NOMBRE: ALVARO BLANCO
LEGAJO: 10622
TRABAJO PRACTICO Hash Table

Ejercicio 1:

Creamos un array con 9 posiciones y cada posicion conteniendo una linkedlist

Insertamos la key 5 con funcion de hash: h(k) = k mod 9
index = 5 mod 9 = 5

Insertamos la key 28 con funcion de hash: h(k) = k mod 9
index = 28 mod 9 = 1

Insertamos la key 19 con funcion de hash: h(k) = k mod 9
index = 19 mod 9 = 1 (colision se linkea con 28)

Insertamos la key 15 con funcion de hash: h(k) = k mod 9
index = 15 mod 9 = 6

Insertamos la key 20 con funcion de hash: h(k) = k mod 9
index = 20 mod 9 = 2

Insertamos la key 33 con funcion de hash: h(k) = k mod 9
index = 33 mod 9 = 6 (colision se linkea con 15)

Insertamos la key 12 con funcion de hash: h(k) = k mod 9
index = 12 mod 9 = 3

Insertamos la key 17 con funcion de hash: h(k) = k mod 9
index = 17 mod 9 = 8

Insertamos la key 10 con funcion de hash: h(k) = k mod 9
index = 19 mod 9 = 1 (colision se linkea con 19)

0: None
1: 28 -> 19 -> 10
2: 20
3: 12
4: None
5: 5
6: 15 -> 33
7: None
8: 17

Ejercicio 2

dictionary.py

```
def hash_function(key, m):
    return key % m


def insert(D, key, value):
    index = hash_function(key, len(D))
    if D[index] is None:
        D[index] = [(key, value)]
    else:
        for i, (k, v) in enumerate(D[index]):
            if k == key:
                D[index][i] = (key, value)
                break
            else:
                D[index].append((key, value))
    return D


def search(D, key):
    m = len(D)
    index = hash_function(key, m)
    for k, v in D[index]:
        if k == key:
            return v
    return None


def delete(D, key):
    m = len(D)
    index = hash_function(key, m)
    for i, (k, v) in enumerate(D[index]):
        if k == key:
            del D[index][i]
            return D
    return D


def traverseHashTable(D):
    size = len(D)
    for i in range(size):
        print('Index: ', i)
        if D[i] != None:
            for key, value in D[i]:
                print('Key: ', key, 'Value: ', value)
        else:
            print(None)
```

Ejercicio 3

```
import math
keys = [61,62,63,64,65]

A = (math.sqrt(5) - 1)/2

def hash(k,A):
 m=1000
 val = k * A % 1

 h = math.floor(m*val)

 return h

def printIndex():
 for k in keys:
   print(hash(k,A))
```

keys = [700, 318, 936, 554, 172]

Ejercicio 4

```
def isPermutation(s, p):
    char_count = {}
    for c in s:
        if c in char_count:
            char_count[c] += 1
        else:
            char_count[c] = 1
    for c in p:
        if c not in char_count or char_count[c] == 0:
            return False
        char_count[c] -= 1

    return True
```

Complejidad: O(N+M) con N = len(s) y M = len(p)

Ejercicio 5

```
def checkUnique(L):
    elements = []
    for value in L:
        if value in elements:
            return False
        elements.append(value)
    return True
```

Complejidad: O(n)

Ejercicio 6

```
def hash_codigo_postal(codigo_postal):
 hash_value = 0
 for c in codigo_postal:
   if c.isdigit():
     hash_value += ord(c) - ord('0')
   else:
     hash_value += ord(c) - ord('A') + 10
 return hash_value
```

Ejercicio 7

```
def compressStr(s):
 compress = ''
 cont = 0

 for i in range(len(s)-1):
   if s[i] == s[i+1]:
     if i == 0 :
       cont += 2
     else:
       cont += 1
   else:
     compress += s[i] + str(cont)
     cont = 1
 compress += s[-1] + str(cont)
 if len(compress)>len(s):
   return s
 else:
   return compress

Complejidad: O(n)
```

Ejercicio 8

```
def kmp_search(text, pattern):
    n = len(text)
    m = len(pattern)
    f = [0] * m  # Inicializamos la tabla de fallos

    # Construimos la tabla de fallos
    j = 0
    for i in range(1, m):
        while j > 0 and pattern[i] != pattern[j]:
            j = f[j - 1]
        if pattern[i] == pattern[j]:
            j += 1
        f[i] = j

    # Realizamos la búsqueda del patrón en el texto
    i = 0
    j = 0
    while i < n and j < m:
        if text[i] == pattern[j]:
            i += 1
            j += 1
        elif j > 0:
            j = f[j - 1]
```

```
        else:
            i += 1
    if j == m:
        return i - j  # Devolvemos el índice de la primera ocurrencia
    else:
        return -1  # No se encontró ninguna ocurrencia del patrón


Complejidad O(K+L)
```

Ejercicio 9

```
def isSubSet(S,T):
 if len(T)>len(S):
   return False
 else:
   dict = {}
   for t in T:
     dict[t] = t

   for s in S:
     if s not in dict:
       return False
   return True


Complejidad: O(n+m)
```

Ejercicio 10

```
def mostrarTablas(llaves,m):
 print('Linear Probing')
 print(hash_linear_probing(llaves,m))
 print('Quadratic Probing')
 print(hash_quadratic_probing(llaves,m))
 print('Double hashing Probing')
 print(hash_double_hashing(llaves,m))



def hash_linear_probing(llaves, m):
 tabla = [None] * m
 for llave in llaves:
   i = llave % m
   while tabla[i] is not None:
     i = (i + 1) % m
   tabla[i] = llave
 return tabla


def hash_double_hashing(llaves, m):
 tabla = [None] * m
 for llave in llaves:
   i = llave % m
   j = 1 + (llave % (m - 1))
   while tabla[i] is not None:
     j += 1
     i = (i + j) % m
```

```
    tabla[i] = llave
  return tabla


def hash_quadratic_probing(llaves, m):
  tabla = [None] * m
  c1 = 1
  c2 = 3
  for llave in llaves:
    i = llave % m
    j = 0
    while tabla[i] is not None:
      j += 1
      i = (llave + c1*j + c2*j**2) % m
    tabla[i] = llave
  return tabla
```

```
Linear Probing
[22, 88, None, None, 4, 15, 28, 17, 59, 31, 10]
Quadratic Probing
[22, None, 88, 17, 4, None, 28, 59, 15, 31, 10]
Double hashing Probing
[22, None, 88, 17, 4, 59, 28, None, 15, 31, 10]
```

Ejercicio 12

A y B le faltan elementos. D presenta encadenamiento. Respuesta: C

Ejercicio 13

Opcion correcta: C.

Aplicando la funcion a hash a todos los valores que seran insertados, sabemos que el 42 y el 52 dan el mismo indice de tabla: 2 y que 23 y 33 dan ambas indice: 3.
Por lo que analizando la tabla resultado:

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

Con la primer lista (A) quedaria en la posicion 3 el 52, por lo que se descarta
Con la segunda lista (B) se descarta porque quedan intercambiados los ultimos 2 elementos
La opcion C es la opcion correcta, ya que su salida es igual a la tabla dada
La opcion D se descarta ya que el 33 se ingresa antes que el 23 y ocupa su posicion en el indice 3

LINK: https://replit.com/@AlvaroBlanco/hashtable