

22/04/2023

Actividad. – Hash Table

Estructura de Datos

Chávez Rodríguez Mauricio Yosef



Programa:

Crear un programa en Python que tenga las siguientes características:

- Crear la clase Hash.
- Crear el constructor.
- Crear la función Hash.
- Crear la función para agregar un elemento nuevo.

Explicación de Código:

```
class hash:
    def __init__(self):
        self.table = [None] * 23
```

En esta primera sección creamos la clase hash la cual es la requerida para la actividad, partimos de crear un constructor para la clase, la cual constará de un solo atributo, un arreglo con 23 espacios vacíos, número designado en base a una lógica que se explica más adelante.

```
def get_hash(self, key):
    # my_hash = (my_hash + ord(letter) *
    h = 0
    i = 1
    j = len(key)
    a = 7
    b = 11
    for char in key:
        h += (h * a + ord(char) * b * i)
        i += 1
    index = h % 23
    return index
```

Dentro de nuestra clase creamos distintos métodos siendo el primero el cual nos generará el índice donde nuestros datos se guardarán, la función get_hash consta de un acumulador (h), un contador (i), y 3 datos seleccionados bajo las siguientes reglas:

- Número primo más pequeño que el número de los datos (a).
- Número primo cercano a el número de los datos. (b).
- Número primo mayor que el número de los datos. (mod implementado)

Para generar los índices utilizamos un ciclo el cual realiza una serie de operaciones en conjunto de los números primos seleccionados, mientras va acumulando los resultados de cada iteración en la variable h, finalmente se hace un mod de lo generado por el ciclo y el resultado es el índice el cual es retornado por la función.

```
def add(self, key, edad):
    index = self.get_hash(key)
    if self.table[index] is None:
        self.table[index] = edad
        return True
    else:
        print("Colision...")
        return False
```

Para la función add encargada de agregar nuestros datos a la tabla hash, utilizamos el método antes mencionado (get_hash) para que nos genere un índice y saber donde posicionar nuestro valor, cuando se inicializó nuestro arreglo que guardaría nuestros datos se le asigno a los espacios el valor de None como predeterminado, esto con el fin de facilitar el conocer si dentro del índice ya existe un dato, pues de no existir agrega nuestro valor (edad),pero de existir nos advierte que existirá colisión.

```
def search(self, key):
    index = self.get_hash(key)
    if self.table[index] is None:
        print("No existe el Elemento ingresado")
        return False
    else:
        print(f"La edad de {key} es de : {self.table[index]}")
        return self.table[index]
```

La función search se encarga de buscar el dato que almacena una key especificada, para ello se le ingresa una key la cual será mandada al método (get_hash) donde se nos retornará el índice donde se encuentra su valor, con ello se verifica que exista un elemento dentro de la key ingresada, de no existir un elemento se nos avisa que no existe, de existir se nos muestra el elemento y se nos retorna.

```
def remove(self, key):
    index = self.get_hash(key)
    if self.table is None:
        print("No hay elemento para borrar")
    else:
        print("Eliminando elemento...")
        self.table[index] = None
```

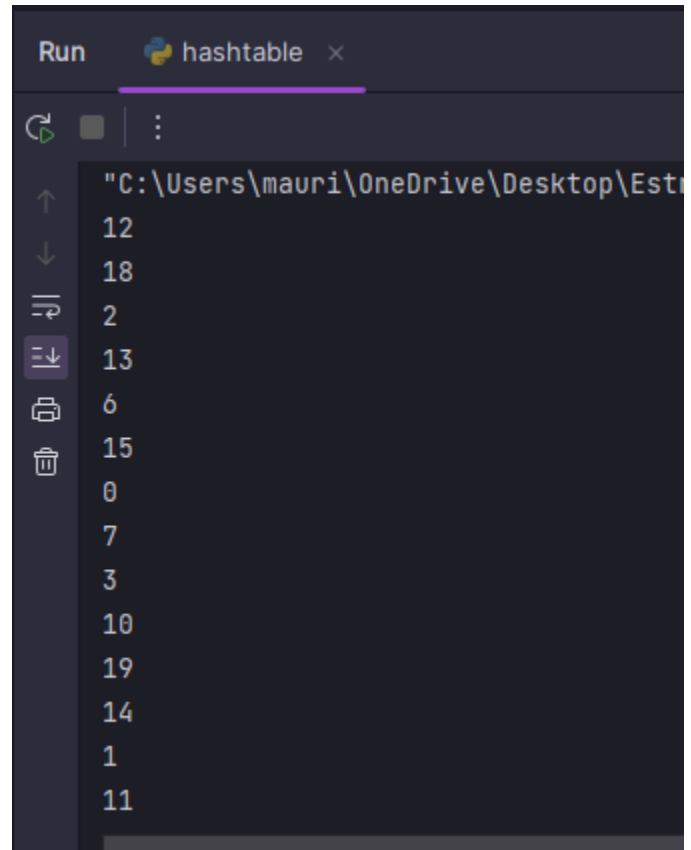
El método remove se encarga de borrar el valor almacenado de una key, su lógica es similar al search, pues primero se busca el índice de la key y después se corrobora que exista el dato, si no existe se avisa que no hay elemento para borrar, si existe se cambia el valor del índice especificado por None, borrando el dato.

Prueba de Escritorio:

```
print(h.get_hash("mario"))
print(h.get_hash("mauricio"))
print(h.get_hash("fernanda"))
print(h.get_hash("gael"))
print(h.get_hash("rafael"))
print(h.get_hash("julian"))
print(h.get_hash("samantha"))
print(h.get_hash("leag"))
print(h.get_hash("oiram"))
print(h.get_hash("adnanref"))
print(h.get_hash("leafar"))
print(h.get_hash("nailuj"))
print(h.get_hash("oicirvam"))
print(h.get_hash("ahtnamas"))
```

Para esta primera prueba se le mandaron directamente los datos a la función `get_hash` para corroborar que dicha función no genere colisiones al crear índices, como podemos observar en los datos impresos, ningún número se repite, por lo cual la generación de índice es correcta.

```
print(h.add("mauricio", 19))
h.search("mauricio")
h.remove("mauricio")
h.search("mauricio")
```



```
Run hashtable x
"C:\Users\mauri\OneDrive\Desktop\Est
12
18
2
13
6
15
0
7
3
10
19
14
1
11
```

```
True
La edad de mauricio es de : 19
Eliminando elemento...
No existe el Elemento ingresado
```

```
Process finished with exit code 0
```

Para esta última prueba se le añade el valor de 19 con la key mauricio, y realizamos una impresión del retorno de la función para observar que nos devolvió verdadero, lo cual nos indica que el dato ingresado se añadió de manera correcta, posteriormente se le indica a la función `search` que nos indique que dato se encuentra almacenado en la key mauricio, la cual nos indica que el valor de 19 se encuentra en dicha key, finalmente se remueve él lo almacenado en la key mauricio y se verifica con la función `search` que ya no exista nada en ese índice.