TDA Hash

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

Alumno:	PELLEGRINO, Dafne
Número de padrón:	104368
Email:	dpellegrino@fi.uba.ar

Índice

1. Introducción

Se pide implementar una Tabla de Hash abierta (direccionamiento cerrado) en C. Para ello se brindan las firmas de las funciones publicas a implementar y se deja a criterio del alumno la creación de las estructuras y las funciones privadas del TDA para el correcto funcionamiento de la Tabla de Hash. La función de Hash a utilizar sera interna de la implementación y también debe ser decidida por el alumno. Para esta implementación las claves admitidas por la tabla seran solamente strings. Adicionalmente se pide la creación de un iterador interno que sea capaz de recorrer las claves almacenadas en la tabla.

2. Teoría

El hash es un tipo de dato que consiste en una tabla que relaciona claves con elementos. Los elementos obtienen una posición en la tabla mediante una función (la función de hash) que no es conocida por el usuario, por lo tanto tampoco lo es la posición en la que se guarda el elemento. Los pares <clave,elemento>se pueden acceder siempre mediante la clave. El usuario debe proveer el comparador necesario para poder buscar el elemento deseado mediante la clave.

Existen al menos 2 tipos de hash. Hash abierto (de direccionamiento cerrado) y hash cerrado (de direccionamiento abierto). La diferencia en ambos consiste en la cantidad de elementos que es posible almacenar en relación a la cantidad de casilleros de la tabla. También difiere en el manejo de las colisiones (es decir cuando se quiere insertar un elemento cuya posición luego de realizar el hash coincide con la de un elemento insertado con anterioridad).

En el hash abierto, se pueden almacenar más claves que casilleros existentes en la tabla, ya que en casos de colisiones utiliza el método de encadenamiento (la forma del mismo puede ser variada, desde una lista a un árbol u otros métodos de enlazamiento) para poder almacenar más de una clave por posición en la tabla. Esto puede dificultar la búsqueda ya que es posible que haya muchos elementos y haya que iterar hasta n veces (siendo n la cantidad de elementos en la tabla) en el peor de los casos para poder encontrar una clave. Se lo puede llamar de direccionamiento cerrado porque los elementos quedan almacenados en la posición que se obtiene aplicando la función de hash.

En el hash cerrado, no se pueden almacenar más claves que casilleros existentes en la tabla, ya que se almacena solo un elemento por casillero. Este tipo de hash maneja las colisiones recorriendo la tabla para ver cual es el siguiente casillero libre e insertando la clave ahí. Esto ultimo se puede hacer mediante "probing lineal"(buscar el siguiente espacio libre inmediato),"probing cuadrático"((intentos fallidos)2 para intentar insertar) o hash doble (aplicar una segunda función de hash). Esto puede dificultar la búsqueda ya que es posible que haya muchos elementos que no se encuentran en la posición esperada y haya que recorrer la tabla para poder encontrarlos.

3. Detalles de implementación

Para compilar el programa se puede utilizar el atajo de makefile "make pruebasz luego para ejecutar el programa corriéndolo en valgrind para chequeo de memoria puede sencillamente ejecutar el comando definido en el makefile con "make valgrind-pruebas".

Para la implementación del hash defino una estructura en la que se almacenará la cantidad de casilleros de la tabla, la cantidad de casilleros ocupados, la cantidad de elementos almacenados y un puntero a la tabla. Para la implementación de la tabla utilizo el TDA Lista implementado previamente para esta materia, quedando la tabla definida como un vector de listas.

