Trabajo Práctico — TDA Hash

[7541/9515] Algoritmos y Programación II

Primer cuatrimestre de 2022

Alumno:
Vainstein Aranguren, Tomás

Número de padrón:
109043

Email:
tvainstein@fi.uba.ar

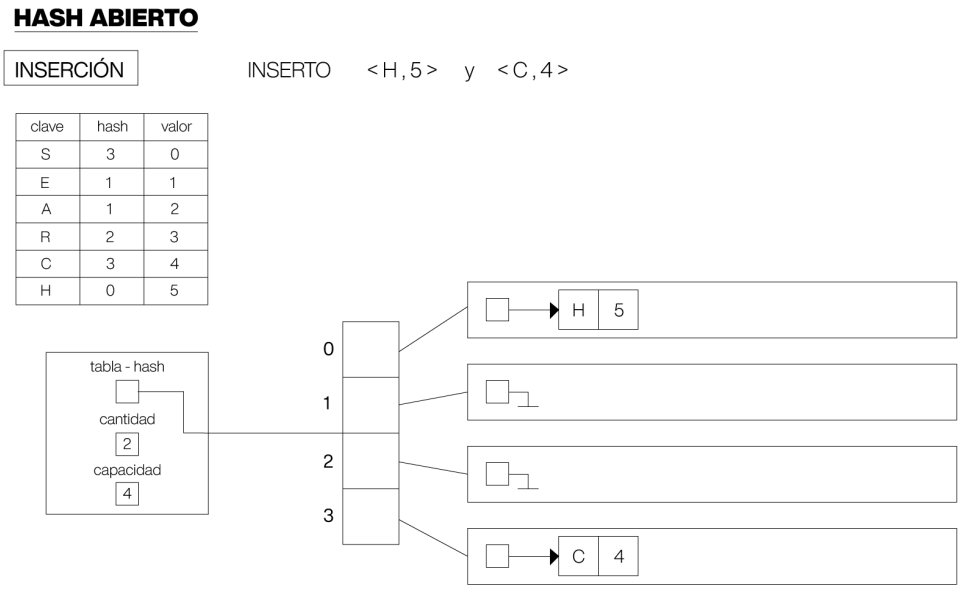
1. Introducción

Para este TP se pidió armar el código de un TDA Hash que permita el ingreso y borrado de datos de tipo no determinado, utilizando funciones como insertar, quitar, buscar e iterar, estableciendo estructuras privadas y criterios propios para manipular estos datos (función de hash, criterio de rehasheo, forma de implementación en cada posición de la tabla de hash).
Aparte, se crearon pruebas para facilitar el desarrollo del código utilizando la biblioteca pa2m.h.

