Tablas de Hash

Complejidad promedio

* Insertar O (1) (constante)
* Borrar O(1)
* Buscar O(1)

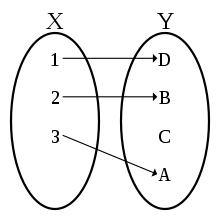
La complejidad en el peor de los casos con funciones de hash no inyectivas (no perfectas) es O(n)

Una estructura para almacenar con acceso directo. La tabla como un arreglo.

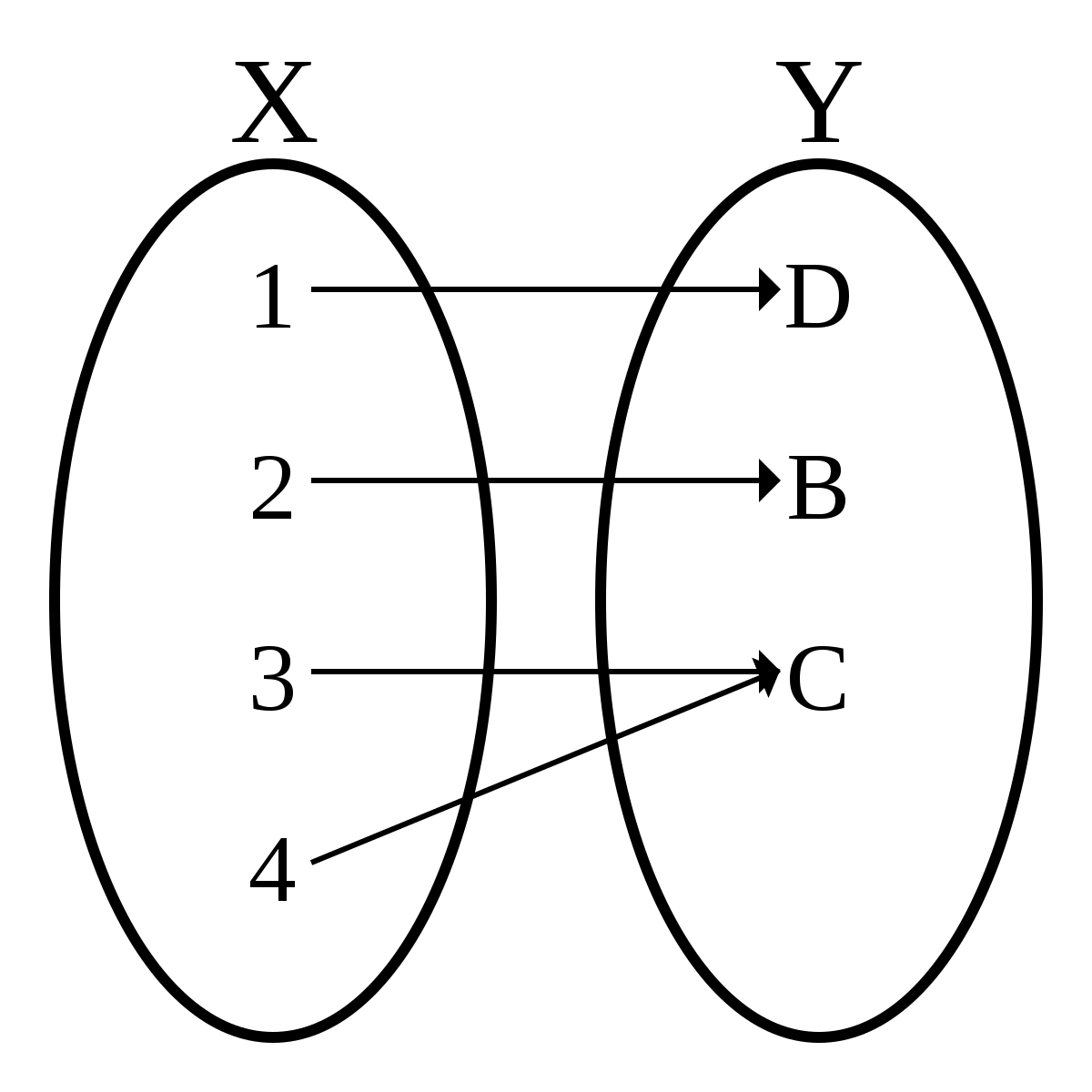
Necesitamos una función que tome un dato de cualquier tipo y devuelva un entero: la función de Hash.

