Carátula para entrega de prácticas

Facultad de Ingeniería | Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Jorge Solano
Asignatura:	Estructura de Datos y Algoritmos II
Grupo:	2
No de Práctica(s):	5
Integrante(s):	Martínez Ostoa Néstor Iván
Semestre:	2019-1
Fecha de entrega:	19/Septiembre/2018
Observaciones:	

CALIFICACIÓN: _____

Práctica 5: Algoritmos de búsqueda Parte 2

Objetivo

Diseñar una función hash propia para la implementación de un algoritmo de búsqueda por transformación de llaves sobre un conjunto no ordenado de nodos.

Desarrollo:

Se implementó una función Hash, que es una función que recibe un entero como parámetro y devuelve otro entero, con la intención de obtener el menor número de colisiones sobre un conjunto de Nodos (id, nombreCiudad). Una colisión ocurre cuando al aplicar una función Hash sobre dos números distintos, esta función retorna el mismo valor. El valor de retorno de una función Hash representa la posición que ocupará cierto elemento, o Nodo en nuestro caso, dentro de una tabla Hash. Esta tabla Hash nos representará una colección de números donde se guardará la información de un cierto conjunto.

Recordemos que estamos implementando una búsqueda por transformación de llaves, lo que nos lleva a analizar lo siguiente:

1. **Desarrollar** una función hash -> hash(int) : int
2. **Insertar** un conjunto 'n' de nodos dentro de una tabla Hash ocupando la función Hash para obtener el número de posición de tal Nodo.
3. Para evitar problemas con las colisiones, que por cierto, siempre se tienen que **manejar**, nuestra tabla Hash será una lista de listas. De esta forma, cuando encontremos una colisión, el algoritmo de inserción agregará un Nodo a la lista de la casilla 'n' de la tabla Hash.
4. **Buscar** dentro de la tabla Hash al elemento deseado. Para esto, indicaremos el id del nodo, despúes, obtendremos mediante la aplicación del función Hash, la posición que ocupa dentro de la tabla Hash dicho nodo. Posteriormente, iteramos sobre la lista en la casilla 'n' de la tabla Hash hasta encontrar el Nodo deseado.

Código - implementación en Python del Encadenamiento

1. Función hash propuesta:

```
def hashNestor(number):
    position = ""
    number = str(number)
    lista = list(number)
    for x in range(0,3):
        position += number[random.randint(0,len(number)-1)]
    listaDos = list(position)
    rz = random.randint(1,10)
    if rz < 6:
        if int(lista[0]) == 1 or int(lista[0]) == 2 or int(lista[0]) == 3:
            rr = random.randint(4,9)
            listaDos[0] = rr
    else:
        ra = random.randint(0,2)
        if int(lista[ra]) == 1 or int(lista[ra]) == 2 or int(lista[ra]) == 3:
            listaDos[ra] = random.randint(4,9)

    ps = ""
    for i in range(len(listaDos)):
        ps += str(listaDos[i])
    return int(ps)
```

2. Algoritmo de inserción

```
def insertIntoHashTable(listOfItems, hashTable):
    for x in range(len(hashTable)):
        key = hashNestor(listOfItems[x].id)
        if hashTable[key] is None:
            hashTable[key] = [listOfItems[x]]
        else:
            hashTable[key].append(listOfItems[x])
```

3. Algoritmo de búsqueda

```
def searchInHashTable(item, hashTable):
    global counter
    numberToLookAt = hashNestor(item)
    if hashTable[numberToLookAt] is None:
        print("NONE")
    else:
        for x in range(len(hashTable[numberToLookAt])):
            counter += 1
            if hashTable[numberToLookAt][x].id == item:
                return (hashTable[numberToLookAt][x].city )
        print("Did not found any city with this " + str(item) + " id.")
```

