Análisis y Algoritmos

Sergio Juan Díaz Carmona
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.sdiaz@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Julio 04, 2019

1) Hash Table

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), 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.

Codigo:

Multiplicacion:

```
int hashMul(int key, int consta)
{
        float cons = 0.5;
        key = (key * cons);
        int numtab = key % consta;
        hashTable[numtab].push_back(value);
        return key;
}
```

Division:

```
int hashDiv(int key, int consta)
{
        int cons = consta;
        key = (key ^ cons) ^ (key >> cons);
        key = key / cons;
        int numtab = key % consta;
        hashTable2[numtab].push_back(key);
        return key;
}
```

Universal:

```
int hashUn(int key,int value, int consta, int numkeys)
```

```
{
        float constante = consta;
        float a = (1 / constante);
        int genKey = a * key;
        int newKey = genKey % consta;
        if (newKey> consta-1)
        {
                cout << "checa o: " << newKey << "\n";
                return 0;
        }
        hashTable3[newKey].push_back(value);
        return 0;
}
```

El resultadon con 100 numeros utilizando 7 buckets:

```
Multiplicacion:
1,2,15,16,29,30,43,44,57,58,71,72,85,86,99,100,

3,4,17,18,31,32,45,46,59,60,73,74,87,88,

5,6,19,20,33,34,47,48,61,62,75,76,89,90,

7,8,21,22,35,36,49,50,63,64,77,78,91,92,

9,10,23,24,37,38,51,52,65,66,79,80,93,94,

11,12,25,26,39,40,53,54,67,68,81,82,95,96,

13,14,27,28,41,42,55,56,69,70,83,84,97,98,

Division:
2,3,4,5,6,7,8,49,50,51,52,53,54,55,97,98,99,100,

1,11,12,13,14,15,16,58,59,60,61,62,63,64,

9,10,20,21,22,23,24,57,67,68,69,70,71,72,

17,18,19,29,30,31,32,65,66,76,77,78,79,80,

25,26,27,28,38,39,40,73,74,75,85,86,87,88,

33,34,35,36,37,47,48,81,82,83,84,94,95,96,

41,42,43,44,45,46,56,89,90,91,92,93,

Universal:
1,2,3,4,5,6,7,50,51,52,53,54,55,56,99,100,

8,9,10,11,12,13,14,57,58,59,60,61,62,63,

15,16,17,18,19,20,21,64,65,66,67,68,69,70,

22,23,24,25,26,27,28,71,72,73,74,75,76,77,

29,30,31,32,33,34,35,78,79,80,81,82,83,84,

36,37,38,39,40,41,42,85,86,87,88,89,90,91,

43,44,45,46,47,48,49,92,93,94,95,96,97,98,
```