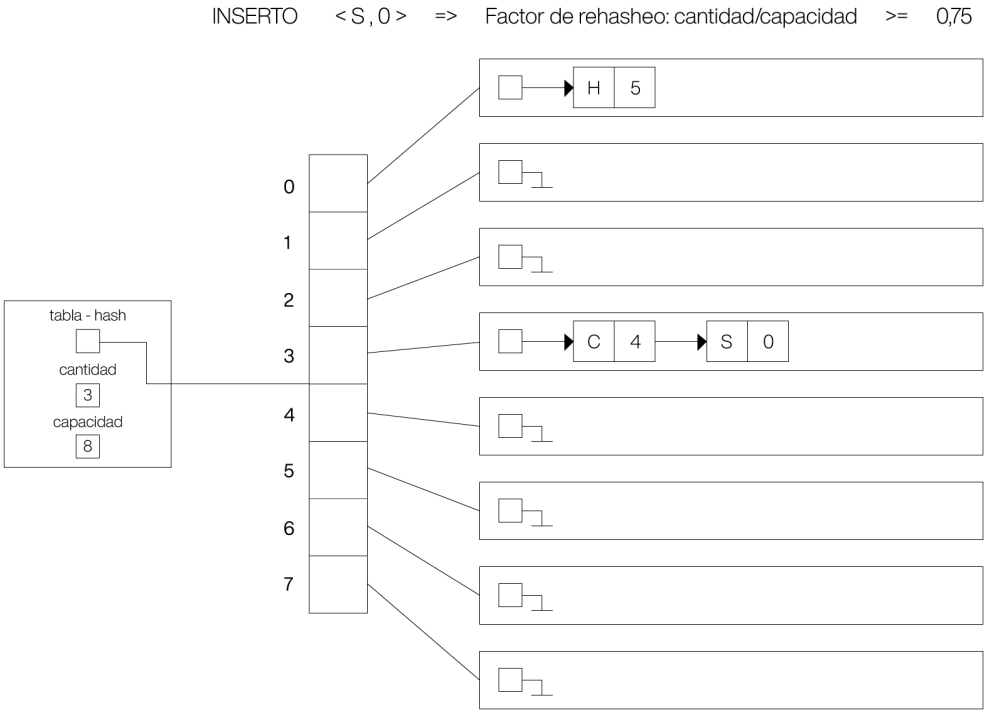
2. Teoría

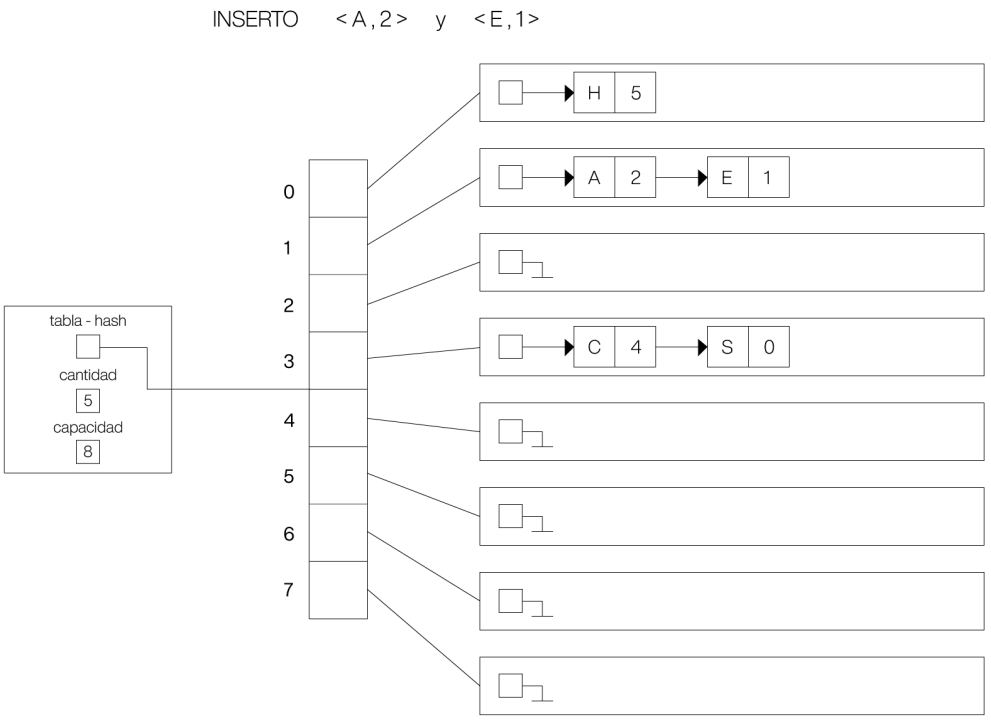
1. Tabla de hash abierta:

   Una tabla de hash abierta es una estructura con claves y elementos insertados con direccionamiento cerrado. En este tipo de implementación la tabla se trata de un vector de punteros que apuntan cada uno a una lista distinta.
   Para insertar en un hash abierto se utiliza la función de hashing para obtener la posición en el vector de listas e insertar en la lista correspondiente.