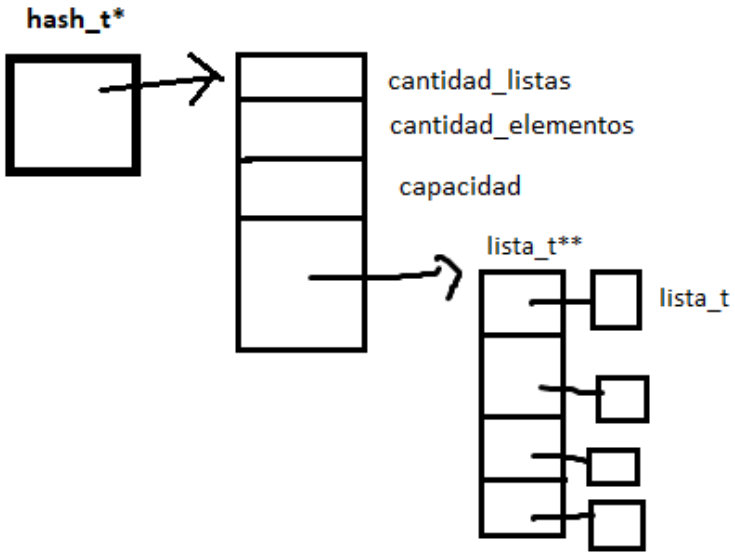

Figura 1: Diagrama 1: Estructura hash_t

Para poder almacenar los elementos en la lista ya definida sin cambiar la implementación que ya tengo, creo una estructura elemento de hash en el que se almacena la clave y el puntero al dato del usuario.

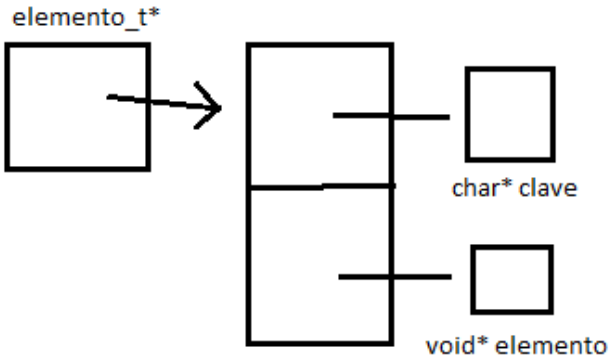

Figura 2: Diagrama 2: Estructura elemento_t

Para esta implementación en particular solo se admitirán claves de tipo string, por lo que la función utilizada para comparar claves no es necesario obtenerla del usuario. Además se tiene una cantidad mínima de casilleros para poder crear un hash. En caso de que la cantidad de casilleros deseada sea menor a 3, automáticamente se cambia ese valor a 3.

3.1. Detalle creación, inserción y rehash

La función de creación reserva memoria para un hash (tipo de dato hash_ t) y lo inicializa con cantidad 0 de listas y de elementos. Es importante que chequee que la capacidad con la que se debe crear el hash, no debe ser menor a 3.

Para insertar se recibe la clave, el elemento a insertar, un puntero a su *anterior* y el hash en el que lo inserta. Se devuelve el hash si se pudo insertar el elemento y NULL si no se pudo insertar. También se chequea si es necesario rehashear (aumentar el tamaño del hash y volver a insertar los elementos antes de la inserción. El factor de carga es 60 % y lo tomo considerando cantidad de elementos en cada lista dentro del hash, ya que al reutilizar el TDA lista y no poder acceder a sus estructuras internas, muchas operaciones se complejizan cuando las listas son muy largas. Tomo como máxima cantidad en una lista 8 elementos. La función de rehash aumenta la cantidad de casilleros en la tabla y recorre todo el hash volviendo a insertar los mismos para que estén bien distribuidos. No utiliza la función de insertar hash ya que esta al recorrer la lista en busca de duplicados haria que tarde mas y al rehashear no es necesario chequear por duplicados porque todos son elementos que ya estaban en un hash.

3.2. Detalle de quitar y destruir / destruir todo

Para las funciones de destrucción se recorre todo el hash liberando la memoria reservada para las claves y en el caso del destruir todo también de los valores almacenados si es que se recibe una función de destrucción.

En el caso de quitar un elemento particular se determina en que posición de la tabla esta mediante la función de hash y luego se busca comparando. Una vez encontrado el elemento se quita invocando lista quitar de posición.

3.3. Detalle de obtener y contiene

Para ambas funciones se determina en que posición de la tabla está mediante la función de hash y luego se busca comparando mediante la función lista con cada elemento y un comparador apropiado.

3.4. Detalle de hash con cada clave

La función recorre todo el hash aplicando la función recibida a todos los elementos hasta que la misma retorna false. Devuelve la cantidad de veces que fue llamada la función.