Aida Laricchia

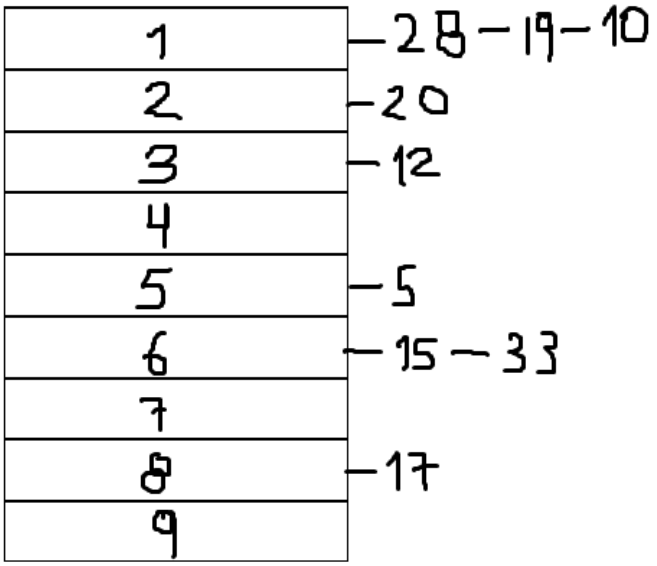
PARTE 1

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:

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

HASH:



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**

```
def hash_mode(k,m):
    return (k%m)

def CreateHashTable(Dim):
    Hash=[]
    #crea un Hash de M posciones
    for i in range (0,Dim):
        L=[]
        Hash.append(L)
    return Hash

def Insert(D,key,value):
    if len(D)==0 or D==None:
        print("crear tabla hash")
        return None
    else:
        index=hash_mode(key,len(D))
        if D[index]==None:
            list=[]
            tupla=(key,value)
            list.append(tupla)
            D[index]=list
        else:
            tupla=(key,value)
            D[index].append(tupla)

def search(D,key):
```

```
    index=hash_mode(key,len(D))
    for elemento in D[index]:
        if elemento[0]==key:
            return (elemento[1])

def delete (D,key):
    if search(D,key)!=None:
        index=hash_mode(key,len(D))
        for i in range (len(D[index])):
            if D[index][i][0]==key:
                D[index].pop(i)
                return D
```

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.
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.

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

```
def string_permutation(s,p):
    if len(s)!=len(p):
        return False
    else:
        dicS=[None]*(ord("z")-ord("a"))
        dicP=[None]*(ord("z")-ord("a"))
        for i in range(len(s)):
            dicS.Insert(hash_mode(ord(s[i])-ord("a"),len(dicS)),s[i])
        for i in range(len(p)):
            dicP.Insert(hash_mode(ord(p[i])-ord("a"),len(dicP)),p[i])
        for i in range(len(dicS)):
            if dicS[i] !=None:
                if len(dicS[i])!=len(dicP[i]):
                    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 (DETECTAR COLISION)

```
def list_repetidos(L):
    D=[None]*len(L)
    for i in range (len(L)):
        if search(D,L[i]) !=None:
            return False
        Insert(D,L[i],L[i])
    return True
```

El coste de tiempo es O(n^2) ya que entra al bucle n veces y debe realizar un search en el diccionario, el cual tiene un costo de O(n). Las operaciones de insert en el diccionario son O(1)

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.

(FOTO PIZARRON) HACER FUNCION HASH

```
def hash_code(code):
    for i in range(len(code)):
        if code[i].isdigit():
            num=num+code[i]
        else:
            num=num+ord(code[i])
    return(hash_mode(num,len(code)))
```

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.

```
def compre_char(cadena):
    letra=cadena[0]
    cadena_nueva=''
    cont=1
    for i in range (len(cadena)):
        if i !=0:
            if letra==cadena[i]:
                cont=cont+1
            else:
                cadena_nueva=cadena_nueva+letra+str(cont)
                cont=1
                letra=cadena[i]
    cadena_nueva += letra + str(cont)
    if (len(cadena_nueva)<(len(cadena))):
        return cadena_nueva
    else:
        return cadena
```

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)

