Algoritmos y Estructuras de Datos II

Hash Tables

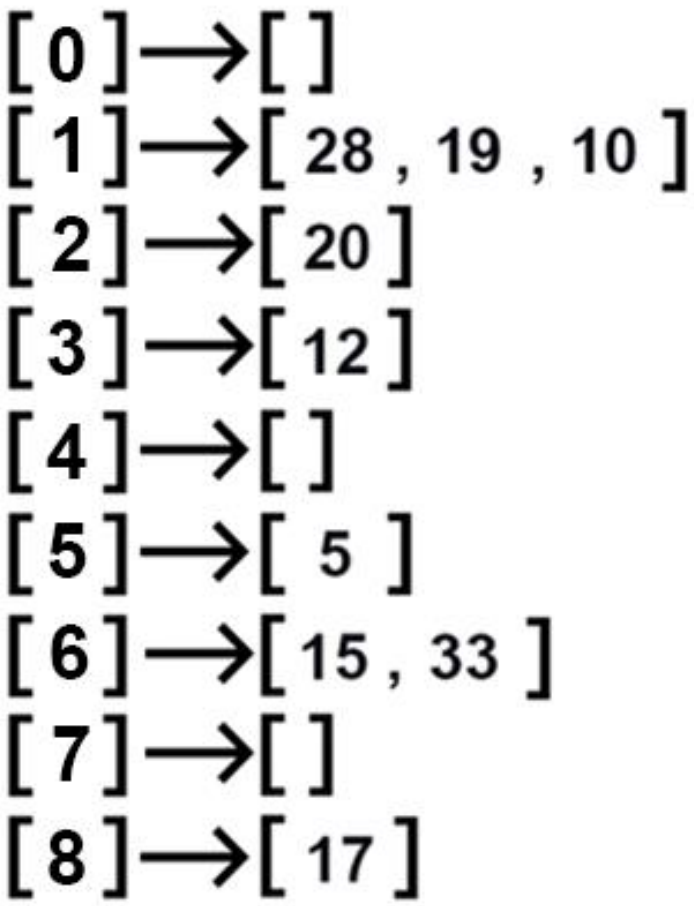
Link Repl.it: https://replit.com/@Paulonia/hashtables#dictionary.py

PARTE 1:

Ejercicio 1

Ejemplificar que pasa cuando insertamos las llaves 5, 28, 19, 15, 20, 33, 12, 17, 10 en un **HashTable** con la colisión resulta por el método de chaining. Permita que la tabla tenga 9 slots y la función de hash:

$$H(k) = k \bmod 9 \qquad (1)$$



=

Ejercicio 2

A partir de una definición de diccionario como la siguiente:

```
dictionary = Array(m,0)
```

Crear un módulo de nombre **`dictionary.py`** que **implemente** las siguientes especificaciones de las operaciones elementales para el **TAD diccionario** .

Nota: puede **`dictionary`** puede ser redefinido para lidiar con las colisiones por encadenamiento

insert(D,key, value)

Descripción: Inserta un key en una posición determinada por la función de hash (1) en el diccionario (dictionary). Resolver colisiones por encadenamiento. En caso de keys duplicados se anexan a la lista.
Entrada: el diccionario sobre el cual se quiere realizar la inserción y el valor del key a insertar
Salida: Devuelve D

search(D,key)

Descripción: Busca un key en el diccionario
Entrada: El diccionario sobre el cual se quiere realizar la búsqueda (dictionary) y el valor del key a buscar.
Salida: Devuelve el value de la key. Devuelve **None** si el key no se encuentra.

delete(D,key)

Descripción: Elimina un key en la posición determinada por la función de hash (1) del diccionario (dictionary)
Poscondición: Se debe marcar como nulo el **key** a eliminar.
Entrada: El diccionario sobre el se quiere realizar la eliminación y el valor del key que se va a eliminar.
Salida: Devuelve **D**

```python
def hash(key,m):
    #Función de Hash usada
    return key % m

#D = dictionary (lista)
def insert(D,key,value):
    #Inserta un key en una posición determinada por la función de hash
    #Uso la funcion hash para obtener la posición
    if D==None or key==None:
        return "Error"
    position=hash(key,len(D))
    #Creamos una tupla con key y value
    tupla=(key,value)
    if D[position]==None:
        #Si no hay nada en esa posición, va a crear una lista vacía y meter la tupla
        newList=[]
        newList.append(tupla)
        D[position]=newList
    else:
        D[position].append(tupla)

def search(D,key):
    #Busca un key en el diccionario
    position=hash(key,len(D))
    i=0
    for i in range (0,len(D[position])):
        if key==D[position][i][0]:
            return D[position][i][1]
```

```python
def delete(D,key):
  #Elimina un key en la posición determinada por la función de hash
  position=hash(key,len(D))
  for i in range(0,len(D[position])):
    if key==D[position][i][0]:
      D[position].pop(i)
      return D
```

