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:

0	
1	28 -> 19 -> 10
2	20 ->
3	12 ->
4	
5	5 ->
6	15 -> 33 ->
7	
8	17 ->

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

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

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

def add_hash(current,node):
    if current.key == node.key and current.value == node.value:
        current.count += 1

    elif current.nextNode is None:
        current.nextNode=node
    else:
        if current.key != node.key and current.value != node.value:
            add_hash(current.nextNode,node)

def insert(D,key, value):
    m=len(D.head) # El diccionario sera una lista array y el encadenamiento
sera de tipo linkedLit
    if m > 0 :
        node=dictionarynode()
        node.value=value
        node.key=key
        idx=h(key,m)
        if D.head[idx] is None:
            D.head[idx]=node
        elif D.head[idx] is not None:
            if D.head[idx].key != key and D.head[idx].value != value:
                add_hash(D.head[idx],node)
            else:
                D.head[idx].count += 1
    return 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.

```
def google(current,key):
    if current!=None:
        if current.key == key:
            return current.value
        else:
            return google(current.nextNode,key)
    return None
def search(D,key):
    m=len(D.head)
```

```
    if m > 0 :
        idx=h(key,m)
        return google(D.head[idx],key)
    return None
```

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 delete_key(current,key):
    if current.nextNode != None:
        if current.nextNode.key==key:
            current.nextNode=current.nextNode.nextNode
        else:
            return delete_key(current.nextNode,key)

def delete(D,key):
    m=len(D.head)
    if m > 0 :
        idx = h(key,m)
        if D.head[idx] is not None:
            if D.head[idx].key == key:
                D.head[idx]= D.head[idx].nextNode
                return D
            else:
                delete_key(D.head[idx],key)
    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.

$H(61) = 1000 \times (61 \times ((\sqrt{5}-1) \div 2) - 37) = 700$
$H(62) = 1000 \times (62 \times ((\sqrt{5}-1) \div 2) - 38) = 318$
$H(63) = 1000 \times (63 \times ((\sqrt{5}-1) \div 2) - 38) = 936$
$H(64) = 1000 \times (64 \times ((\sqrt{5}-1) \div 2) - 39) = 554$
$H(65) = 1000 \times (65 \times ((\sqrt{5}-1) \div 2) - 40) = 172$

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 update_count(current, key, value):
    if current is None:
        return
    if current.key == key and current.value == value:
        current.count -= 1
    else:
    update_count(current.nextNode, key, value)
    return current.count

def delete_count(T,key,char):
    m=len(T.head)
    idx=h(key,m)
    if T.head[idx] is None:
        return None
    else:
        if update_count(T.head[idx],key,char) == 0:
            delete(T,key)

def permutation(list1,list2):
    if len(list1) != len(list2):
        return False
    list1=list1.lower()
    list2=list2.lower()
    T=dictionary()
    T.head=[None]*len(list1)
    for char in list1:
        insert(T,ord(char),char)
    for char in list2:
        delete_count(T,ord(char),char)
    for node in T.head:
```

```
            if node is not 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

```
def list_unico(lst):
    m=len(lst)
    S=dictionary()
    S.head=[None]*m
    for key in lst:
        if search(S,key) == key:
            return False
        insert(S,key,key)
    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.

1.Toma los primeros 3 caracteres de la cadena
2.Asigna un numero único ala combinación de 3 cadenas
3.Multiplica por 1000 (o cualquier otro valor) para darle más dispersión

```
def hash_codigo_postal(codigo_postal):
    trio= codigo_postal[:3].upper()
    key=0
    for char in trio:
        key += ord(char)
    key *=1000
    return key
```

Ejercicio 7

Implemente un algoritmo para realizar la compresión básica de cadenas utilizando el recuento de caracteres repetidos. Por ejemplo, la cadena '*aabccccaaa'* 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 compres_string(list):
    if len(list)>1:
        comprimido=[]
        cont=1
        for n in range(1,len(list)):
            if list[n] == list[n-1]:
                cont +=1
            else:
                comprimido.append(list[n-1]+str(cont))
                cont=1
    comprimido.append(list[n-1]+str(cont))
    comprimido=''.join(comprimido)
    if len(list)<=len(comprimido):
        return list
    return comprimido
```

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 Karp_key(s,primo):
    key=0
    for n in range (0,len(s)):
        key += ord(s[n])*(primo**n)
    return key

