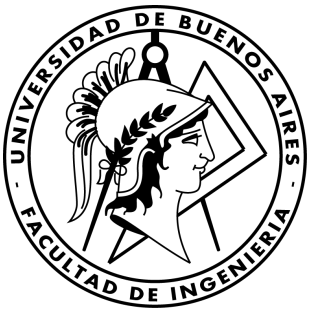
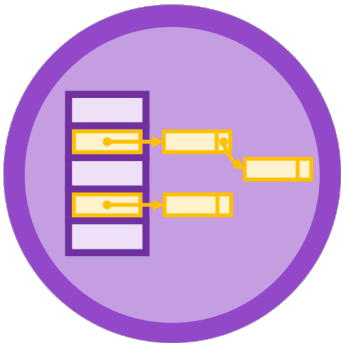


TDA Hash



[7541/9515] Algoritmos y Programación II
Primer cuatrimestre de 2022

Alumno:	Davies, Alen
Número de padrón:	107084
Email:	adavies@fi.uba.ar

Índice

1. Introducción

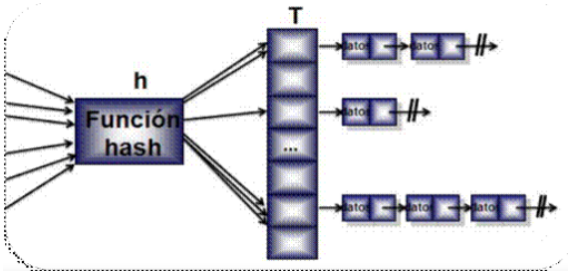
Para este TP se pide implementar una tabla de hash abierta. Para ello en la carpeta src provista se encuentran las firmas de las funciones a implementar en archivos .c y .h. También se pide implementar un iterador interno para recorrer el hash.
Además de la carpeta src que contiene los archivos con las firmas de las funciones a implementar, se brinda tambien un archivo de ejemplo para verificar que el TDA funcione correctamente.
Otra de las cosas pedidas para este TP es la creación de nuestras propias pruebas y para ello se incluye un archivo pruebas.c.
El TDA entregado deberá compilar y pasar las pruebas de la cátedra en Chanutron sin errores y sin pérdida de memoria.

2. Tabla de Hash

Una tabla de hash es una estructura de datos que asocia claves con valores lo que permite el acceso a los elementos almacenados a partir de una clave generada.
Funciona transformando la clave con una función hash, en un número que identifica la posición donde la tabla hash localiza el valor deseado.
Cuando al aplicarle la función hash a dos claves diferentes, me da el mismo número, es decir que estoy queriendo meter un par clave valor en una posición que está ocupada, se denomina colisión.
Dependiendo de como manejemos las colisiones, el tipo de hash puede ser abierto (direccionamiento cerrado) o cerrado (direccionamiento abierto).

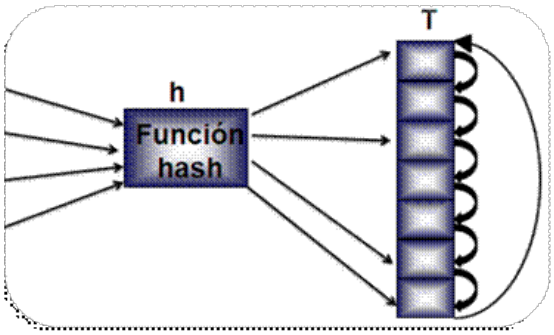
2.1. Tabla de hash abierta

Este tipo de hash usa direccionamiento cerrado. En este hash, cada posición es un puntero a una lista. Entonces al insertar un elemento se inserta en la lista que corresponde a esa posición de la tabla de hash, y si se quiere insertar otro elemento en una posición ocupada se enlaza a la lista de esa posición.
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.



2.2. Tabla de hash cerrada

Este tipo de hash usa direccionamiento abierto, todos los valores se guardan dentro de la misma tabla. Si hay una colisión, se le asigna otra posición a ese elemento dentro de la tabla de hash.



2.3. Operaciones

Las operaciones básicas de una tabla de hash son:

- Crear.
- Insertar.
- Obtener.
- Cantidad.
- Quitar.
- Contiene.
- Destruir.

