Trabajo Práctico — TDA HASH

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

Alumno:	Oriz, Omar
Número de padrón:	108777
Email:	ooriz@fi.uba.ar

Índice

Tabla de contenido

1 Introducción **1**

2 Hash (teoría)

2.1 Tabla de hash abierta **2**

2.2 Tabla de hash cerrad **5**

3 Detalles de implementación **9**

1. Introducción

Para este trabajo, se pidió la implementación de una Tabla de Hash abierta(direccionamiento cerrado) en lenguaje C, partiendo de la firmas de las funciones públicas a implementar. Adicionalmente se crearon estructuras y funciones auxiliares privadas del TDA cuyas firmas se encuentran en el archivo estructuras_y_auxiliares.h. Para esta implementación se pide que las claves admitidas por la tabla sean solamente strings. Adicionalmente se pide la creación de un iterador interno que recorra todas las claves almacenadas en la tabla.

Para esta tarea se implementó una metodología de desarrollo **orientada a pruebas**, realizando una serie de pruebas estándares de caja negra según el funcionamiento esperado de cada primitiva del TDA y agregando nuevas ante cada bug encontrado durante el testeo.

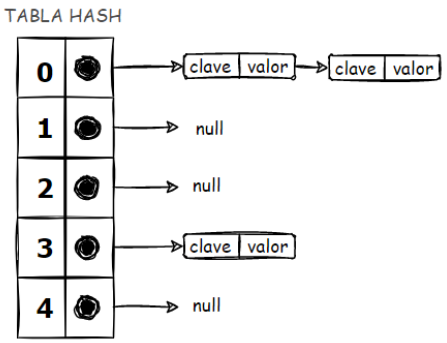
2. Hash (teoría)

Una Tabla de Hash es una estructura de datos que asocia claves con valores. La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos almacenados a partir de la clave única con la que se los guarda (con complejidad promedio O(1)) . Funciona transformando la clave con una función hash en un código numérico, que, aplicándole la operación de resto de una división entera (%) por la capacidad de la tabla, identifica la posición donde se almacena el valor deseado en dicha tabla.

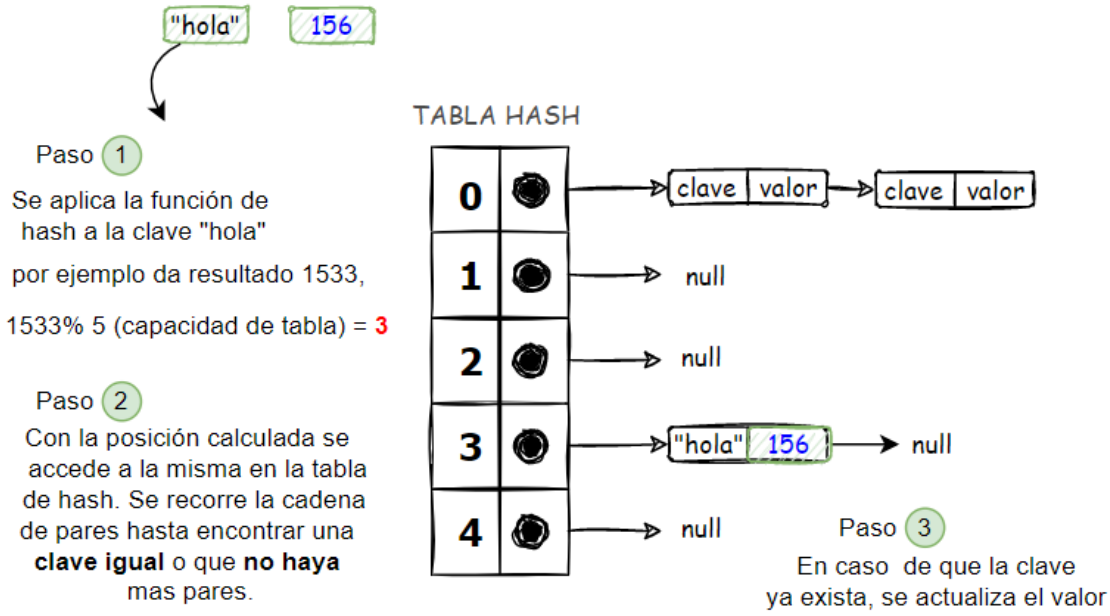
Existen dos tipos de implementaciones posibles según su método de resolución de colisiones de pares clave – valor a los que les corresponde la misma posición en la tabla.

