Hashing

Look-up tables

Una tabla de look-up es un diccionario con velocidad de acceso muy eficiente. Suele ser $O(1)$. Se implementan con un arreglo en el que la clave del diccionario se usa como indice al arreglo. Esta eficiencia la tienen tanto para lectura como para escritura.

Esta eficiencia se logra calculando el indice en el cual colocar un dato a partir de su clave. Entonces tengo una forma de corresponder una clave con un indice. El rendimiento en memoria de una look-up table, entonces, sera mejor para tablas mas densas.

Hash Tables

Para mejorar este problema de la eficiencia de memoria de las tablas look-up se hicieron las tablas hash. En una tabla hash, la clave se convierte mediante una **funcion hash**, que devuelve un **valor hash**, que es el indice en el arreglo de la tabla.

En una tabla hash, las entradas no estan ordenadas. Esto quiere decir que no se puede hacer una busqueda dentro de un rango de la manera que se podria en las tablas look-up comunes.

La ventaja de las tablas hash es que permiten mapear entradas mas dispersas a un espacio denso, para mejorar la eficiencia en memoria.

Cosas que las tablas hash deberian hacer:

- producir, para cada clave, un valor deterministico
  - si no sucede abiertaesto entonces una clave puede coresponder a mas de una entrada. Esto es desaastroso
- aprovechar todos los bits
- los valores resultantes deberian estar distribuidos uniformemente en todas las posibles entradas de la tabla hash
  - no queremos que valores se concentren en una region de la tabla hash
- produzcan valores hash muy diferentes para entradas muy similares
- ser eficientes en calculo

Funciones Hash

Estas hay que tomarlas como si fueran un camino solamente de ida. Este es el motivo por el cual funcionan. Una funcion hash tiene como objetivo tomar una clave y devolver un valor hash. Sirven para seguridad y autenticacion porque es practicamente imposible partir del valor hash y obtener la clave. Segun el numero de bits que tenga la funcion hash, se puede tardar mas o menos en hallar la clave. Una funcion hash de 256 bits devuelve hasta 2^256 valores hash.

Ejemplos de funciones hash:

- **Folding**: esta separa la clave en secciones de N bits y luego las suma o combina con `xOR`
- **Mid-squares**: esta eleva la clave al cuadrado y devuelve los N bits centrales de ese resultado (esta no da los mejores resultados)
- **Multiplicativa**: `h(n) = w * n`, donde `w` es un coprimo del tamano de la tabla hash.
- **Criptograficas**: MD5, SHA-1, SHA-2, SHA-3

Puede suceder que dos claves distintas devuelvan hashes iguales. Esto se conoce como una colision. Para esto se escriben **politicas de resolucion de colisiones**.

Una politica **abierta** inserta una segunda estuctura dentro de la tabla. Cada entrada seria una lista o un AVL. Entonces varias entradas pueden vivir con distintas claves y valores en el mismo indice en la tabla. Esto permite crear una nueva funcion hash a partir de la anterior. `h2(n) = h(n+K)`.

Una politica **cerrada** guarda el valor en la siguiente posicion libre en la memoria. Es menos usado.

Factor de Carga

Es la relacion entre la cantidad de entradas ocupadas en la tabla hash y el tamano de la tabla hash. Las tablas hash suelen ser optimas cuando el factor de carga esta entre 0,6 y 0,75. SI es superior, tendremos muchas colisiones y si es menor se esta desperdiciando mucha memoria. Cuando nos estamos yendo del rango de valores eficientes, es conveniente redimensionar dinamicamente la tabla hash.

El problema con esto es que van a cambiar los indices de la tabla. Para subsanar este efecto es que se lleva a cabo el proceso de **rehashiing**. Este consiste en volver a calcular los valores hash e indices para todas las claves previamente existentes. Puede ser un proceso largo ya que no hay mas remedio que iterar sobre todos los elementos existentes en la tabla hash original.

En C++

En STL, las tablas hash estan implementadas en `unordered_set`, `unordered_map`, `unordered_multiset` y `unordered_multimap`.