4. Otros algoritmos útiles para el desarrollo de la práctica

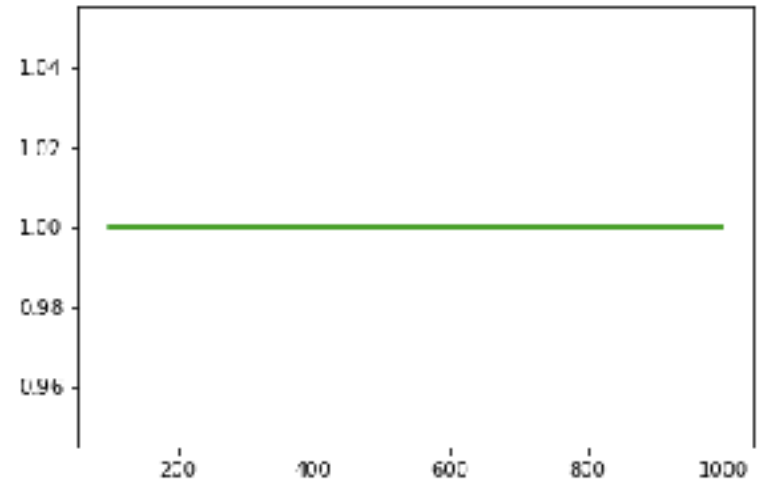
```
def generateHashTable():
    hash_table = [None] * TAM
    return hash_table

def printHashTable(hashTable):
    for i in range(len(hashTable)):
        if hashTable[i] is not None:
            print(str(i) + " -> " + str(len(hashTable[i])))
            for j in range(len(hashTable[i])):
                print("[" + str(hashTable[i][j].id) + " : " + hashTable[i][j].city + "]")
```

Prueba de que funciona + gráficas

1. Mejor Caso -> cuando buscamos siempre un elemento que se encuentre en la primer casilla en lista dentro de la tabla Hast en una posición arbitraria.

```
print("Best case\n")
counter = 0
city = searchInHashTable(2791975824, hashTable)
print("City ->" + city)
print("Number of iterations -> " + str(counter))
x = []
y = []
for i in range(len(hashTable)):
    counter = 0
    if hashTable[i] is not None:
        x.append(i)
        searchInHashTable(hashTable[i][0].id,hashTable)
        y.append(counter)
plt.plot(x, y, 'g')
plt.show()
```

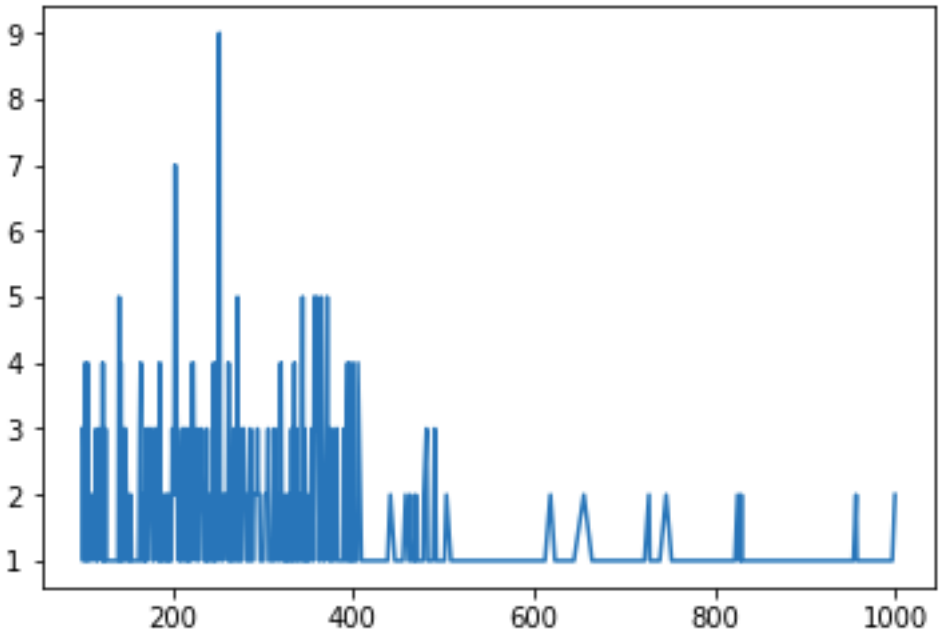
```
Best case

City ->Sacramento
Number of iterations -> 1
```



