), es decir, en función del número de elementos.

Comparada con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información.

Las tablas hash almacenan la información en posiciones pseudo-aleatorias, así que el acceso ordenado a su contenido es bastante lento. Otras estructuras como árboles binarios auto-balanceables tienen un tiempo promedio de búsqueda mayor (tiempo de búsqueda O(log n)), pero la información está ordenada en todo momento.

En mi hash table yo genero el id utilizando el numero de entrada y una constante, y despues saco el modulo del resultado para saber en que parte de la tabla ponerlo

```
int hasha(int key, int consta)
{
        int cons=consta;
        key = (key ^ cons) ^ (key >> cons);
        key = key * cons;
        int numtab=key% consta;
        hashTable[numtab].push_back(key);
        return key;
}
```