FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TDA Hash.

[7541/9515] Algoritmos y Programación II

Primer cuatrimestre de 2022

Alumno: Martín Abramovich
Padrón: 108762
Email: mabramovich@fi.uba.ar

Contenido

1. Introducción

Se solicita implementar una tabla de hash abierta (direccionamiento cerrado) en C. Para ello se cuenta con las firmas de las funciones públicas a implementar.
Se implementa además una metodología orientada a pruebas, en que se solicita la creación de pruebas pertinentes para cada una de las diferentes primitivas del TDA.

2. Conceptos teóricos: tabla de hash abierta y cerrada

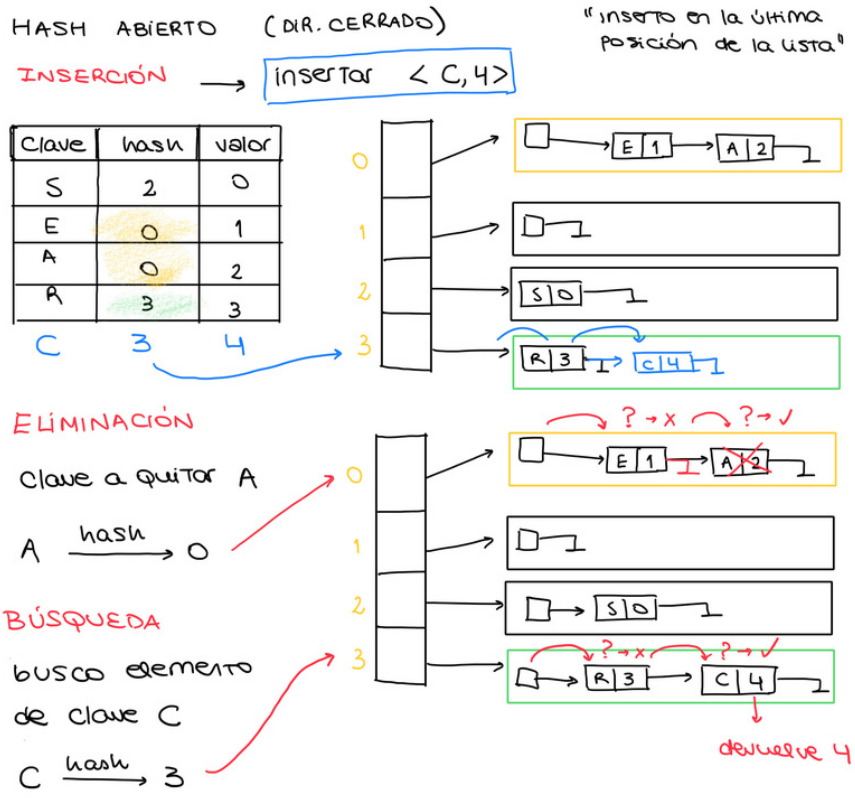
Una tabla de hash es una estructura de datos que asocia claves con valores. La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos almacenados a partir de una clave generada. Funciona transformando la clave con una función hash en un hash, un número que identifica la posición donde la tabla hash localiza el valor deseado.

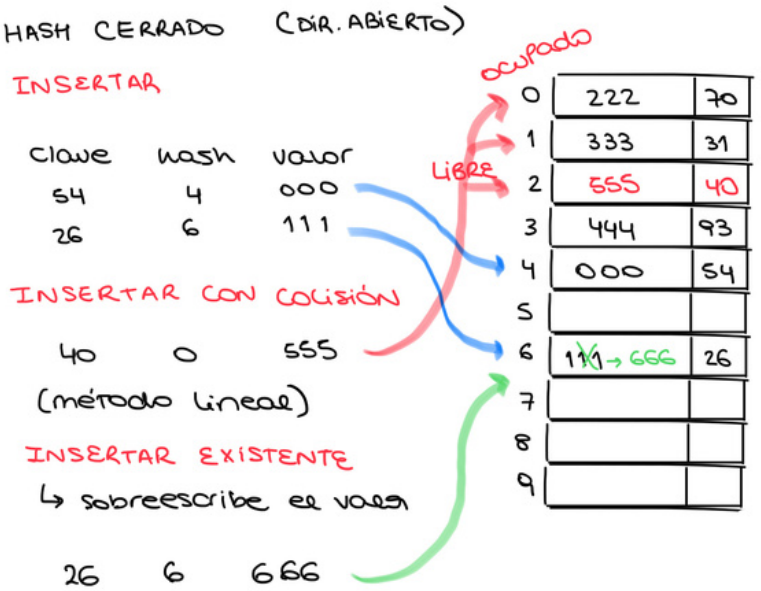
Cuando claves distintas dan el mismo valor de "hash" se produce una colisión. En estos casos, debemos encontrar otra ubicación donde almacenar el nuevo registro, y hacerlo de tal manera que podamos encontrarlo cuando se requiera.