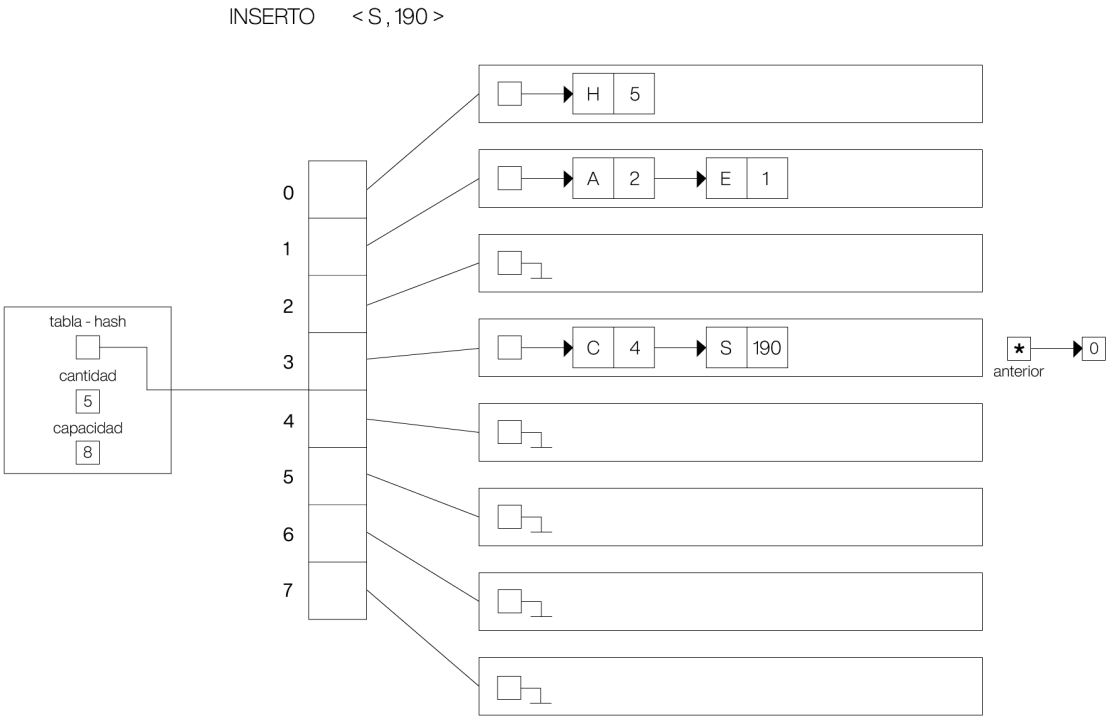


   En caso de insertar cierta cantidad de elementos en la tabla determinada por un cálculo entre la capacidad y la cantidad del hash, se rehashea la tabla. Esto consiste en aumentar la capacidad de de la misma para evitar que haya colisiones que conviertan una de las posiciones de la tabla en una lista que sea muy compleja de recorrer para insertar, quitar o recorrer elementos.