2. Caso promedio -> cuando buscamos un elemento de manera aleatoria dentro de una lista perteneciente a la tabla Hash en una posición arbitraria.

```
print("Average case\n")
counter = 0
city = searchInHashTable(2563475534, hashTable)
print("City -> " + city)
print("Number of iterations -> " + str(counter))
x = []
y = []
for i in range(len(hashTable)):
    counter = 0
    if hashTable[i] is not None:
        x.append(i)
        randomElement = random.randint(0,len(hashTable[i])-1)
        searchInHashTable(hashTable[i][randomElement].id,hashTable)
        y.append(counter)
plt.plot(x, y)
plt.show()
```

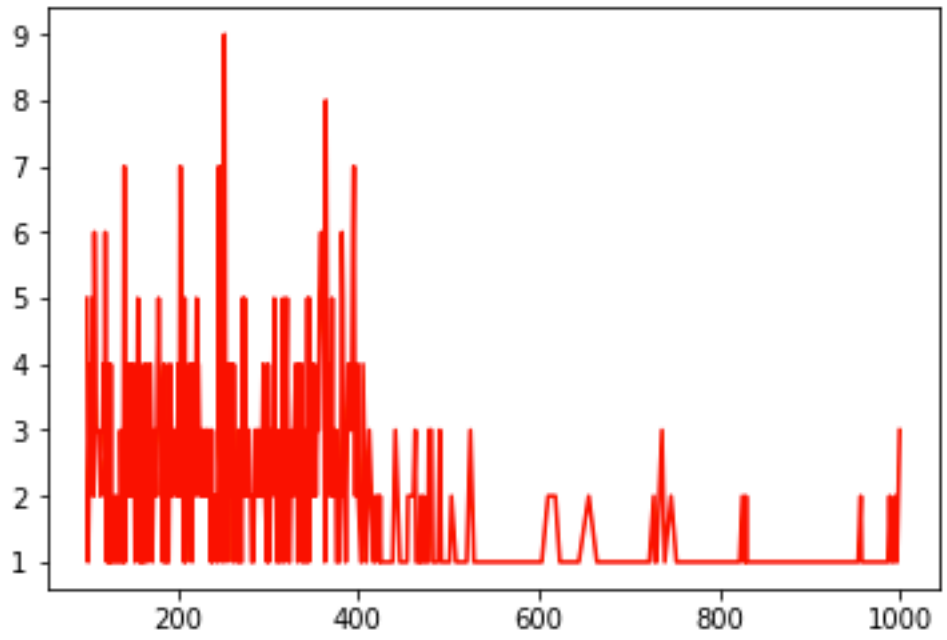
```
Average case

City -> Pocpo
Number of iterations -> 3
```



3. Peor caso -> cuando buscamos al último elemento dentro de una lista perteneciente a la tabla Hash en una posición arbitraria.

```
print("Worst case\n")
counter = 0
city = searchInHashTable(2408077994, hashTable)
print("City ->" + city)
print("Number of iterations -> " + str(counter))
x = []
y = []
for i in range(len(hashTable)):
    counter = 0
    if hashTable[i] is not None:
        x.append(i)
        searchInHashTable(hashTable[i][len(hashTable[i])-1].id,hashTable)
        y.append(counter)
plt.plot(x, y, 'r')
plt.show()
```

```
Worst case

City ->Ninh Giang
Number of iterations -> 5
```



Código - implementación en Python de Arreglos Anidados

1. Algoritmo de inserción

```
def insertIntoHashTable(listOfItems, arrayIn2D):
    for k in range(TAM):
        key = hashNestor(listOfItems[k].id)
        if arrayIn2D[key][0] is None:
            arrayIn2D[key][0] = listOfItems[k]
        else:
            positionToInsert = 0
            while(arrayIn2D[key][positionToInsert] is not None):
                positionToInsert += 1
            arrayIn2D[key][positionToInsert] = listOfItems[k]
```