def Rabin_Karp(p,a):
    k=len(p)
    l=len(a)
    primo=3
    if k>l:
```

```
        cadena=p[:l]
        key = Karp_key(cadena,primo)
        key_a = Karp_key(a,primo)
        for n in range(l,k):
            key = (key-ord(cadena[0]))/primo + ord(p[n])*(primo**(l-1))
            cadena = cadena[1:]+p[n]
            if key_a == key and a == cadena:
                return n-l+1
    return -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?

```
def is_subset(S,T):
    Q=dictionary()
    Q.head=[None]*len(S)
    for element in S:
        insert(Q,element,element)
    for m in T:
        if search(Q,m) is None:
            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

   $h(k,i) = (h'(k) + i)\ mod\ m = (k + i)\ mod\ 11$

   $h(10,0) = (10 + 0)\ mod\ 11 = 10$
   $h(22,0) = (22 + 0)\ mod\ 11 = 0$
   $h(31,0) = (31 + 0)\ mod\ 11 = 9$
   $h(4,0) = (4 + 0)\ mod\ 11 = 4$
   $h(15,0) = (15 + 0)\ mod\ 11 = 4$
   $h(15,1) = (15 + 1)\ mod\ 11 = 5$
   $h(28,0) = (28 + 0)\ mod\ 11 = 6$
   $h(17,0) = (17 + 0)\ mod\ 11 = 6$
   $h(17,1) = (17 + 1)\ mod\ 11 = 7$
   $h(88,0) = (88 + 0)\ mod\ 11 = 0$

$h(88,1) = (88 + 1)\ mod\ 11 = 1$

$h(59,0) = (59 + 0)\ mod\ 11 = 4$

$h(59,0) = (88 + 1)\ mod\ 11 = 5$

$h(59,0) = (88 + 2)\ mod\ 11 = 6$

$h(59,0) = (88 + 3)\ mod\ 11 = 7$

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

$h(k,i) = (h'(k) + C_1 * i + C_2 * i^2)\ mod\ m = (k + i + 3 * i^2)\ mod\ 11$

$h(10,0) = (10 + 0 + 3 * 0^2)\ mod\ 11 = 10$

$h(22,0) = (22 + 0 + 3 * 0^2)\ mod\ 11 = 0$

$h(31,0) = (31 + 0 + 3 * 0^2)\ mod\ 11 = 9$

$h(4,0) = (4 + 0 + 3 * 0^2)\ mod\ 11 = 4$

$h(15,0) = (15 + 0 + 3 * 0^2)\ mod\ 11 = 4$

$h(15,1) = (15 + 1 + 3 * 1^2)\ mod\ 11 = 8$

$h(28,0) = (28 + 0 + 3 * 0^2)\ mod\ 11 = 6$

$h(17,0) = (17 + 0 + 3 * 0^2)\ mod\ 11 = 6$

$h(17,1) = (17 + 1 + 3 * 1^2)\ mod\ 11 = 10$

$h(17,2) = (17 + 2 + 3 * 2^2)\ mod\ 11 = 9$

$h(17,3) = (17 + 3 + 3 * 3^2)\ mod\ 11 = 3$

$h(88,0) = (88 + 0 + 3 * 0^2)\ mod\ 11 = 0$

$h(88,1) = (88 + 1 + 3 * 1^2)\ mod\ 11 = 4$

$h(88,2) = (88 + 2 + 3 * 2^2)\ mod\ 11 = 3$

$h(88,3) = (88 + 3 + 3 * 3^2)\ mod\ 11 = 8$

$h(88,4) = (88 + 4 + 3 * 4^2)\ mod\ 11 = 8$

$h(88,5) = (88 + 5 + 3 * 5^2)\ mod\ 11 = 3$

$h(88,6) = (88 + 6 + 3 * 6^2)\ mod\ 11 = 4$

$h(88,7) = (88 + 7 + 3 * 7^2)\ mod\ 11 = 0$

$h(88,8) = (88 + 8 + 3 * 8^2)\ mod\ 11 = 2$

$h(59,0) = (59 + 0 + 3 * 0^2)\ mod\ 11 = 4$

$h(59,1) = (59 + 1 + 3 * 1^2)\ mod\ 11 = 8$

$h(59,2) = (59 + 2 + 3 * 2^2)\ mod\ 11 = 7$

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

$h(k,i) = (h_1(k) + i * h_2(k))\ mod\ m = (k + i * (1 + (k\ mod\ 10\ )))\ mod\ 11$

$h(10,0) = \big(10 + 0 * \big(1 + (10\ mod\ 10\ )\big)\big)\ mod\ 11 = 10$

$h(22,0) = \big(22 + 0 * \big(1 + (22\ mod\ 10\ )\big)\big)\ mod\ 11 = 0$

$h(31,0) = \big(31 + 0 * \big(1 + (31\ mod\ 10\ )\big)\big)\ mod\ 11 = 9$

$h(4,0) = \big(4 + 0 * \big(1 + (4\ mod\ 10\ )\big)\big)\ mod\ 11 = 4$

$h(15,0) = \big(15 + 0 * \big(1 + (15\ mod\ 10\ )\big)\big)\ mod\ 11 = 4$

$$h(15,1) = \left(15 + 1 * \left(1 + (15 \ mod \ 10\ )\right)\right) mod\ 11 = 10$$

$$h(15,2) = \left(15 + 2 * \left(1 + (15 \ mod \ 10\ )\right)\right) mod\ 11 = 5$$

$$h(28,0) = \left(28 + 0 * \left(1 + (28 \ mod \ 10\ )\right)\right) mod\ 11 = 6$$

$$h(17,0) = \left(17 + 0 * \left(1 + (17 \ mod \ 10\ )\right)\right) mod\ 11 = 6$$

$$h(17,1) = \left(17 + 1 * \left(1 + (17 \ mod \ 10\ )\right)\right) mod\ 11 = 3$$

$$h(88,0) = \left(88 + 0 * \left(1 + (k \ mod \ 10\ )\right)\right) mod\ 11 = 0$$

$$h(88,1) = \left(88 + 1 * \left(1 + (88 \ mod \ 10\ )\right)\right) mod\ 11 = 6$$

$$h(88,2) = \left(88 + 2 * \left(1 + (88 \ mod \ 10\ )\right)\right) mod\ 11 = 7$$

$$h(59,0) = \left(59 + 0 * \left(1 + (59 \ mod \ 10\ )\right)\right) mod\ 11 = 4$$

$$h(59,1) = \left(59 + 1 * \left(1 + (59 \ mod \ 10\ )\right)\right) mod\ 11 = 3$$

$$h(59,2) = \left(59 + 2 * \left(1 + (59 \ mod \ 10\ )\right)\right) mod\ 11 = 2$$

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)

Rta (c)

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

Rta (C)

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.**~~