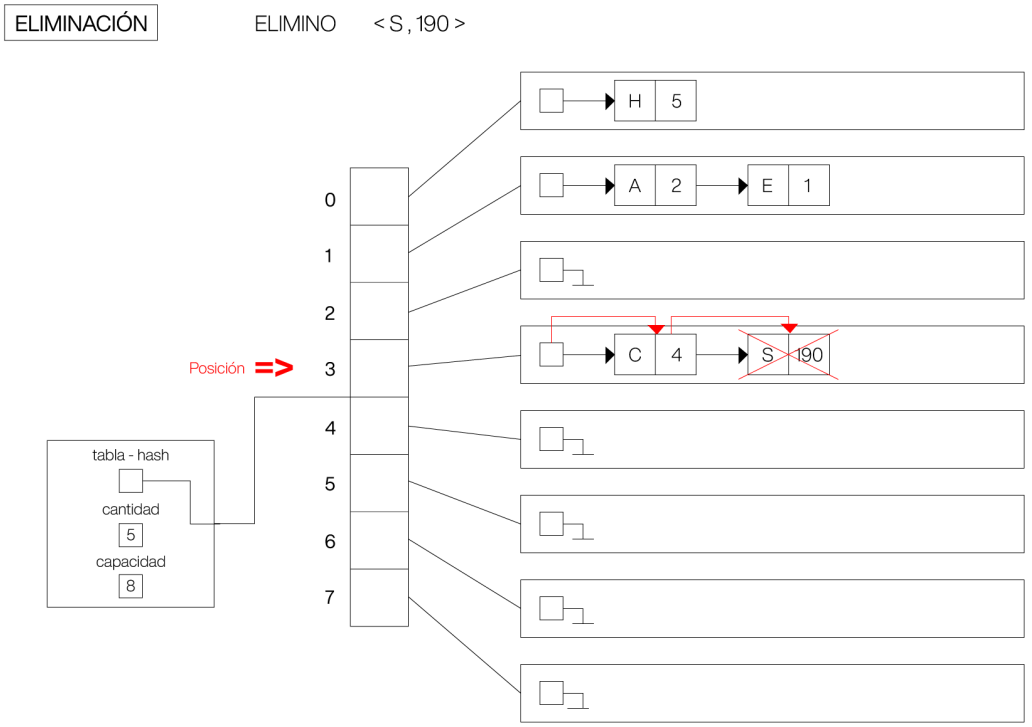


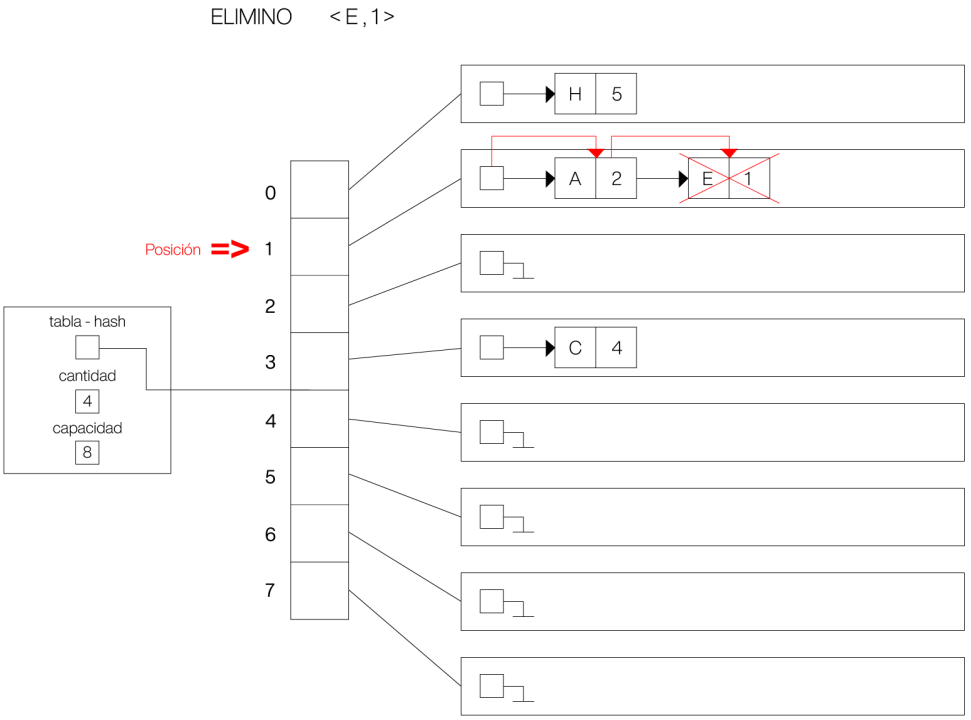


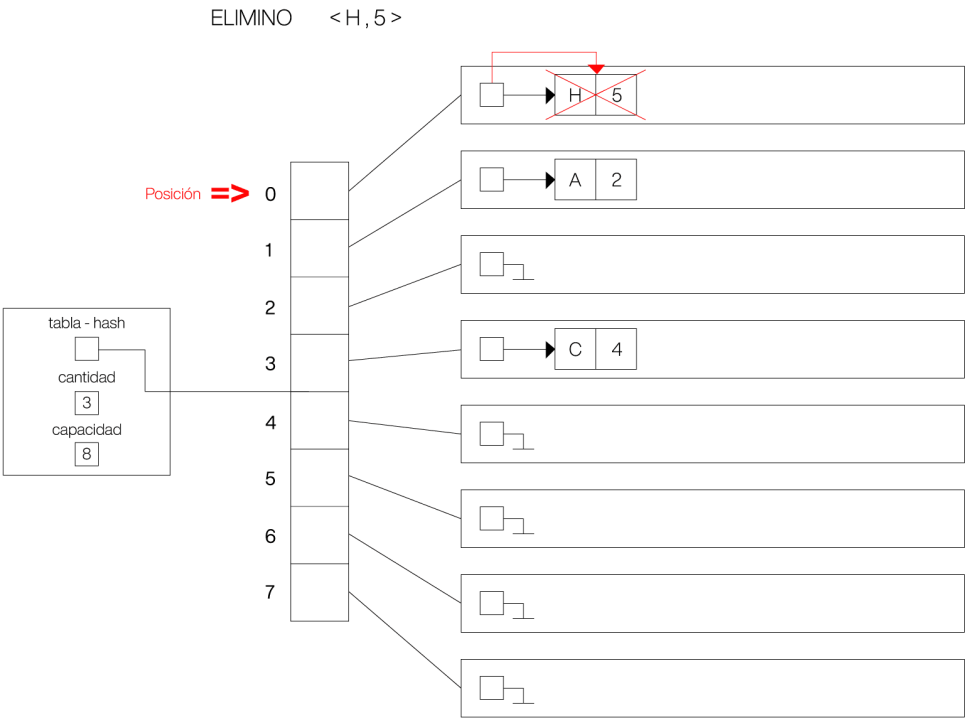
En caso de insertarse una clave en el hash con un valor distinto, se actualiza el valor, la cantidad del hash se mantiene igual y se utiliza la variable *anterior para determinar el valor que se encontraba insertado anteriormente.