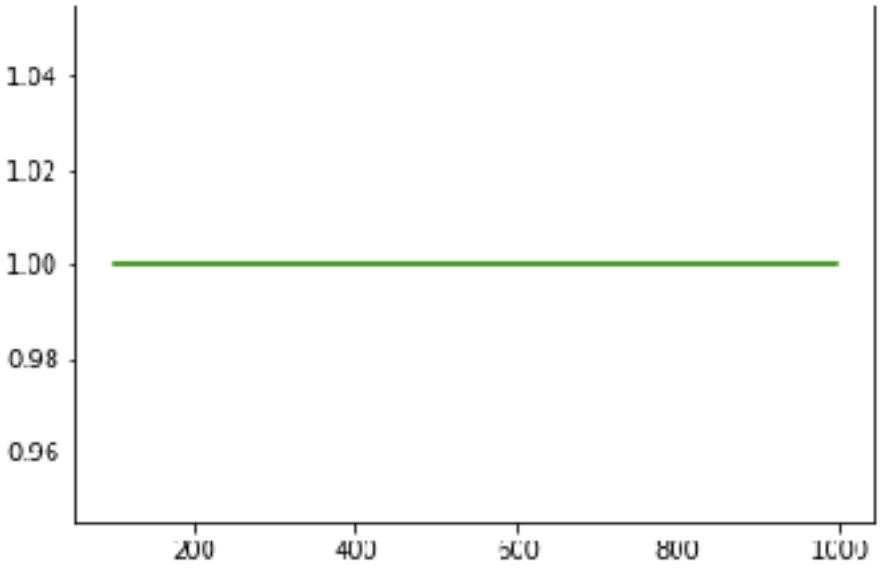
2. Algoritmo de búsqueda:

```
def searchInHashTable(element, arrayIn2D):
    global counter
    key = hashNestor(element)
    if arrayIn2D[key][0] is None:
        counter = 1
        return 'City not found'
    else:
        counter = 0
        positionToLookAt = 0
        while (arrayIn2D[key][positionToLookAt] is not None):
            counter += 1
            if arrayIn2D[key][positionToLookAt].id == element:
                return 'City found : ' + str(arrayIn2D[key][positionToLookAt].city)
            positionToLookAt += 1
```

3. Gráficas

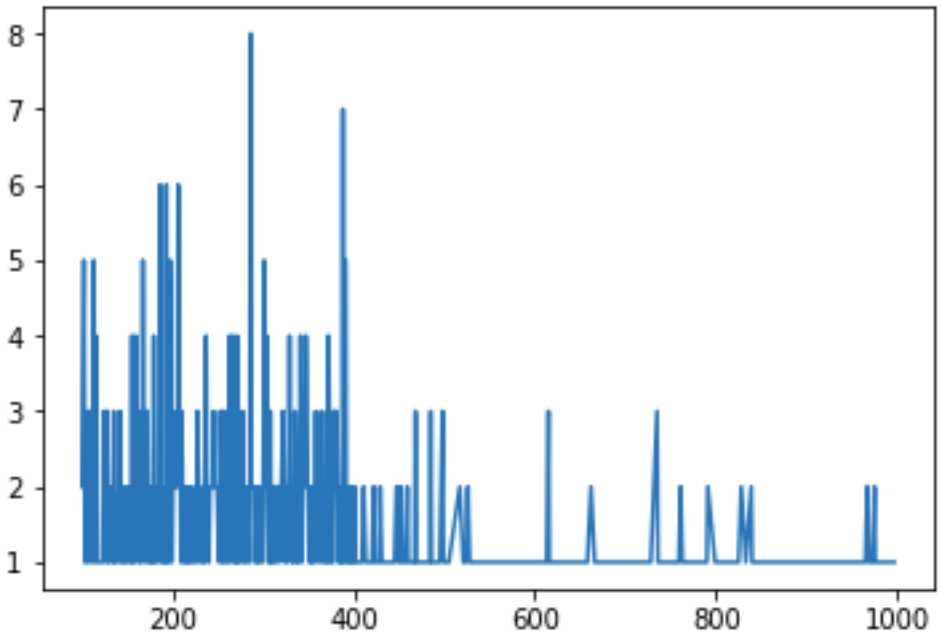
3.1 Gráfica del mejor caso

```
print("Best Case")
x = []
y = []
counter = 0
for i in range(TAM):
    if hashTable[i][0] is not None:
        x.append(i)
        counter = 0
        searchInHashTable(hashTable[i][0].id,hashTable)
        y.append(counter)

print(len(x))
print(len(y))
plt.plot(x, y, 'g')
plt.show()
```