2.1. Tabla de hash abierta

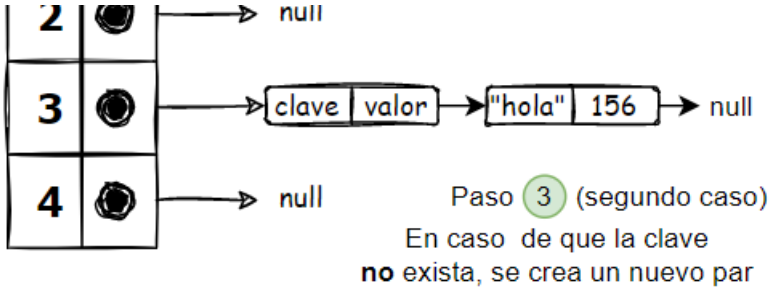
Las tablas de hash abiertas tienen por característica su direccionamiento cerrado, esto se debe a la estrategia que se adopta en este tipo de hashes para las colisiones, que es el encadenamiento o chaining. El chaining consiste en encadenar pares de clave valor a los que les corresponde la misma posición, por fuera de la tabla de hash y manteniendo la dirección de esta cadena desde la posición mencionada. Por lo tanto, el direccionamiento cerrado hace referencia a que los pares clave valor siempre se almacenarán relacionados a la posición calculada que les toca inicialmente, como se muestra en la figura:



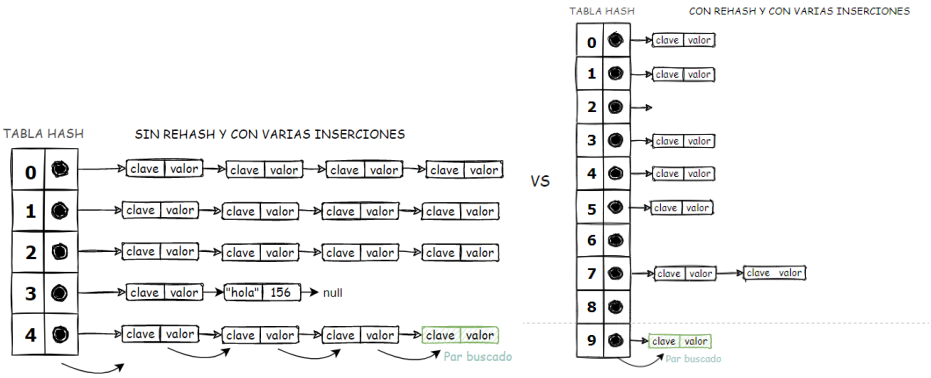
A continuación, con diagramas, se ilustra la operación de **inserción** en este tipo de hash:



Y el otro caso:



Además de los pasos ilustrados, también debemos incluir una operación para mantener la complejidad promedio de las operaciones en O(1): el **rehash**. El rehash es una operación que se encarga de aumentar el tamaño de la tabla y redistribuir los pares ya creados en el hash calculándoles sus *nuevas* posiciones. De esta manera se evita el encadenamiento de varios pares en una sola posición de la tabla:



Esta operación se realiza con un criterio que involucra un factor de carga de la estructura. Es decir, si con una inserción nueva se sobrepasaría ese factor de carga se aplica la operación de rehash.

