

Algoritmo y Estructura de Datos II

Alumno:Gonzalez Sanchez Gabriel
Legajo: 12007

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 resuelta por el método de chaining. Permita que la tabla tenga 9 slots y la función de hash:

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

Algoritmo y Estructuras de datos 2

Tip Hash table

Parte 1

① Ejemplificar que pasa cuando insertamos las llaves 5, 28, 19, 15, 20, 33, 12, 17, 10 en un HashTable con la colisión resuelta por el método de chaining. Permita que la tabla tenga 9 slots y la función de Hash:

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

②

Primero se creará la tabla con 9 posiciones

H	
0	
1	10 → 19 → 28 → None
2	20
3	12
4	5
5	
6	33 → 15 → None
7	
8	17

(b) Luego al insertar una llave, esta es pasada por la función hash

insert(H, 5)

$$H(5) = 5 \bmod 9 = 5$$

luego la llave es insertada en la posición que devuelve el hash.

$$H(28) = 28 \bmod 9 = 1$$
$$H(19) = 19 \bmod 9 = 1$$

Cuando 2 llaves coinciden en su posición se resuelve creando una lista en la posición.

Los elementos de la lista siempre son de key única y se insertan en la cabecera O(1).

$$H(15) = 15 \bmod 9 = 6 \text{ 3 colisiones}$$
$$H(20) = 20 \bmod 9 = 2$$
$$H(33) = 33 \bmod 9 = 6 \text{ 3 colisiones}$$
$$H(12) = 12 \bmod 9 = 3$$
$$H(17) = 17 \bmod 9 = 8$$
$$H(10) = 10 \bmod 9 = 1 \text{ 3 colisiones}$$

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**

#Defino el diccionario como un Array de nodos dictionary

```
def Dictionary(m):  
    return Array(m,dictionary())
```

#Insert

#Inserta un value a través de una key en un diccionario D

#Devuelve el diccionario con la insercion

#Las colisiones se resuelven por encadenamiento

```
def insert(D,key,value):  
    h = hash(key,len(D))  
    if D[h]==None:  
        L = dictionary()  
        D[h] = L  
  
    newNode = dictionaryNode()  
    newNode.key = key  
    newNode.value = value  
    newNode.nextNode = D[h].head  
    D[h].head = newNode  
    return D
```

#Utilizo una llave y un modulo para aplicar metodo de la division

#Segun el tipo de dato(numerico,caracter o string) aplica diferentes calculos

```
def hash(key,module):  
    if type(key)==int:  
        return key % module  
    elif type(key)==str:  
        return ord(key) % module  
    elif type(key) == String:  
        n = len(key)  
        for i in range(0,n):  
            if i==0:  
                j = n-1  
                newkey = ord(key[i])*(255**j)  
            else:  
                j -= 1  
                newkey += ord(key[i])*(255**j)  
        return newkey % module
```

#Search

#Busca la key en el diccionario D

#Devuelve el value asociado a la key o None si no lo encuentra

```
def search(D,key):
    h = hash(key,len(D))
    currentNode = D[h].head
    while currentNode!=None:
        if currentNode.key==key:
            return currentNode.value
        currentNode = currentNode.nextNode
    return None
```

#Delete

#Elimina una Key en un diccionario D

#Devuelve el diccionario

```
def delete(D,key):
    h = hash(key,len(D))
    #Caso 1: Posicion vacia
    if D[h] == None:
        return D
    else:
        currentNode = D[h].head
        #Caso 2: El key esta en la cabeza de la lista
        #Si se vacia una lista se busca que el slot de el diccionario quede vacio
        if currentNode.key == key and currentNode.nextNode == None:
            D[h] = None
        #Caso 3: Hay colisiones en la posicion
        else:
            currentNode=D[h].head
            #Caso elemento en el primer nodo
            if currentNode.key==key:
                D[h].head=currentNode.nextNode
            elif currentNode.nextNode!=None:
                flag = True
                while currentNode!=None and flag:
                    if currentNode.nextNode==None:
                        flag = False
                    elif currentNode.nextNode.key==key:
                        currentNode.nextNode=currentNode.nextNode.nextNode
                        flag = False
                currentNode=currentNode.nextNode
    return D
```

#Actualiza el value de un nodo en un dictionary

```

def update(D, key, value):
    m = len(D)
    currentNode = D[h(key, m)]
    if currentNode == None:
        return None
    else:
        currentNode = currentNode.head
        while currentNode != None:
            if currentNode.key == key:
                currentNode.value = value
                return value
            currentNode = currentNode.nextNode
        return None

```

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.

```

#Ejercicio 3
def ex3():
    i = 61
    while i <= 65:
        A = ((1000** (1/2)) - 1)/2
        print(int(6 * ((i * A) % 1)))
        i += 1

```

Parte 2

Ejercicio 3: Considerar una tabla hash de tamaño $m = 1000$ y una función hash correspondiente al método de la multiplicación donde $A = (\sqrt{5} - 1)/2$. Calcular la ubicación para las claves 61, 62, 63, 64 y 65.

$$h(k) = \lfloor m(k \cdot A \bmod 1) \rfloor \rightarrow \text{parte entera de } \text{int}(x) \text{ de python}$$

$$h(k) = \text{int}((1000 \cdot (k \cdot (\sqrt{5} - 1)/2) \bmod 1))$$

$$h(61) = 700$$

$$h(62) = 318$$

$$h(63) = 936$$

$$h(64) = 554$$

$$h(65) = 172$$

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.

#Ejercicio 4

#Determina si un string es permutacion de otro

#Su tiempo de ejecucion es $O(m^2)$ siendo m la longitud de cadena

#Seria más optimo si delete retornara True o False en lugar de D

#De esta forma podriamos detener la ejecucion si no logra eliminar un caracter de string2 en D

```
def isPermutation(string1,string2):
    n = len(string1)
    m = len(string2)

    #Check de ancho de palabra
    if m == n:
        D = Dictionary(m)
        #inserto string 1 en D
        for i in range(0,len(string1)):
            insert(D,string1[i],string1[i])

        #Elimino string 2 en D
        for i in range(0,len(string2)):
            delete(D,string2[i])

        flag = True
        for i in range(0,len(D)):
            if D[i]!=None:
                flag = False
        if flag:
            return flag
    return False
```


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

```
#Ejercicio 5
#Determina si una lista L posee elementos unicos
#Busca el elemento en D, si existe previamente devuelve False
def Unique(D,L):
    if L == None:
        return
    else:
        for i in range(0,len(L)):
            flag = search(D,L[i])
            if flag == None:
                insert(D,L[i],L[i])
            else:
                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.

```
def postalHash(k,m):
    cityreference = 0
    streetreference = ''
    count1 = 3
    for i in range(0,8):
        if (i<=0 or i>=5):
            print(ord(k[i])-ord('A'))
            print(10**count1)
            cityreference = cityreference + (ord(k[i])-ord('A'))*(100**count1)
            count1 -= 1
        else:
            streetreference += k[i]

    keystr = str(cityreference) + streetreference
    key = int(keystr)
    #Utilizo metodo de la multiplicacion con A = ((5**(1/2))-1)/2
    A = ((5**(1/2))-1)/2
    return int(m*((key*A)%1))
```