```
def ocurrencia (cadena,subcadena):
    dictionary=[]
    for i in range (len(cadena)-len(subcadena)+1):
        sublista=[]

        for j in range (len(subcadena)):
            sublista.insert(j,cadena[i+j])
        tupla=(sublista,i)
        #if sublista == list(subcadena):
            #print(i)
        dictionary.insert(hash_subcadena(sublista,len(cadena)),tupla)
    haski_p=hash_subcadena(subcadena,len(cadena)) #key de la subcadena
    print(dictionary)
    print((dictionary[haski_p][1]))
    return(dictionary[haski_p][1]) #devuelve la posicion de la cadena donde se
encnotro la subacadena
```

El costo de tiempo es O(n^2) dado por los for anidados. Las operaciones de insert en el diccionario son O(1)

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 ⊆ T (S subconjunto de T). ¿Cuál es la complejidad temporal del caso promedio del algoritmo propuesto?
Mejor caso=caso promedio=peor caso=O(s)

```
def conjuntos(S,T):
    hash=CreateHashTable(len(T))
    for numeroT in T:
        Insert(hash,hash_mode(numeroT,len(T)),numeroT)
    for numeroS in S:
        if search(hash,hash_mode(numeroS,len(T))) != numeroS:
```

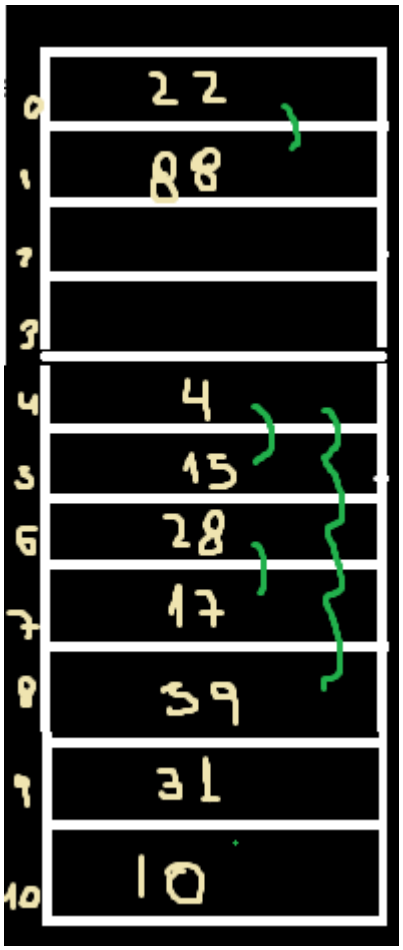
```
            return False
    return True
```

Parte 3

Ejercicio 10

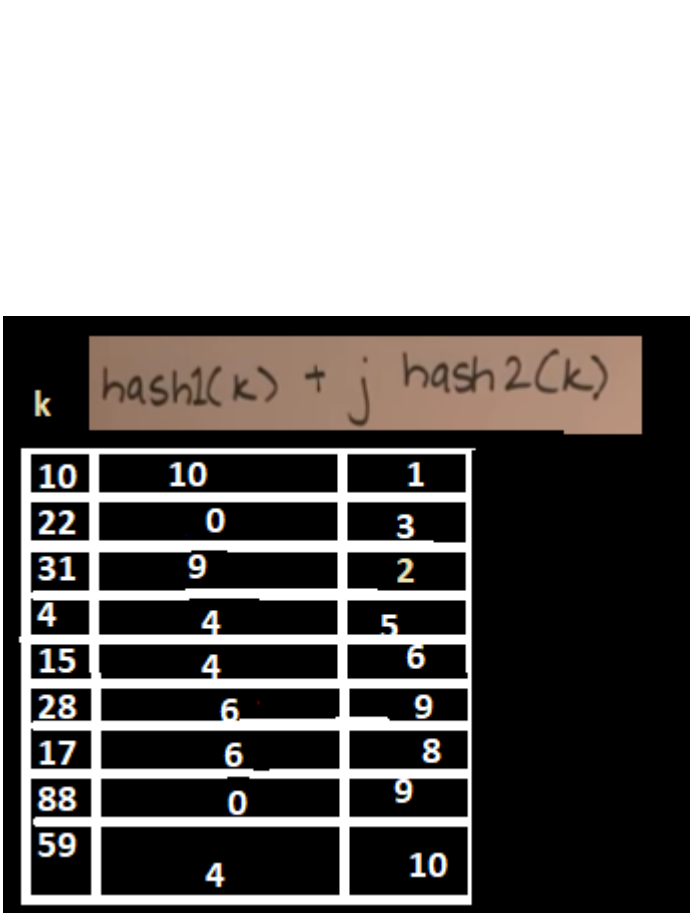
