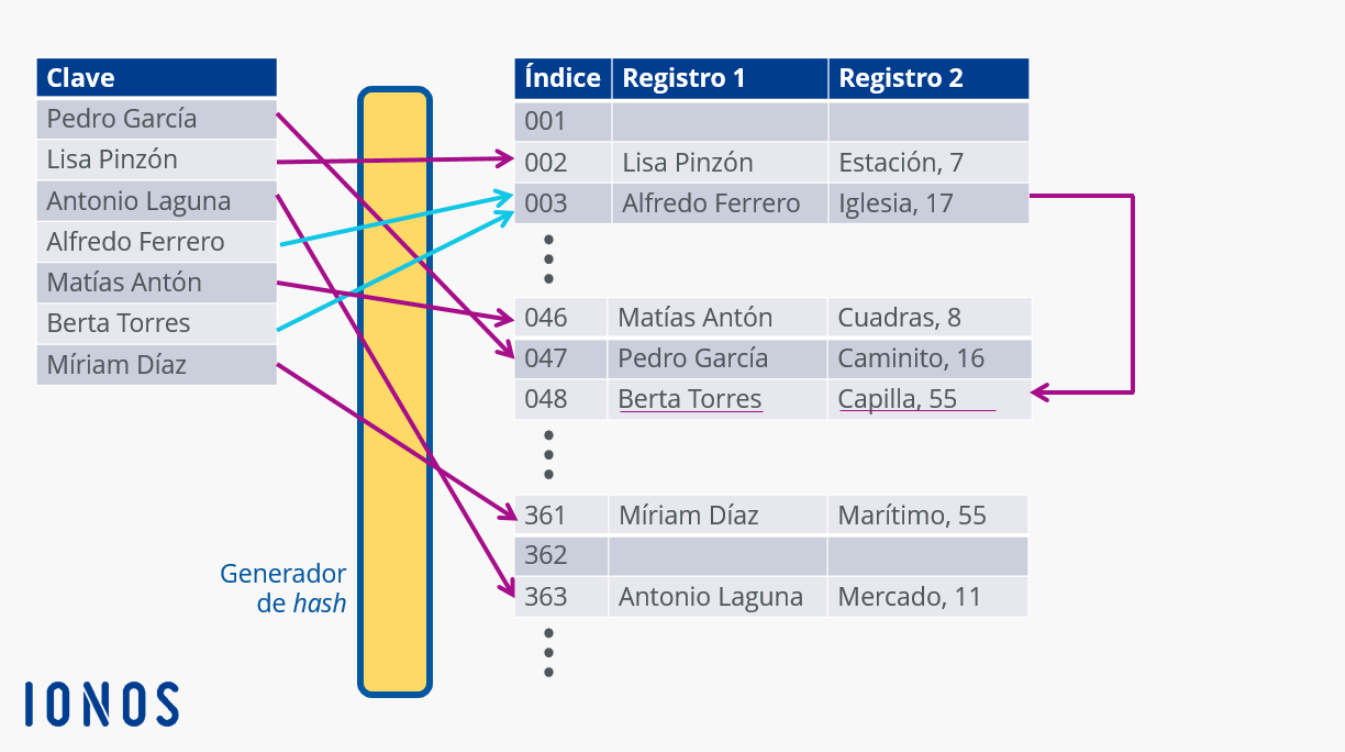
6/05/2023

Actividad. -*Tablas Hash*

Estructura de Datos

Chávez Rodríguez Mauricio Yosef



Programa:

- Crear la clase Hash.
- Crear el constructor.
- Crear la función Hash.
- Crear la función para agregar un elemento nuevo.
- Crear la función para obtener elemento.
- Crear la función para imprimir la tabla.
- Manejo de colisiones (Separate Chaining).

Explicación de Código:

```
class hash:
    def __init__(self,size):
        self.size=size
        self.table = [[] for _ in range(self.size)]
```

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 2 atributos uno centrado en el número de espacios generados para la tabla el cual es un arreglo con espacios vacíos, mientras que el otro atributo es la propia tabla donde se concentraran los datos, creada a partir del primer atributo mencionado.

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 datos o tamaño de la tabla.

