Práctica 6 - Tablas Hash

1. Lea la implementación provista de tablas hash, y el ejemplo presentado en el archivo `main.c`. Explique qué pasa si se ejecuta `tablahash_insertar` dos veces con la misma clave. Explique la razón por la cual se produce un error en la ejecución de `main.c`.

2. La implementación actual sufre de un problema: `tablahash_buscar` compara las claves usando la igualdad. Explique qué problemas puede traer esto.
Modifique la implementación, agregando un miembro a la estructura `TablaHash` del tipo:

```
typedef int (*FuncionIgualdad)(void *, void *)
```

El objetivo de este miembro es determinar cuándo punteros a claves representan la misma clave. Modifique las funciones que sean necesarias para funcionar con este nuevo miembro.
Luego de haberlo implementado, compruebe que `main2.c` funciona correctamente.

Direccionamiento abierto

3. Modifique la implementación dada para soportar colisiones, utilizando direccionamiento abierto con un sondeo lineal. Será necesario diferenciar las casillas vacías de las eliminadas.

4. Modifique la implementación del ejercicio anterior, para considerar un sondeo doble, utilizando una segunda función de hash h_2 que determine el intervalo de sondeo:

$$h_2(k) = 1 + (k \ mod \ (m-1)) \qquad \text{(para claves numéricas)}$$

Donde m es el número de casillas, el cual debe ser un número primo, por ejemplo: 101.

5. Modificar la operación de búsqueda de manera que, al realizar el sondeo, registre si visita una casilla marcada como eliminada, y en dicho caso, mueva el dato a dicho lugar antes de retornar. De manera que las siguientes búsquedas de la misma clave sean más eficientes.

Encadenamiento separado

6. Modifique la implementación dada para resolver colisiones, por encadenado.

7. Cuando una tabla hash tiene demasiados valores insertados es conveniente aumentar la cantidad de casillas para disminuir la cantidad de colisiones. Para esto, es necesario recalcular el hash de todas las claves de la tabla para tomar en cuenta el nuevo máximo número de casillas. Esta operación se denomina comúnmente *rehash*.
Implemente la función: `void tablahash_redimensionar(TablaHash *)` que duplique la cantidad de casillas de la tabla y reposicione todos los elementos de acuerdo a la nueva posición que le asigne la función de hash.

8. Se denomina *factor de carga* a la relación entre el número de elementos insertados en la tabla y el número de casillas (`NumElementos / NumCasillas`).
Modifique la implementación de tablas hash para que en cada inserción se calcule el *factor de carga* y en caso de ser mayor a un cierto límite (por ejemplo: 0,7) se redimensione la tabla al doble de su tamaño.

9. Cree un programa que simule el uso de una agenda telefónica. Dicha agenda debe implementarse con una tabla hash donde las claves serán los nombres de las personas en la agenda y los valores serán los números telefónicos (implementar una función hash para strings). Los comandos que deberá soportar por entrada estándar son:

Comando	Argumentos	Resultado	Ejemplo
insertar	nombre, número	Inserta el par (nombre, número) indexado por nombre, modifica el número si ya existe el nombre en la agenda	insertar Juan Perez, 4101010
eliminar	nombre	Borra la entrada perteneciente a nombre (si existe)	eliminar Juan Perez
buscar	nombre	Imprime el número de 'nombre' si está en la agenda	buscar Juan Perez