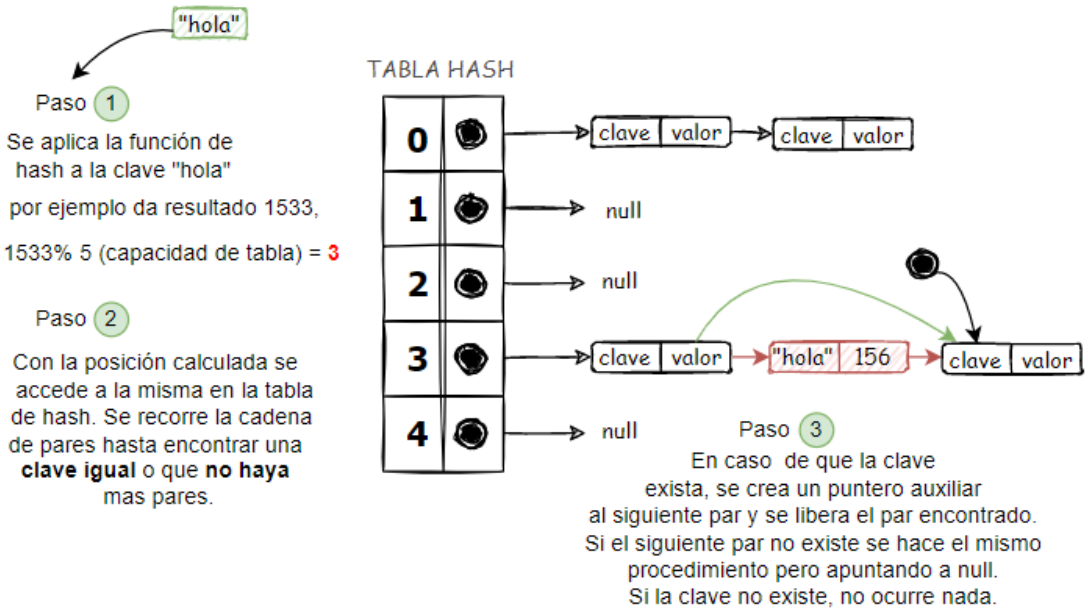
Este factor de carga se calcula como:

$$\frac{(pares\ existentes + 1)}{capacidad\ de\ tabla}$$

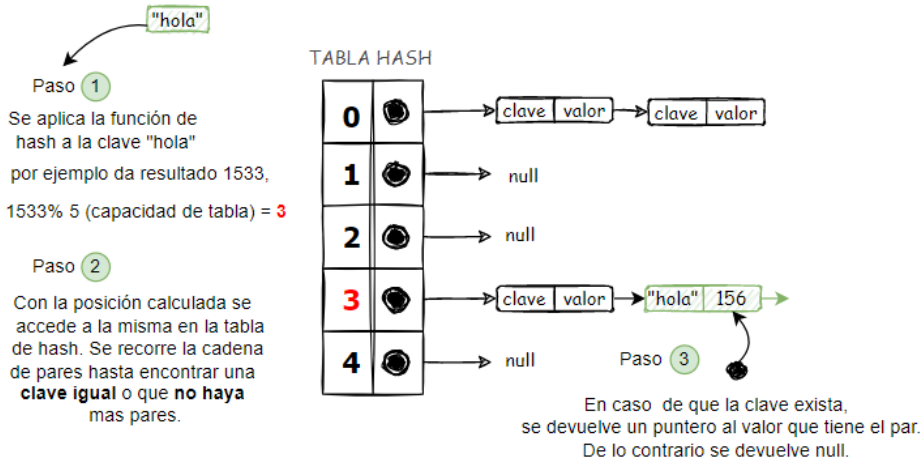
El valor a sobrepasar es seteado por el diseñador del TDA según las necesidades de uso del mismo. (80% → 0,8; 70% → 0,7; etc.). El aumento de tamaño de la tabla de hash en esta tarea también puede variar dependiendo las necesidades de uso.

El resultado de aumentar la capacidad de la tabla es que al calcular la posición real de la tabla de un código de hash de una clave y el código de otra clave que dé una posición real igual en el hash anterior, se obtengan posiciones reales diferentes en la nueva tabla.