Deseamos evitar colisiones. Para lograr que la tabla de hash sea eficiente queremos que el número de colisiones por casilla sea O(1).

* Hay funciones inyectivas: funciones Hash perfectas. Relación uno a uno, no hay colisiones.

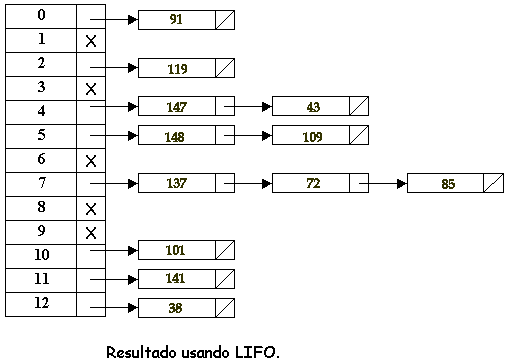


* Hay funciones sobreyectivas en donde más de un dato puede ser mapeado a una misma posición.



Dos estrategias para evitar colisiones.

* Si la posición está ocupada, buscamos la siguiente posición disponible. Poco eficiente cuando las colisiones crecen muy rápido.
* Encadenamiento. Dentro de la posición dada por la función de hash, tenemos otra estructura de datos a la que le insertamos los datos con el mismo entero dado por la función. El entero nos diría en cuál de nuestras estructuras de datos debemos de buscar. Las estructuras dentro de la tabla de hash podrían ser listas enlazadas, árboles, etc.



Características de la función de hash

* La función debe distribuir uniformemente los datos del dominio
* Minimizar colisiones
* Que el rango sea del tamaño de la tabla
* Determinista y que dependa del dato
* Que sean fáciles, rápidos de calcular O(1)

función hash md5: Recibe un dato y regresa un número en hexadecimal, es una función perfecta, no hay colisiones. Por ejemplo, con un archivo, toma sus contenidos y regresa un entero. El problema es que se tarda mucho en generar el entero.

Los diccionarios de Python son tablas de Hash. Pares de llaves y valores.

Tablas de Hash con encadenamiento

Básicamente, tenemos un arreglo de estructuras de datos. Tenemos que expandir el arreglo al doble cuando se quede sin espacio.

La clase tiene un arreglo de listas enlazadas, contador, funciones para insertar, eliminar, buscar y una función de hash. El arreglo sería un arreglo de NodoHash<T>, podría ser un nodo centinela o ahorrar ese espacio para insertar el primer elemento de la lista.

La función de Hash nos regresará un número muy grande, para poder acotarlo a las posiciones de la tabla usamos módulo (%) con el tamaño de la lista el cual nos regresa un número menor al tamaño de la lista. Al encontrar la posición en la que debemos insertar, colocamos el nodo nuevo al frente de la lista.

El expande no sólo es copiar las listas, ya que al duplicar el tamaño del arreglo, algunos elementos cambiarán de posición. Por lo tanto, para expandir, debemos de volver a insertar los elementos de la lista original.

Bloom filters

* Insertar(1), buscar(1), no hay eliminar.
* Sirve para comprobar si un dato forma parte de un grupo.
* Tiene falsos positivos, no tiene falsos negativos. El falso positivo nos dice que sí insertamos el dato, pero en realidad no lo insertamos.
* Es una estructura de datos probabilística.
* Los bloom filters son un arreglo de booleanos, al insertar un elemento en una posición dada por la función de hash la posición en el arreglo la ponemos como true.