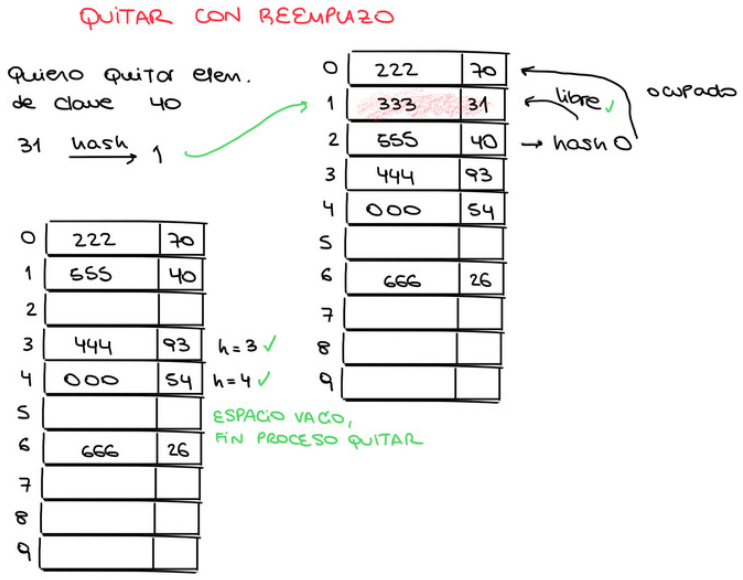
Un hash puede ser abierto (con direccionamiento cerrado) o cerrado (con direccionamiento abierto).

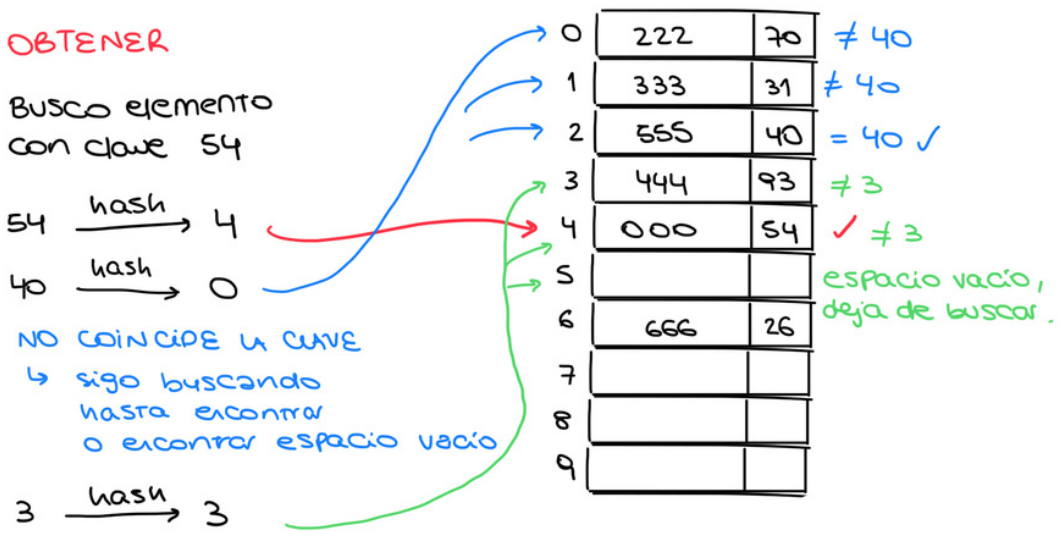
En un hash abierto los elementos se almacenan fuera de la tabla. Cada casilla en la tabla de hash referencia a una lista con los registros insertados que colisionan en dicha casilla. La inserción consiste en encontrar la casilla correcta y agregar al final de la lista correspondiente. El borrado consiste en buscar y quitar de la lista. La complejidad de encontrar una clave con su correspondiente valor en caso de colisión es O(n), donde n es la cantidad de elementos que colisionaron. Es decir, hay que recorrer la lista enlazada.

