TDA 3 - Hash

[7541/9515] Algoritmos y Programación II

Segundo Cuatrimestre - 2021

Alumno:	**González Calderón, Julián**
Padrón:	107938
Email:	gonzalezcalderonjulian@gmail.com

1. Introducción

La idea general del trabajo practico era implementar el hash abierto. Este consta de una tabla, donde cada elemento de la tabla es un tipo de dato externo al hash que se utiliza para almacenar los elementos que corresponden a ese mismo índice.

Debíamos implementar una función hash que convierta una clave en un valor pseudo-único. Luego utilizaríamos este valor para conseguir un índice valido de la tabla.

Para seguir con los requisitos del trabajo practico, teníamos un ***contrato*** que explicaba las funciones que debíamos diseñar con sus debidas especificaciones.

Además de implementar el hash, nosotros debemos diseñar nuestras propias pruebas para verificar el correcto funcionamiento del mismo.

2. Teoría

¿Que es un hash?

Un hash es una estructura que contiene valores, puedo acceder a estos valores a partir de una clave. La función que convierte una clave en un índice de la tabla se denomina *Función Hash***.**

Esta función puede tener colisiones, es decir. Distintas claves que redireccionan al mismo índice.

¿Que tipos de hash existen?

Existen dos tipos de hash, el *hash abierto* y el *hash cerrado*.

¿En que se diferencian?

Un *hash abierto*, o de direccionamiento cerrado. Es aquel en el que el elemento se encuentra obligatoriamente en el índice de su clave.

Se denomina abierto porque depende de una estructura externa al hash. Cada posición del hash apunta a una estructura que contiene todos los valores que direccionan a esa posición.

Se utiliza una lista con nodos (cada elemento de la lista apunta al siguiente), por lo que se denomina *Chaining* o *Encadenamiento*.

En un *hash cerrado*, o de direccionamiento abierto, todos los valores se guardan en la misma tabla. Esto implica que, cuando hay una colisión, se debe redireccionar la clave hacia el índice siguiente.

Se denomina *cerrado* porque no depende de una estructura externa.

3. Detalles de implementación

Algunas funciones principales que merecen ser explicadas detalladamente

1. `hash_crear()`

   Para crear el hash, la función reserva memoria para el hash en si, y su tabla (una vector de listas. Luego inicializa cada elemento de la tabla con *lista_crear()*.

   Esta función podría no crear las listas, verificando si la lista existe a la hora de operar con el hash. Sin embargo, considero que esto es innecesario ya que la mayoría de las listas se va a crear eventualmente.

   Debido a esto, las listas se crean inicialmente y luego se opera con ellas sin necesidad de verificarlas.

2. `hash_insertar()`

   Para insertar elementos, primero busco en que índice de la tabla se encontraría la clave, luego verifica que esa clave no exista ya en la tabla.

   Si la clave existe, entonces accede a su valor y lo modifica. Si la clave no existe, entonces simplemente lo agrega al final de la lista.

Antes de insertar un elemento nuevo, Aplica un *rehash* de ser necesario.

3. `hash_quitar()`

   El algoritmo de quitar elementos es similar al anterior, primero busco en que indice de la tabla se encontraría la clave, luego busco en que posición de la lista esta la clave.

   Finalmente elimino ese elemento de la lista, destruyendo el elemento y liberando la memoria necesaria.

4. `hash_con_cada_clave()`

   Accede a cada lista de la tabla del hash, y luego utiliza un iterador interno para acceder a cada clave y aplicar la función pasada por parámetro.

A continuación, el funcionamiento de algunas funciones auxiliares.

1. `indice_de_clave_en_lista()`

   Esta función devuelve en que posición de la lista se encuentra el elemento con la clave especificada.

   Para esto, utiliza *lista_con_cada_elemento()*. Llama con cada elemento de la lista una función que compara la clave a buscar con la clave actual. En el momento que lo encuentra, corta la iteración. De esta forma, según cuantas veces se aplica la función comparador, podemos averiguar en que posición de la lista se encuentra la clave buscada.

2. `destruir_tabla()`

   Para destruir la tabla, accede a cada lista de la misma y utiliza un iterador interno para acceder a cada elemento de la lista. Luego destruye cada par *[calve, valor].*

3. `rehash()`

   Para aplicar un rehash, entonces utiliza *hash_con_cada_elemento() para acceder a cada* par *[clave, valor] del hash, e insertarlo en un nuevo hash.*

   Para esto, recibe una función que dado el antiguo hash, la clave, y el nuevo hash. Accede al elemento del hash antiguo y lo inserta en el nuevo hash.