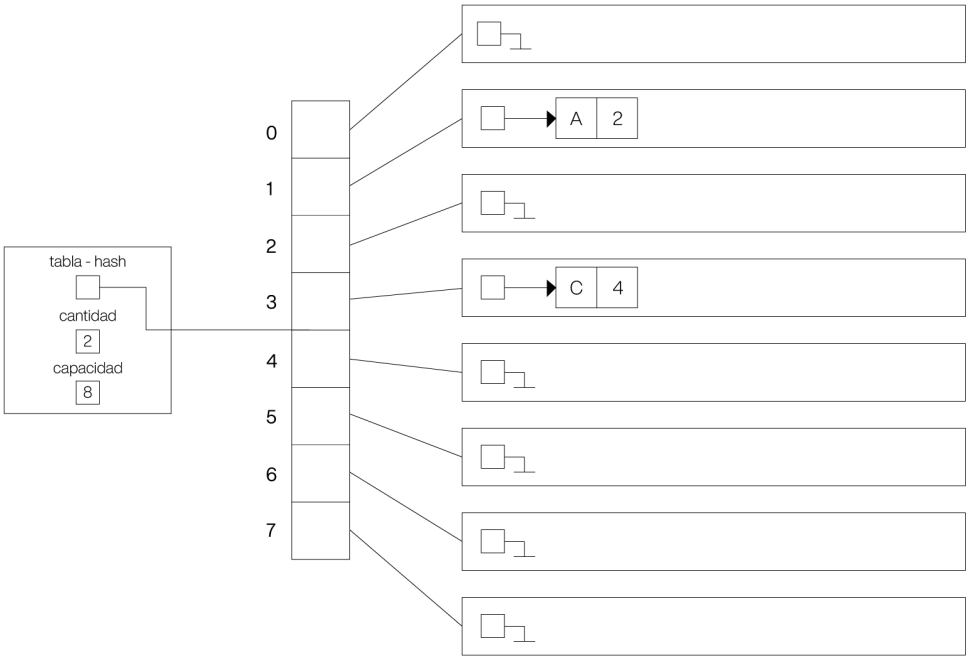


Para eliminar un elemento de un hash con direccionamiento cerrado, se ubica la posición de la clave que se quiere eliminar en la tabla y se recorre la lista correspondiente hasta encontrar el par.



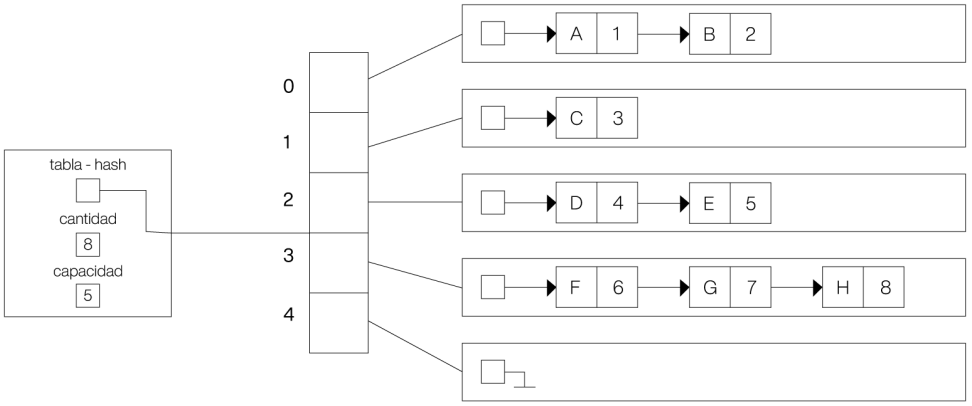

ELIMINO <E,1>
H 5
A 2
E 1
C 4
Posición =>
0
1
2
3
4
5
6
7
tabla - hash
cantidad
4
capacidad
8