En un hash cerrado (direccionamiento abierto) todos los valores se guardan dentro de la misma tabla. Aquí el tamaño de la tabla es mayor o igual al número de claves.
En caso de colisión, se sigue recorriendo el array hasta encontrar el próximo espacio libre. Entre los tipos de métodos de búsqueda se encuentran el probing lineal (buscar el siguiente espacio libre inmediato), el probing cuadrático ((intentos fallidos) ^ 2 para intentar insertar) y el hash doble (aplicar una segunda función de hash a la clave cuando hay colisión).


HASH ABIERTO (DIR. CERRADO)
"inserto en la última posición de la lista"
INSERCIÓN
insertar <C,4>
Clave | hash | valor
S | 2 | 0
E | 0 | 1
A | 0 | 2
R | 3 | 3
C 3 4
ELIMINACIÓN
Clave a quitar A
A hash 0
BÚSQUEDA
busco elemento de clave C
C hash 3
devuelve 4


HASH CERRADO (DIR. ABIERTO)
INSERTAR
Clave hash valor
54 4 000
26 6 111
INSERTAR CON COLISIÓN
40 0 555
(método lineal)
INSERTAR EXISTENTE
↳ sobreescribe el valor
26 6 666
ocupado
libre


QUITAR CON REEMPLAZO
Quiero quitar elem. de clave 40
31 hash 1
ocupado
libre ✓
→ hash 0
h=3 ✓
h=4 ✓
ESPACIO VACÍO, FIN PROCESO QUITAR


OBTENER
Busco elemento con clave 54
54 hash 4
40 hash 0
NO COINCIDE LA CLAVE
↳ sigo buscando hasta encontrar o encontrar espacio vacío
3 hash 3
≠ 40
≠ 40
= 40 ✓
≠ 3
✓ ≠ 3
espacio vacío, deja de buscar.

3. Implementación y decisiones de diseño

Decidí implementar mi hash abierto utilizando el tda lista que implementé para la materia, ya que sabía que los elementos iban a ser almacenados en listas.

Mi struct hash tiene un size_t cantidad (los elementos almacenados), un size_t capacidad (cantidad de posiciones de la tabla de hash) y una lista_t **tabla. Además cree un struct par_t con una clave y un valor.

Al crear el hash, me aseguro de que la capacidad inicial sea la recibida por parámetro y, de ser menor a la capacidad mínima, que sea la capacidad mínima.
Además inicializo la cantidad del hash en 0 y creo las listas para cada posición de la tabla de hash.

Elegí como factor de carga 0.75, es decir, cuando el factor de carga es mayor o igual a 0.75, rehasheo.
Mi rehash duplica la capacidad de la tabla y va sacando los elementos de las listas de la tabla original. A medida que los saca, calcula su nueva posición y los inserta con esa nueva posición en una nueva tabla llamada tabla_rehash.
Una vez liberada la tabla hash original y todos los elementos insertados con sus nuevas posiciones en la tabla de rehash, le asigno a la tabla del hash la tabla rehasheada.

Para insertar, recorro la lista correspondiente a la posición retornada en la función hash, en caso de encontrar un elemento con la misma clave, sobreescribe el elemento. Caso contrario, inserta el par <clave, valor> en la última posición de la lista. Al terminar de insertar verifico el factor de carga para ver si es hora de rehashear.

Para quitar del hash, recorro la lista correspondiente a la posición retornada por la función hash. En caso de encontrar un elemento con dicha clave, saco el elemento con lista quitar de posicion en la posicion en que lo encontró y retorno el valor del par quitado. Si no encuentra un elemento con la clave dada, retorna NULL.

En hash_obtener, para buscar un elemento, utilizo lista buscar elemento en la lista correspondiente a la posición que retorna la función hash. Si lo encuentra devuelve el valor del par, y caso contrario retorno NULL.

Para destruir todo recorro todas las listas que tiene el hash y voy quitando los elementos hasta que la lista esté vacía. Luego libero la lista, y finalmente la tabla y el hash.

En las pruebas accedí al struct hash para poder probar que la capacidad se inicializaba de manera correcta.