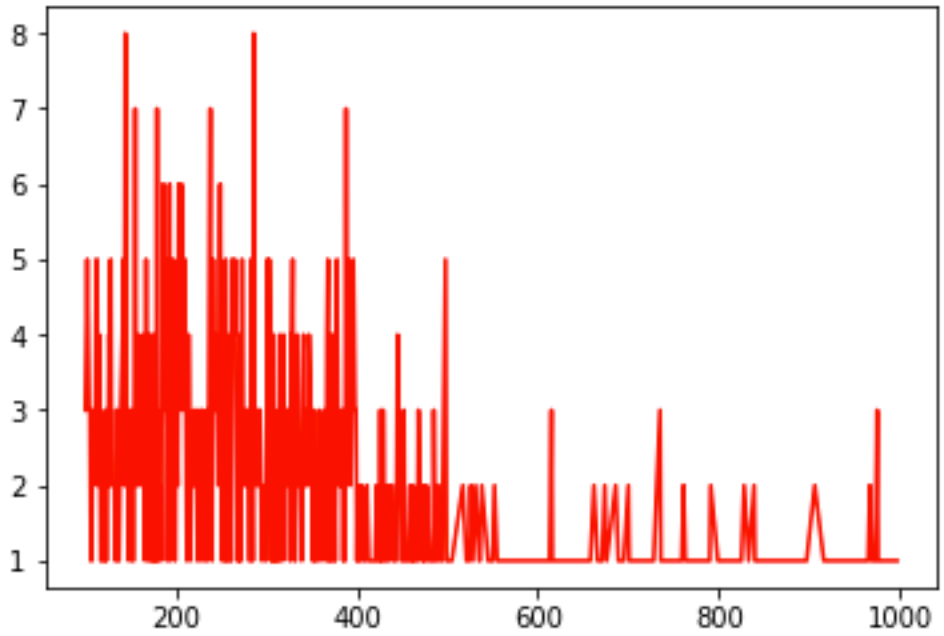
3.2 Gráfica del caso promedio

```
print("Average Case")
x = []
y = []
counter = 0
for i in range(TAM):
    if hashTable[i][0] is not None:
        x.append(i)
        pos = 0
        for j in range(TAM):
            if hashTable[i][j] is not None:
                pos += 1
        randomPos = random.randint(0, pos-1)
        counter = 0
        searchInHashTable(hashTable[i][randomPos].id, hashTable)
        y.append(counter)


print(len(x))
print(len(y))
plt.plot(x, y)
plt.show()
```



3.3 Gráfica del peor caso

```
print("Worst Case")
x = []
y = []
counter = 0
for i in range(TAM):
    if hashTable[i][0] is not None:
        x.append(i)
        pos = 0
        for j in range(TAM):
            if hashTable[i][j] is not None:
                pos += 1
        counter = 0
        searchInHashTable(hashTable[i][pos-1].id, hashTable)
        y.append(counter)


print(len(x))
print(len(y))
plt.plot(x, y, 'r')
plt.show()
```



Conclusiones

La búsqueda por transformación de llaves es bastante eficiente a la hora de buscar/ insertar, puesto que el tiempo de búsqueda/inserción es casi siempre constante. Sin lugar a dudas nos vemos restringidos por la función hash y el manejo de colisiones, pero cuando queremos buscar un elemento, bastará con recorrer solo una lista dentro de una cantidad grande de listas. Esto presenta una gran ventaja en comparación con una búsqueda lineal.

Analizando las gráficas propuestas:

1. Mejor caso: observamos que es constante puesto que siempre estamos buscando al primer elemento de la lista y obtener la posición de búsqueda dentro de la tabla Hash es siempre constante pues nos los da una función Hash.

2. Caso promedio: observamos un comportamiento variable debido a que nosotros no controlamos qué elemento queremos buscar. Sin embargo, sí hemos asegurado que el elemento exista, por lo que la variación de las gráficas está determinado por el número de iteraciones que el algoritmo de búsqueda tiene que hacer sobre una lista en una determinada posición de la tabla Hash.

3. Peor caso: en esta gráfica no podemos obtener un caso peor puesto que nos representa como se comporta el algoritmo cada que recorre todos los elementos de una lista en una posición determinada de la tabla Hash.