Avanzamos sobre la operación de **Eliminación** de pares clave – valor de un hash abierto:

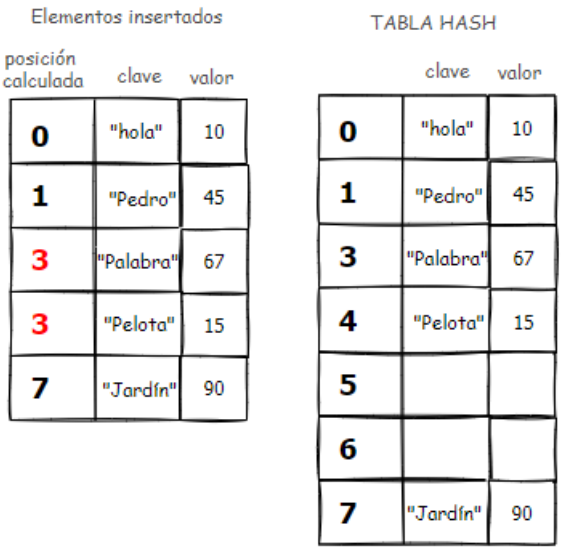


Vemos ahora la operación de **Búsqueda:**

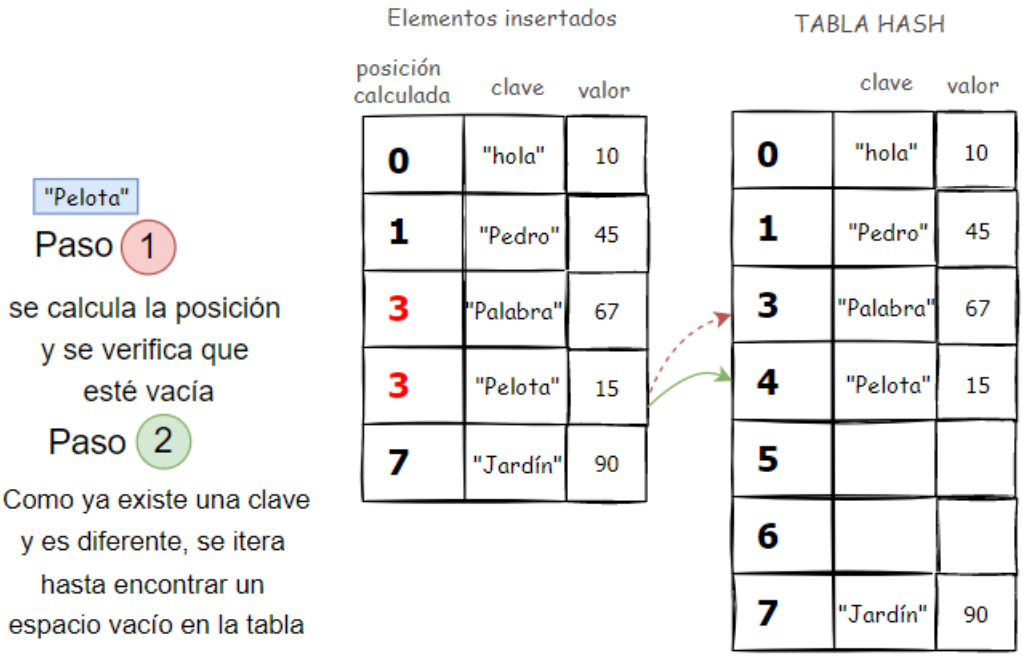


2.2. Tabla de hash cerrada

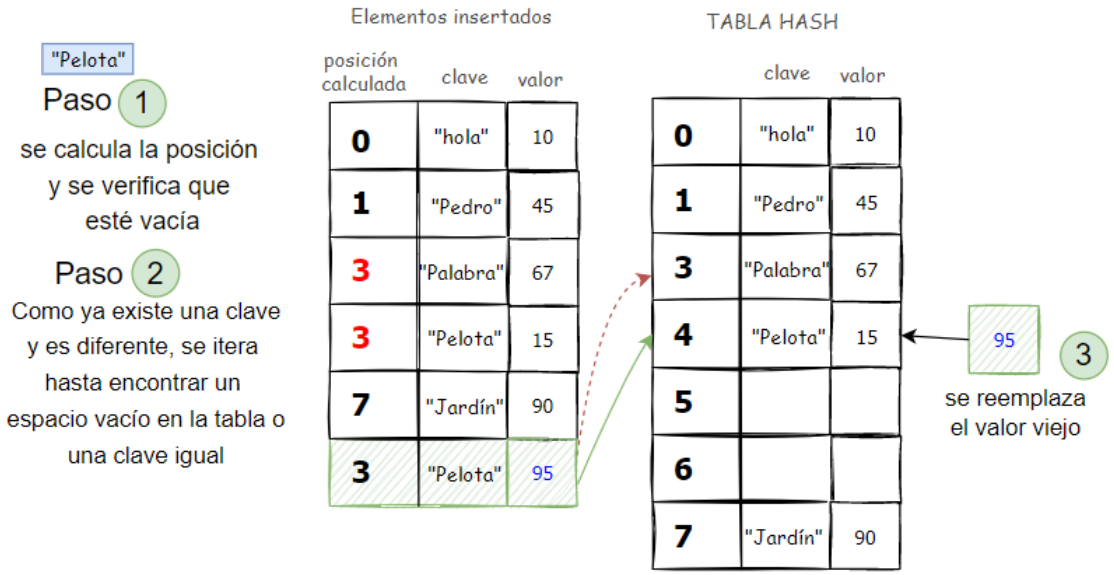
Las tablas de hash cerradas tienen por característica su direccionamiento abierto, esto se debe a la estrategia que se adopta en este tipo de hashes para las colisiones y que los pares se guardan por dentro de la tabla de hash, es decir que si se inserta una clave cuya