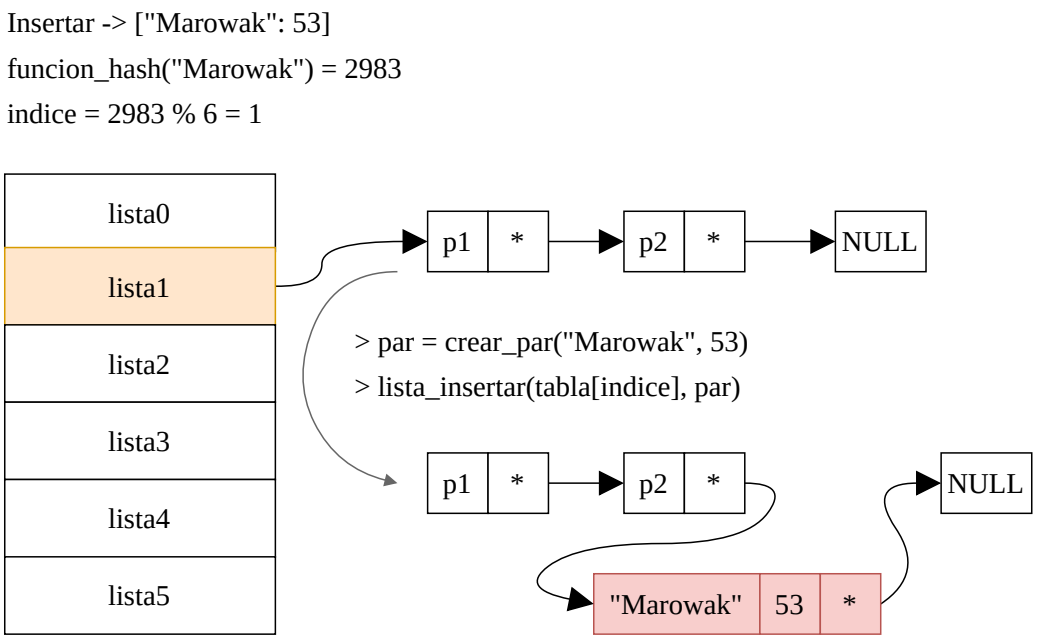
   Una vez hecho esto, libera el hash anterior o el hash nuevo, dependiendo de si hubo un error en la operación o no.

4. Diagramas

Los diagramas son versiones simplificados del algoritmo, se saltean pasos para que sean fáciles de seguir.

1. **Insertar Elemento:**

   En este algoritmo, vemos como se inserta un elemento en un índice de la tabla que ya contiene elementos.



2. **Quitar Elemento:**

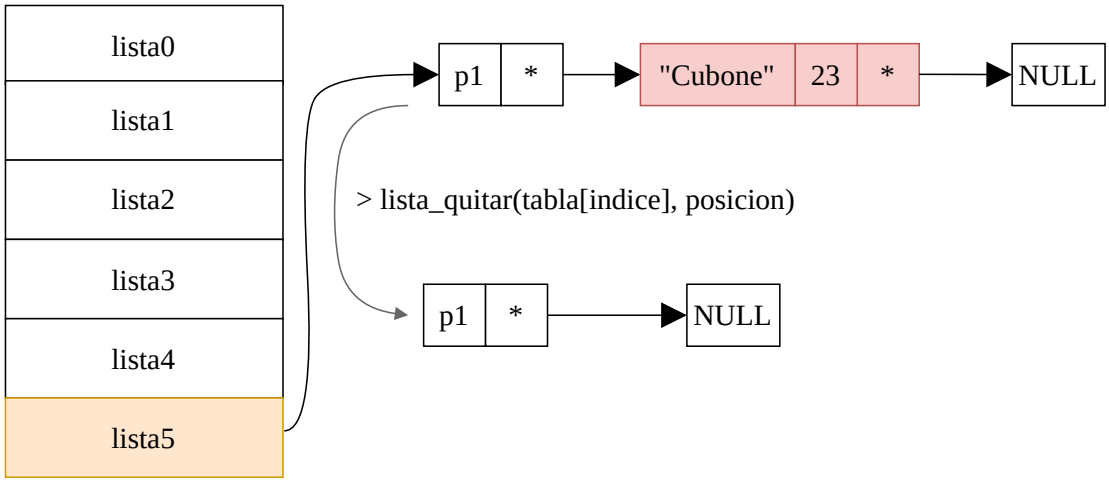
   En este algoritmo, vemos como elimina un elemento en un índice de la tabla que contiene dos elementos.

Quitar -> ["Cubone"]

funcion_hash("Cubone") = 1883

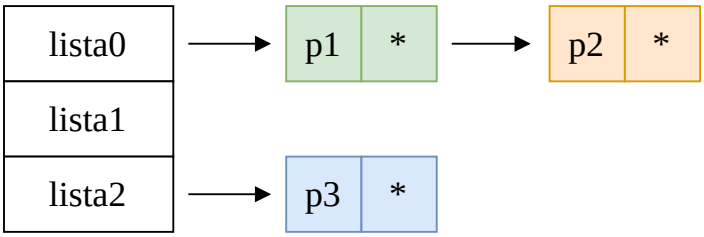
indice = 1883 % 6 = 5

posicion = indice_de_clave_en_lista(tabla[indice], "Cubone")



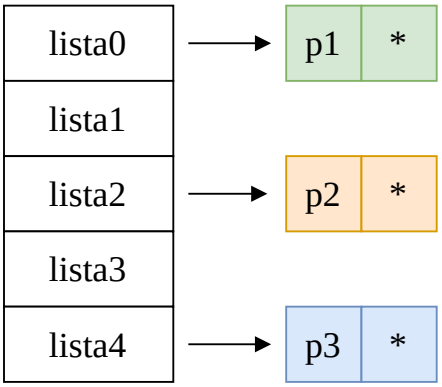
3. **Rehash:**

   En este algoritmo, vemos como se aplica rehash a una tabla que contiene tres elementos, su factor de carga es mayor al permitido.



Aplico Rehash

> hash_insertar(nuevo_hash, p1)
> hash_insertar(nuevo_hash, p2)
> hash_insertar(nuevo_hash, p3)



Una vez insertados los elementos en el nuevo hash, podemos destruir el hash antiguo

De esta forma, el algoritmo de búsqueda se vuelve mas eficiente.