Extra:

```python
def printHash(D):
  #Imprime toda la tabla Hash
  for i in range(0,len(D)):
    print(f"[{i}] -> {D[i]}")
```

PARTE 2:

Ejercicio 3

Considerar una tabla hash de tamaño m = 1000 y una función de hash correspondiente al método de la multiplicación donde A = (sqrt(5)-1)/2). Calcular las ubicaciones para las claves 61,62,63,64 y 65.

Key 61 = **700**

Key 62 = **318**

Key 63 = **936**

Key 64 = **554**

Key 65 = **172**

Ejercicio 4

Implemente un algoritmo lo más eficiente posible que devuelva **True** o **False** a la siguiente proposición: dado dos strings $s_1...s_k$ y $p_1...p_k$, se quiere encontrar si los caracteres de $p_1...p_k$ corresponden a una permutación de $s_1...s_k$. Justificar el coste en tiempo de la solución propuesta.

Ejemplo 1:
Entrada: S = 'hola' , P = 'ahlo'
Salida: True, ya que P es una permutación de S

Ejemplo 2:
Entrada: S = 'hola' , P = 'ahdo'
Salida: Falso, ya que P tiene al carácter 'd' que no se encuentra en S por lo que no es una permutación de S

```python
#Ejercicio 4
def isPermutation(cadenaS,cadenaP):
    if len(cadenaS)!=len(cadenaP):
        #Si son de distinta longitud la cadenaP no puede ser una permutacion
        return False
    size=len(cadenaS)
    D=createHash(size)
    for each in cadenaS:
        #Meto los elementos de la primera cadena en un dictionary
        insert(D,ord(each),each)
    for each in cadenaP:
        #Busco los elementos de la segunda cadena en el dictionary
        sameV=search(D,ord(each))
        if sameV==None:
            return False
    return True
```

Ejercicio 5

Implemente un algoritmo que devuelva True si la lista que recibe de entrada tiene todos sus elementos únicos, y Falso en caso contrario. Justificar el coste en tiempo de la solución propuesta.

Ejemplo 1:
Entrada: L = [1,5,12,1,2]
Salida: Falso, L no tiene todos sus elementos únicos, el 1 se repite en la 1ra y 4ta posición

```python
# Ejercicio 5
def hasUniqueElements(L):
    #devuelve True si la lista que recibe de entrada tiene todos sus
    # elementos únicos, sino False
    D=createHash(len(L))
    #Creo un dictionary y voy metiendo valores de la lista en él,
    # para buscarlos en ese hash para ver si ya están. Si están, retorna False, sino True.
    for i in range(0,len(L)):
        sameV=search(D,L[i])
        insert(D,L[i],L[i])
        if sameV!=None and sameV==L[i]:
            return False
    return True
```

Ejercicio 6

Los nuevos códigos postales argentinos tienen la forma cddddccc, donde c indica un carácter (A - Z) y d indica un dígito 0, . . . , 9. Por ejemplo, C1024CWN es el código postal que representa a la calle XXXX a la altura 1024 en la Ciudad de Mendoza. Encontrar e implementar una función de hash apropiada para los códigos postales argentinos.

```python
#Ejercicio 6
def hashPostales(key,m):
    #los códigos postales son de la forma cddddccc (c=caracter; d=dígito)
    #ord() para dar el ascii de un valor
    return ( ord(key[0])*(10**7) + key[1]*(10**6) + key[2]*(10**5) + key[3]*(10**4)
+ key[4]*(10**3) + ord(key[5])*(10**2) + ord(key[6])*10 + ord(key[7]) ) % m
```

Ejercicio 7

Implemente un algoritmo para realizar la compresión básica de cadenas utilizando el recuento de caracteres repetidos. Por ejemplo, la cadena '*aabccccccaaa*' se convertiría en 'a2blc5a3'. Si la cadena "comprimida" no se vuelve más pequeña que la cadena original, su método debería devolver la cadena original. Puedes asumir que la cadena sólo tiene letras mayúsculas y minúsculas (a - z, A - Z). Justificar el coste en tiempo de la solución propuesta.