posición calculada ya está ocupada en la tabla de hash, se *itera* sobre la misma hasta encontrar una posición vacía (Probing lineal). Por lo tanto, el direccionamiento abierto hace referencia a que los pares clave valor pueden variar de la posición calculada que les toca inicialmente, como se muestra en la figura:



A continuación, con diagramas se ilustra la operación de **inserción** con diagramas en este tipo de hash:



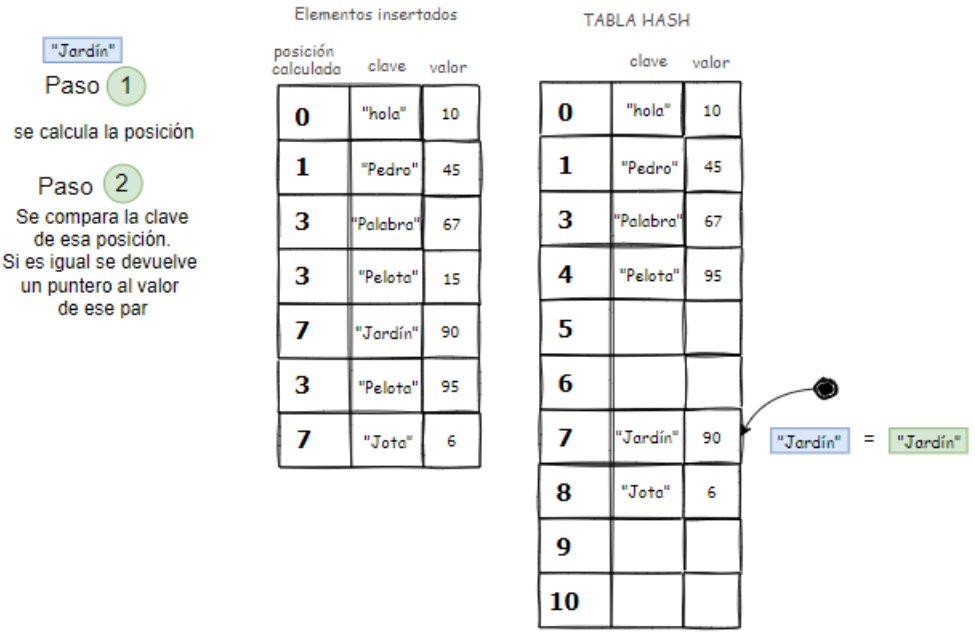
Si la clave ya existe en la tabla de hash, simplemente se reemplaza el valor por el nuevo:



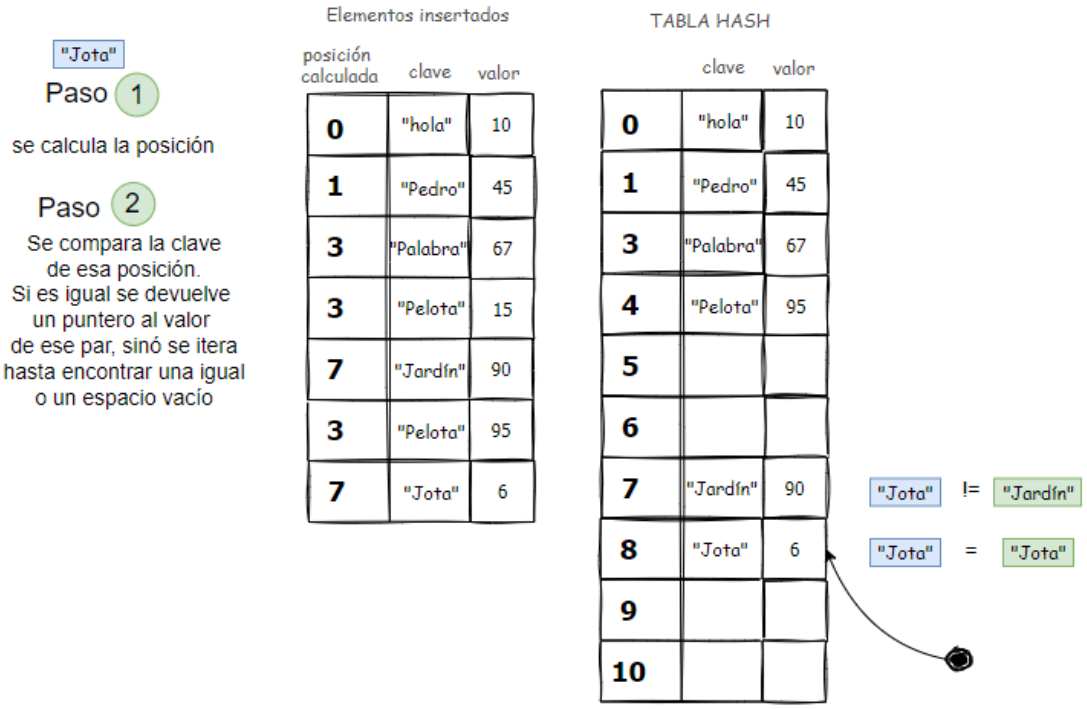
De la misma manera que con el hash abierto, para mantener la eficiencia deseada, se realiza la operación de **rehash** una vez que al querer insertar un par clave – valor se sobrepase el factor de carga establecido. Como vimos anteriormente, se aumenta el tamaño de la tabla y se reubican los

pares existentes en las ubicaciones nuevas que les corresponden de acuerdo al nuevo tamaño y nuevo cálculo de posición.