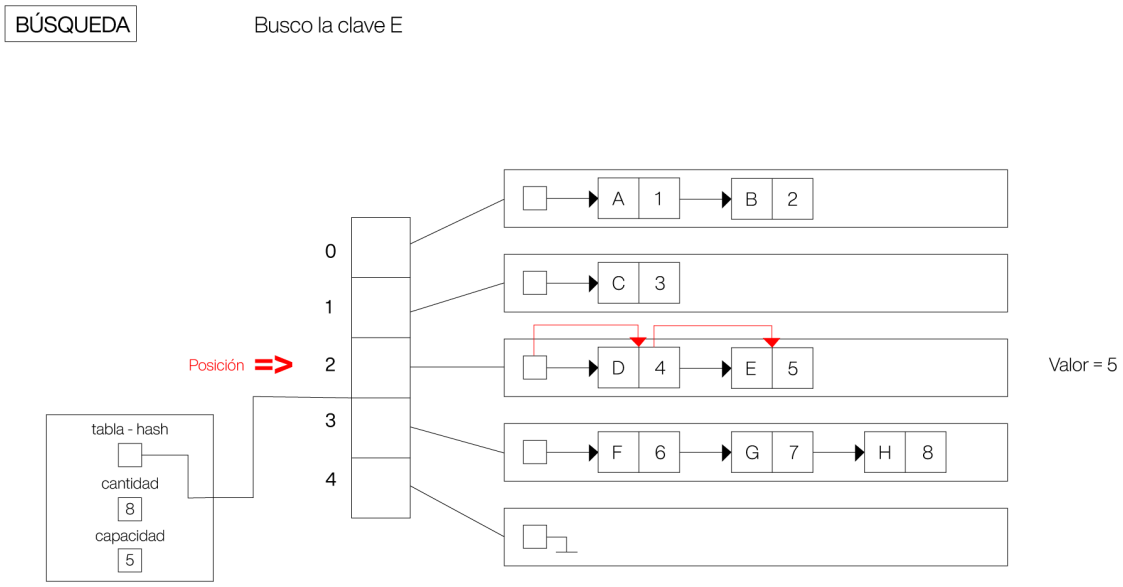

ELIMINO <H,5>
H 5
A 2
C 4
Posición =>
0
1
2
3
4
5
6
7
tabla - hash
cantidad
3
capacidad
8

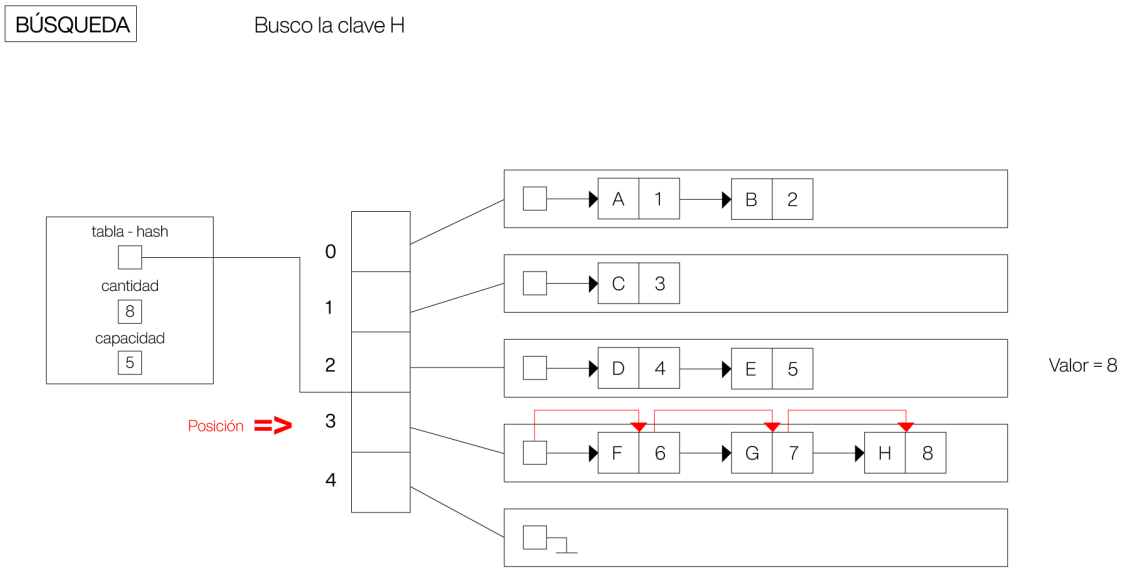


Para buscar en un hash abierto se encuentra la posición de la lista en la tabla con la función de hashing y se recorre la lista hasta encontrar el elemento que corresponde a la clave recibida.

BÚSQUEDA







2. Tabla de hash cerrada:

Al igual que el hash abierto, el hash cerrado es una estructura con claves y elementos insertados, pero con direccionamiento abierto.
A diferencia del hash con direccionamiento cerrado, los elementos y sus claves se insertan directamente en la tabla y no una estructura aparte.
Para insertar en un hash cerrado, las colisiones se manipulan de forma distinta que en el hash abierto, ya que solo puede haber un par por posición.

Hay tres formas de recorrer el hash cerrado:
-Probing lineal: busca el espacio vacío más cercano a la posición obtenida con el hashing.
-Probing cuadrático: (intentos fallidos)² para intentar insertar.
-Hash doble: aplicar una segunda función de hash a la clave cuando hay colisión.
La forma más sencilla para recorrer es el probing lineal.