(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

```
def get_hash(self, key):

    h = 0
    i = 1
    a = 7
    b = 11
    for char in key:
        h += (h * a + ord(char) * b * i)
        i += 1
    index = h % self.size
    return index
```

```python
def add(self, key, edad):
    index = self.get_hash(key)
    if self.table[index] is None:
        self.table[index] = []
    self.table[index].append([key, edad])
```

La función add es la encargada de agregar nuestros datos a la tabla hash, para ello implementa la función get_hash antes mencionado para que nos genere un índice donde ubicar nuestro dato, posteriormente se realiza una condición donde evaluamos el estado de la posición donde insertaremos nuestro dato, si se encuentra vacía crearemos una lista dentro de ese espacio, dicha lista con el objetivo de almacenar los datos que colisionen. Si no se encuentra vacía solo se insertará el dato en la última posición del arreglo.

```python
def get(self, key):
    index = self.get_hash(key)
    if self.table[index] is not None:
        for i in self.table[index]:
            if i[0] == key:
                return i[1]
    else:
        print(f"Key '{key}' no disponible")
        return False
```

La función get 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, una vez realizado se revisa con un if que realmente exista algo en dicha posición pues en caso de no existir se nos avisará, si existe algo en dicha posición se itera en ella hasta encontrar nuestra key para posteriormente retornar el valor en dicha key.

```python
def print_table(self):
    for i, v in enumerate(self.table):
        print(f"Indice {i}: {v}")
```

La funcion print_table como su nombre lo indica se encarga de imprimir todos los espacios de la tabla, es decir, su key junto a su valor para ello se usa un ciclo en conjunto de la función enumerate.

Prueba de Escritorio:

Para la prueba de escritorio se cambio el mod o el tamaño de la tabla a 15 para buscar una colisión y mediante el uso de nuestra función get_hash buscamos valores que generaran una colisión:

```
print(f"mauricio: {h.get_hash('mauricio')}")
print(f"fernanda: {h.get_hash('fernanda')}")
print(f"gael: {h.get_hash('gael')}")
print(f"rafael: {h.get_hash('rafael')}")
print(f"julian: {h.get_hash('julian')}")
print(f"samantha: {h.get_hash('samantha')}")
print(f"leag: {h.get_hash('leag')}")
print(f"oiram: {h.get_hash('oiram')}")
print(f"adnanref: {h.get_hash('adnanref')}")
print(f"leafar: {h.get_hash('leafar')}")
print(f"nailuj: {h.get_hash('nailuj')}")
print(f"oiciruam: {h.get_hash('oiciruam')}")
print(f"ahtnamas: {h.get_hash('ahtnamas')}")
```

```
mauricio: 2
fernanda: 4
gael: 8
rafael: 13
julian: 0
samantha: 14
leag: 4
oiram: 8
adnanref: 10
leafar: 4
nailuj: 3
oiciruam: 5
ahtnamas: 8
```

Como se puede observar en la imagen existen 3 valores que generan colisión (gael, samantha y oiram), conociendo esto usaremos dichos valores para nuestra prueba, mandamos dichos datos a nuestra tabla hash, junto a un dato que no genere colisión y pedimos sus valores (edades) junto al valor de una key no existente en la lista y como se puede observar en la imagen los datos retornados coinciden con los datos que fueron guardados en la tabla hash, a excepción del dato que no existe en la tabla.

```
h.add("ahtnamas",20)
h.add("gael",19)
h.add("oiram",19)
h.add("mauricio",18)
print("-----------------")
print(h.get("ahtnamas"))
print(h.get("gael"))
print(h.get("oiram"))
print(h.get("mauricio"))
h.get("Mau")
```

```
20
19
19
18
Key 'Mau' no disponible
```

Finalmente imprimimos la tabla completa para observar lo que se encuentra guardado dentro de ella:

```
h.add("ahtnamas",20)
h.add("gael",19)
h.add("oiram",19)
h.add("mauricio",18)

h.print_table()
```

```
Indice 0: None
Indice 1: None
Indice 2: [['mauricio', 18]]
Indice 3: None
Indice 4: None
Indice 5: None
Indice 6: None
Indice 7: None
Indice 8: [['ahtnamas', 20], ['gael', 19], ['oiram', 19]]
Indice 9: None
Indice 10: None
Indice 11: None
Indice 12: None
Indice 13: None
Indice 14: None
```