Vemos ahora la operación de **Búsqueda:**



Ahora busquemos “Jota”:



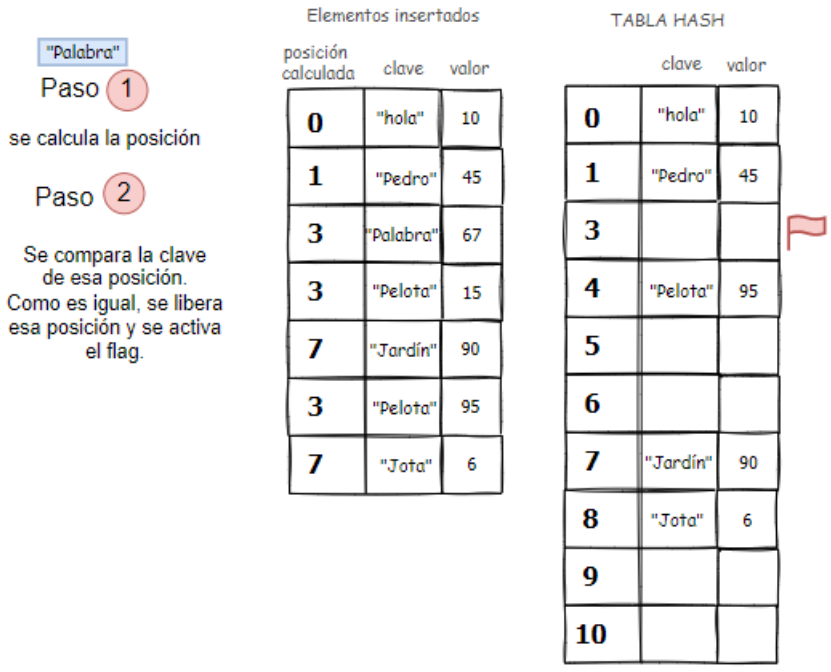
Si buscamos la clave “Jabón” y el calculo de posición de esta nos da 7, se iterará hasta la posición 9 (que es un espacio vacío) por lo tanto la búsqueda de una clave inexistente en el hash será infructuosa.

Avanzamos sobre la operación de **Eliminación** de pares clave – valor de un hash cerrado.

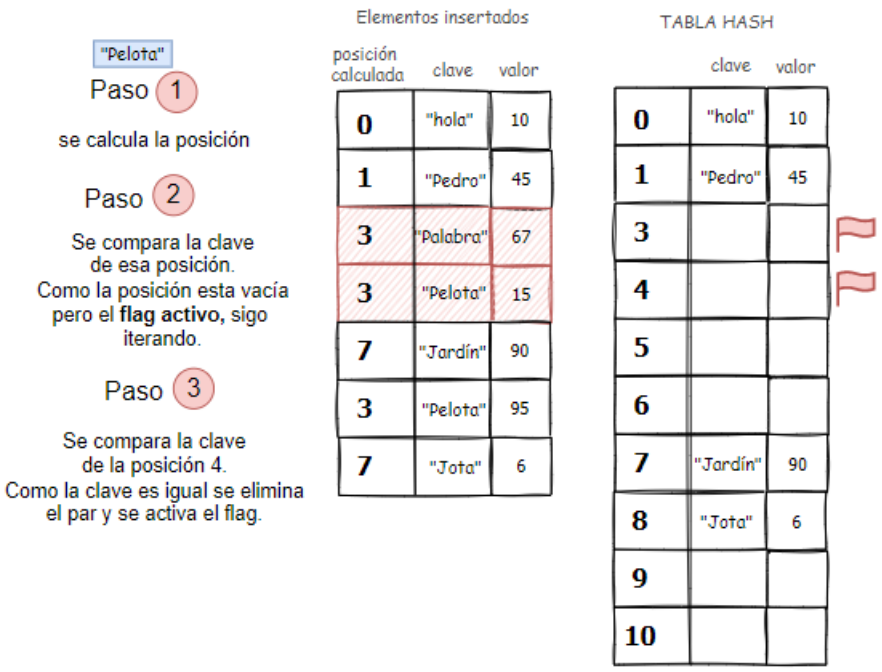
Para eliminar pares sin generar problemas en las operaciones posteriores del hash se tienen en cuenta dos métodos diferentes:

- **Reemplazar el espacio liberado con otro par ya insertado en el hash**. Este par reemplazante debe estar en posiciones contiguas a la posición liberada, sin posiciones vacías entre medio y que le “correspondiera” estar en esa posición liberada (mediante el cálculo de posición). Si efectivamente hay un reemplazante, se genera otro espacio vacío nuevo, por lo que se realiza el mismo procedimiento hasta que se encuentre un espacio vacío al buscar reemplazantes. Si el calculo de posición del posible reemplazante da un resultado que correspondiera a posiciones mayores que las de la posición primeramente liberada se lo deja en el lugar.
- **Utilizar un flag para indicar que se borró algo.** De esta manera si está vacía la posición igualmente se sigue iterando en cualquier operación. Al ser ocupada por un nuevo elemento en una inserción, el flag se setea nuevamente a su valor original.

Utilizando flags, saco la clave “palabra”:



Ahora si quiero buscar (o quitar) “pelota” tengo que hacer el mismo procedimiento que antes, saltando la posición vacía por que el flag está activado. Con el siguiente esquema se muestra como eliminar la clave “Pelota”:



3. Detalles de implementación

En primer lugar, además de los archivos provistos para la implementación, se creó un archivo .h adicional llamado “estructuras_y_auxiliares.h” que contiene las estructuras utilizadas y las firmas de las funciones auxiliares desarrolladas (junto con una breve explicación de su objetivo).
Se tomó la decisión de no utilizar implementaciones de TDAs previamente desarrollados como la Lista o el Árbol binario de búsqueda para simplificar el código y por otro lado para mantener las complejidades algorítmicas se necesitarían modificar estos TDAs con primitivas nuevas.

Entonces para guardar los pares clave – valor se utilizó una estructura llamada “par” que contiene un puntero a char para la clave, un puntero void para el valor (ya que puede ser cualquier cosa) y un puntero hacia otro par “siguiente” que se inicializa en null.
La estructura hash se definió utilizando un doble puntero a la estructura par previamente mencionada, es decir un vector de punteros a par. Adicionalmente se agregaron una variable numérica para la capacidad del vector y otra para la cantidad de pares insertados.

Al principio se implementó una función de hashing que sumaba los valores ascii de cada uno de los caracteres de la clave (string) a insertar y el resultado era ese código de hash. Ante el testeo realizado junto con las pruebas de chanutron’t quedó en evidencia que esta generaba una distribución de los pares mediocre.

Como consecuencia se modificó la función de tal manera que al empezar a calcular el código de una clave se empiece con un numero elevado (5381). Por cada carácter de la string, se realizaba la siguiente operación:

```
valor = ((valor << 5) + valor) + c;
```

que equivale a:

$$Valor = (Valor^5 + Valor) + (valor\ ascii\ del\ caracter\ actual)$$

De esta manera el valor de un carácter ascii, aunque no sea muy elevado si lo será el valor elevado a la quinta potencia al que previamente se le sumó ese carácter. Entonces al calcular la posición real en la tabla, la distribución es superior y el tiempo de ejecución mejoró notablemente.

Cabe destacar que antes de descubrir lo anterior, y con el motivo de mejorar el tiempo de ejecución, se cambiaron varias funciones auxiliares recursivas a su equivalente iterativo, logrando algunos segundos de mejoría en el testeo oficial de la cátedra.

Como criterio para el rehash se eligió un factor de carga máximo de 0,7 (70%) junto con una duplicación del tamaño de la tabla de Hash cada vez que se ejecuta esta operación.
Para evitar hacer reservas de memoria innecesarias, la metodología de rehash adoptada fue de trasladar los pares ya existentes a sus nuevas posiciones, solamente haciendo reserva de memoria dinámica para aumentar el tamaño de la tabla.
Para lo anterior se reserva, durante la ejecución de la función de rehash, el vector de punteros hacia las cadenas de pares en el Stack para que al modificar el tamaño de la tabla con la nueva reserva de memoria dinámica no se pierdan los pares ya insertados.