Ejercicio 8

Se requiere encontrar la primera ocurrencia de un string $p_1...p_k$ en uno más largo $a_1...a_L$. Implementar esta estrategia de la forma más eficiente posible con un costo computacional menor a $O(K*L)$ (solución por fuerza bruta). Justificar el coste en tiempo de la solución propuesta.

Ejemplo 1:
Entrada: S = 'abracadabra' , P = 'cada'
Salida: 4, índice de la primera ocurrencia de P dentro de S (abracadabra)

Ejercicio 9

Considerar los conjuntos de enteros $S = \{s1, \ldots, sn\}$ y $T = \{t1, \ldots, tm\}$. Implemente un algoritmo que utilice una tabla de hash para determinar si $S \subseteq T$ (S subconjunto de T). ¿Cuál es la complejidad temporal del caso promedio del algoritmo propuesto?

PARTE 3:

Ejercicio 10

Considerar la inserción de las siguientes llaves: 10; 22; 31; 4; 15; 28; 17; 88; 59 en una tabla hash de longitud m = 11 utilizando direccionamiento abierto con una función de hash *h'(k)* = *k*. Mostrar el resultado de insertar estas llaves utilizando:

1. Linear probing
2. Quadratic probing con *c1* = *1* y c2 = 3
3. Double hashing con *h1(k)* = *k* y *h2(k)* = *1 +(k mod (m - 1))*

1) Linear probing

22	88			4	15	28	17	59	31	10
0	1	2	3	4	5	6	7	8	9	10

2) Quadratic probing con c1=1 y c2=3

22		88	17	4		28	59	15	31	10
0	1	2	3	4	5	6	7	8	9	10

3) Double hashing

22		59	17	4	15	28	88		31	10
0	1	2	3	4	5	6	7	8	9	10

Ejercicio 11 (opcional)

Implementar las operaciones de **`insert()`** y **`delete()`** dentro de una tabla hash vinculando todos los nodos libres en una lista. Se asume que un slot de la tabla puede almacenar un indicador (flag), un valor, junto a una o dos referencias (punteros). Todas las operaciones de diccionario y manejo de la lista enlazada deben ejecutarse en O(1). La lista debe estar doblemente enlazada o con una simplemente enlazada alcanza?

Ejercicio 12

Las llaves 12, 18, 13, 2, 3, 23, 5 y 15 se insertan en una tabla hash inicialmente vacía de longitud 10 utilizando direccionamiento abierto con función hash h(k) = k mod 10 y exploración lineal (linear probing). ¿Cuál es la tabla hash resultante? Justifique.

0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

(A)

0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

(B)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(C)

0	
1	
2	12, 2
3	13, 3, 23
4	
5	5, 15
6	
7	
8	18
9	

(D)

Es la tabla **(C)** ya que a las tablas A y B le faltan llaves y la D no utiliza lineal probing.

Ejercicio 13

Una tabla hash de longitud 10 utiliza direccionamiento abierto con función hash h(k)=k mod 10, y exploración lineal (linear probing). Después de insertar 6 valores en una tabla hash vacía, la tabla es como se muestra a continuación.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

¿Cuál de las siguientes opciones da un posible orden en el que las llaves podrían haber sido insertadas en la tabla? Justifique

(A) 46, 42, 34, 52, 23, 33
(B) 34, 42, 23, 52, 33, 46
(C) 46, 34, 42, 23, 52, 33
(D) 42, 46, 33, 23, 34, 52

A)

		42	52	34	23	46	33		
0	1	2	3	4	5	6	7	8	9

B)

		42	23	34	52	33	46		
0	1	2	3	4	5	6	7	8	9

C)

		42	23	34	52	46	33		
0	1	2	3	4	5	6	7	8	9

D)

		42	33	23	34	46	52		
0	1	2	3	4	5	6	7	8	9

La correcta es la **(C)**