HASH CERRADO

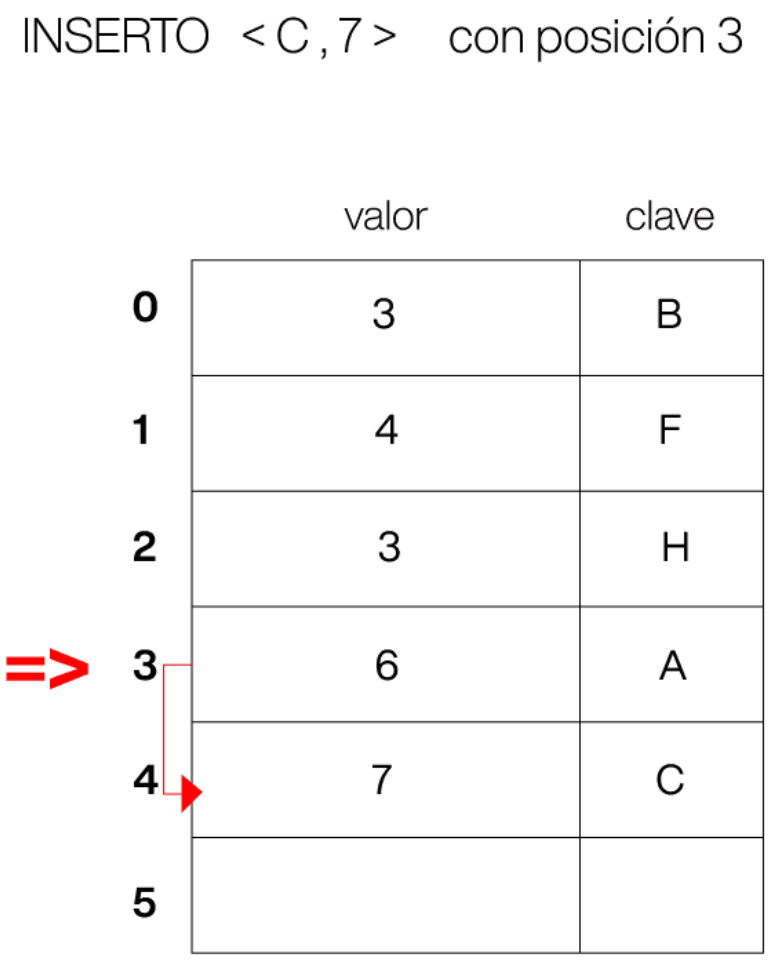
	valor	clave
0	3	B
1	4	F
2		
3	6	A
4		
5		

Para insertar en un hash cerrado se ubica la posición de la tabla y se verifica si la posición está ocupada o no.

INSERTO < H , 3 > con posición 2

		valor	clave
	0	3	B
	1	4	F
=>	2	3	H
	3	6	A
	4		
	5		

En caso de que haya una colisión, se busca la posición disponible más cercana a la colisión.



Si se quiere insertar una clave repetida, se actualiza el valor.

INSERTO < A , 9 > con posición 3

		valor	clave
	0	3	B
	1	4	F
	2	3	H
=>	3	9	A
	4	7	C
	5		

En caso de superar el factor de carga, se rehashea la tabla.

INSERTO < J , 200 > con posición 8

		valor	clave
	0	3	B
	1	4	F
	2	3	H
	3	9	A
	4	7	C
	5		
	6		
	7		
=>	8	200	J
	9		
	10		
	11		

Para quitar o buscar con probing lineal en un hash cerrado, se encuentra la posición en donde se encuentra el par con la función de hash y se verifica si la clave se encuentra en la posición o no.
En caso de que se encuentre la clave, se devuelve el valor en el caso de búsqueda o se libera la memoria en el caso de eliminación.
Si la clave no se encuentra en la posición correspondiente, se sigue recorriendo la tabla hasta encontrar la clave exitosamente o un espacio vacío, determinando esta última situación que el par no se encuentra en la tabla.

3. Detalles de implementación

Para empezar a programar las funciones del Hash primero hice diagramas para estudiar el comportamiento de cada una de las mismas, y cómo se comportan los elementos insertados o eliminados. Se implementó la tabla de hash utilizando un vector de listas con las funciones de la biblioteca “lista.h” para insertar, quitar, buscar o recorrer.
Las estructuras elegidas son las siguientes:

hash:
*-lista_t **tabla_hash; —>* un vector de listas para insertar elementos de tipo void*.
-size_t capacidad; —> una variable que indica la cantidad de listas dentro del hash.
-size_t cantidad; —> una variable que indica la cantidad de elementos en todas las listas del hash.
par hash (estructura utilizada como elemento a insertar en la lista):
*-const char *clave; —>* un string que guarda la clave de cada par.
*-void *valor; —>* un void* que guarda el valor a insertar de cada par.