Considerar la inserción de las siguientes llaves: 10; 22; 31; 4; 15; 28; 17; 88; 59 en una tabla hash de longitud m = 11 utilizando direccionamiento abierto con una función de hash *h'(k)* = *k*. Mostrar el resultado de insertar estas llaves utilizando:

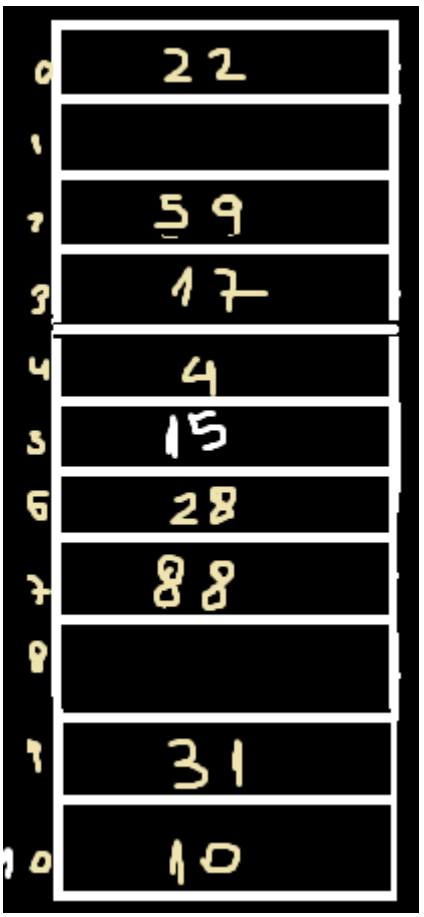
1. Linear probing



2. Quadratic probing con *c1* = *1* y c2 = 3

3. Double hashing con *h1(k) = k* y *h2(k) = 1 +(k mod (m - 1))*



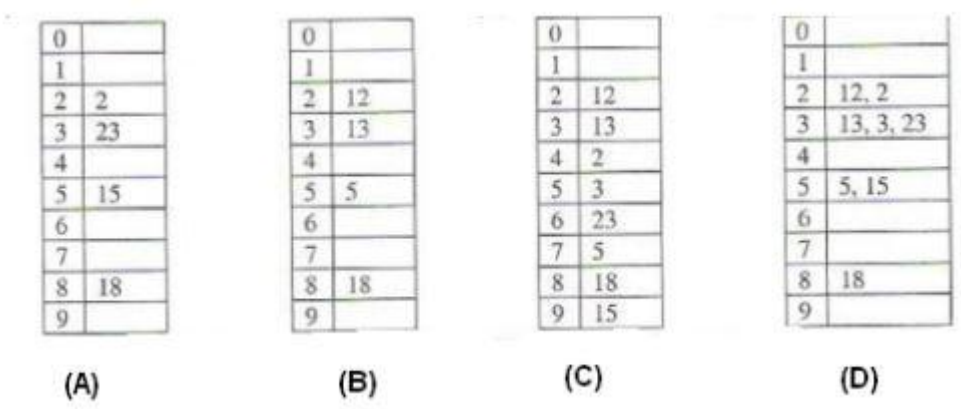


Ejercicio 11 (opcional)

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)

La respuesta correcta es C es visible al insertar la llave 2 donde h(2)=2 el siguiente casillero(el 3) está lleno pero el 4 no entonces insertamos ahí.

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
46 va en el 6
42 va en 2
34 va en 4
52 va a 3 pq 2 esta ocupada
Queda descartada la opción A
B
34 va en 4
42 va en 2
El 23 va en 3
52 va en 5 pq los anteriores estaban ocupados
33 va en 6
Queda descartada la opción B
C
46 va en el 6
34 va a 4
42 va a 2
23 va en 3
52 va en 5 pq los anteriores estaban ocupados
33 va en 7 pq los anteriores estaban ocupados
La C es nuestra respuesta correcta.

A tener en cuenta:

1. **Usen lápiz y papel primero**
2. ~~**No se puede utilizar otra Biblioteca mas allá de algo1.py y las bibliotecas desarrolladas durante Algoritmos y Estructuras de Datos I.**~~