3. Implementación TDA Hash

Para este TP se pidió implementar una tabla de hash abierta.
A continuación se detallarán las operaciones más importantes con su complejidad y su implementación en el TDA.
Quedó a criterio de cada alumno las estructuras a utilizar, mi implementación se baso en las siguientes estructuras:

```
typedef struct nodo {
        void *elemento;
        const char *clave;
        struct nodo *siguiente;
} nodo_t;
```

```
struct hash {
        nodo_t **tabla;
        int capacidad;
        int ocupados;
};
```

Para la implementación de este TDA, realicé varias funciones auxiliares que me permitieron manipular los nodos de cada lista dentro de la tabla de hash.

3.1. Crear

Para crear un hash se reserva memoria para la estructura y para su tabla. Al crear el hash, se le asigna como cantidad la pasada por parámetro a la función de crear.
Su complejidad es O(1) ya que se realiza una sola vez.

3.2. Insertar

Para insertar un elemento en el hash, primero hasheamos la clave a insertar.
Luego al tener la posición, debemos ver si la lista en esa posición está vacía, de ser así creamos un nuevo nodo a insertar en la lista y lo insertamos en la primer posición.
Si la lista no está vacía, debemos ver si la clave está repetida en nuestro hash. Si la clave está repetida debemos reemplazar el elemento que se encuentra en esa posición por el nuevo elemento que se quiere insertar.
Si la lista en esa posición del hash no está vacía y la clave no está repetida, entonces creamos un nuevo nodo para insertar el elemento en la tabla y recorremos la lista de esa posición para encontrar su final y así insertar el nuevo nodo en la última posición de la lista.
Su complejidad es O(n) ya qué al insertar un nuevo nodo en alguna de las listas de la tabla de hash, debemos recorrer cada posición hasta llegar al final y así insertar el nuevo nodo en la última posición de la lista.

3.3. Quitar

Para quitar un elemento de nuestro hash primero chequeamos si la clave existe dentro de nuestro hash, de lo contrario no podremos quitarlo.
Luego, hasheamos la clave del elemento que queremos eliminar para obtener la posición dentro de la tabla. Teniendo ésto, creamos un nodo auxiliar que llamaremos actual y un segundo nodo que va a apuntar siempre al anterior del auxiliar, llamado anterior. Recorremos el hash con el nodo actual hasta encontrar la clave asociada al elemento que se desea eliminar. Una vez encontrada, destruimos el nodo correspondiente a ese elemento pero guardándonos el elemento en una variable para poder devolverlo y luego enlazamos los demás nodos para que la lista siga unida.
Si al llegar al nodo que contiene el elemento que queremos eliminar, nuestro nodo anterior es NULL significa que el actual está apuntando al primer nodo de la lista por lo tanto destruimos el primer nodo y ahora nuestro nodo inicio será actual->siguiente.
Si el nodo anterior no es NULL, entonces apuntamos anterior->siguiente a actual->siguiente y destruimos actual, de esa forma los nodos quedarám enlazados al destruir el nodo que contiene al elemento que queremos eliminar.
La complejidad de quitar es O(n) ya que hay que recorrer las listas del hash buscando el elemento que se quiere quitar.

3.4. Obtener

Para obtener un elemento asociado a una clave, debemos hashear la clave pasada por parámetro para obtener la posición del hash donde esa clave se encuentra y recorrer la lista de esa posición hasta encontrar el elemento asociado a la clave. Cuando se encuentra el elemento, se deja de iterar y se devuelve el mismo.
La complejidad es O(n) ya que debemos recorrer la lista en la que se encuentra la clave pasada por parámetro hasta encontrar el elemento que se desea obtener. El elemento podría estar en la última posición de la lista y se deberían recorrer todos los elementos.

3.5. Destruir

Para destruir el hash se debe recorrer cada posición de la tabla y cada lista y asi ir destruyendo cada uno de los nodos de las listas, la tabla y por último el hash.
Su complejidad es O(n) ya que se recorren todos los elementos del hash para eliminar cada uno de los nodos de las listas de cada posición de la tabla.

3.6. Iterador interno

En la función hash_con_cada_clave se pide recorrer cada elemento del hash, aplicar la función pasada por parámetro a cada par clave-elemento y devolver la cantidad de elementos recorridos. Si la función devuelve false, se debe dejar de recorrer.
Para recorrer cada elemento del hash se debe recorrer la tabla y a su vez cada lista de cada posición.

4. Compilación y ejecución

Dentro del .zip entregado, se encuentra un makefile que contiene los comandos para compilar el programa, correr las pruebas y verificar errores de memoria.