Para crear el hash utilicé la función *hash_crear que reserva memoria para una variable de tipo hash_t e inicializa la capacidad del mismo con la variable recibida o en tres si la esta es menor a tres. Se utiliza la función de crear_tabla_hash para reservar memoria para el vector de listas. En caso de fallar la creación de al menos una lista con la función lista crear se liberan con destruir_tabla_hash todas las anteriores listas que se crearon exitosamente, se devuelve NULL y se libera la memoria del hash_t creado.

3.1. Detalles de hash_insertar

El objetivo de esta función es insertar los elementos en una de las listas de la tabla hash establecida por la funcion_hash que obtiene una posición a partir del módulo entre un valor y la capacidad. Este valor de hashing se determina por la multiplicación del valor de la tabla ASCII de cada char del string recibido y el valor de una variable i que se inicializa en 15 y al cual se le va sumando 25 en cada iteración. De esta forma, la función devuelve un valor distinto para cada string determinado por sus distintos caracteres, su cantidad y el orden de los mismos.
Para obtener el par a insertar, primero se utiliza la función listar_buscar_elemento con la lista establecida por la posición que devuelve la función de rehash, la clave que recibe la función de insertar y la función comparador que compara la clave recibida con la clave del elemento actual de lista que se está recorriendo, para determinar si la clave ya existe en la tabla de hash o no. Si la clave ya fue insertada en el hash, se actualiza el elemento guardado en esta clave con el elemento recibido por parámetro, y si *anterior es distinto de NULL, se iguala al valor que estaba previamente guardado en esa posición.

En caso de que no se encuentre el par con la función lista_buscar_elemento, se obtiene un nuevo par con la función crear_nuevo_par, la cual reserva memoria para una variable par_hash_t, almacena el valor en el campo valor del nuevo par y la clave duplicada con la función duplicar_clave.
Se utiliza la función lista_insertar para insertar el par creado en la lista correspondiente. Si la inserción es exitosa, se suma uno a la cantidad del hash, y en caso de que *anterior sea distinto de NULL, se le asigna NULL.
Al final de la función, utilizando la función hay_sobrecarga, se verifica si es necesario rehashear comparando que la cantidad del hash dividida por la capacidad sea menor o igual al valor 0,75. Si la función devuelve true, se llama a funcion_rehash, la cual multiplica por 2 la capacidad del hash y crea una nueva tabla con la capacidad actualizada, la cual se asigna como nueva tabla de hash.
Para reinsertar todos los elementos de la tabla de hash anterior, se recorren todas las listas de la tabla, y por cada una se obtiene cada par de cada posición con la función lista_quitar_de_posicion con posición 0 para hacer la eliminación siempre desde el nodo inicio y se inserta en la nueva tabla hasta que cada lista del hash anterior esté vacía. Al finalizar la iteración de cada lista, la misma se destruye con lista_destruir. Cuando se termina de recorrer toda la tabla anterior, se libera la memoria de la misma.

3.2. Detalles de hash_quitar

La función de abb_quitar se encarga de borrar un par de la tabla hash determinado por la clave recibida por parámetro y la función de hashing.
Para recorrer la lista determinada por la posición obtenida se crea un iterador de lista que recorre todos los pares hasta llegar al final o hasta que la función comparador devuelva 0. Si la clave recibida y la clave del elemento actual son iguales, esto último se cumple y se elimina el par de la lista usando la cantidad de iteraciones como posición en la cual se debe eliminar y se resta uno de la cantidad del hash. Luego se libera la clave, el par, se destruye el iterador y se devuelve el elemento del par eliminado.
En caso de que la función devuelva un valor distinto de 0, se avanza en el iterador y se suma un valor al contador de iteraciones. Si nunca se cumple la condición de la función, se destruye el iterador igualmente y se devuelve NULL.

3.3. Detalles de hash_obtiene/hash_contiene

Para buscar elementos en el hash se utiliza la función de hashing para obtener la posición de la lista en la tabla en donde se encuentra el elemento de la clave correspondiente y se utiliza la función de lista_buscar_elemento, en la cual se determina si la clave recibida coincide con la clave del elemento que se está buscando con la función comparador. Si se obtiene el par exitosamente, se devuelve el valor del mismo.
Para la función hash_contiene, se utiliza la función hash_obtiene para determinar si se encuentra el elemento en el hash.

Para compilar el TDA Hash, se utilizó el archivo makefile adjuntado en la entrega.
Los comandos en cuestión son:
- *make ejemplo*, para compilar el trabajo.
- *make valgrind*, para ejecutar el programa con Valgrind.
- *make pruebas*, para compilar el archivo de pruebas.
- *make valgrind-pruebas*, para ejecutar las pruebas con Valgrind.