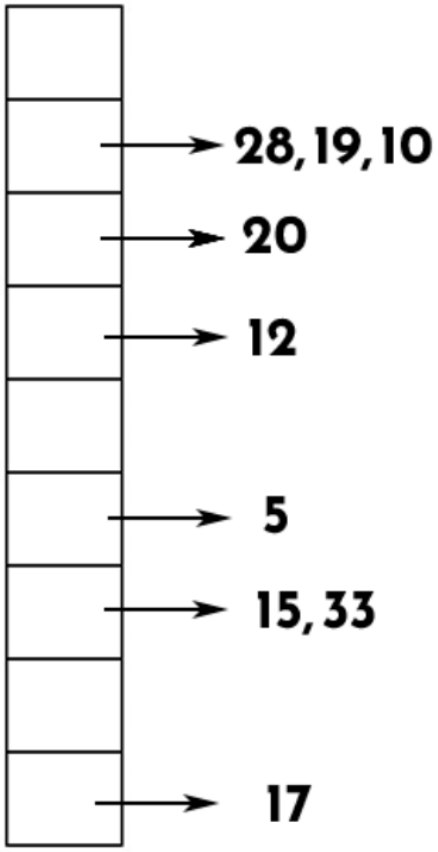
Alumno: Manuel Matías QUESADA RICCIERI

PARTE 1

Ejercicio 1

Inserción
5, 28, 19, 15, 20, 33, 12, 17, 10



Hash H(k) = k % 9

H(5) = 5
H(28) = 1
H(19) = 1
H(15) = 6
H(20) = 2
H(33) = 6
H(12) = 3
H(17) = 8
H(10) = 1

Ejercicio 2

insert(D,key, value)

```
def insert(D, key, value):
    newNode = (key, value)
    newKey = hash(key, len(D))

    if D[newKey] == None:
        newList = linkedlist.LinkedList()
        linkedlist.addList(newList, newNode)
        D[newKey] = newList
    else:
        linkedlist.addList(D[newKey], newNode)
    return D
```

search(D,key)

```python
def searchinList(D, key, newKey):
    currentNode = D[newKey].head
    while currentNode != None:
        if currentNode.value[0] == key:
            return currentNode.value[1]
        currentNode = currentNode.nextNode
    return None

def search(D, key):
    newKey = hash(key, len(D))

    if D[newKey] == None:
        return None
    else:
        return searchinList(D, key, newKey)
    return None
```

delete(D,key)

```python
def delete(D,key):
    newKey = hash(key, len(D))

    if D[newKey] == None:
        return D

    if D[newKey].head.value[0] == key:
        D[newKey].head = D[newKey].head.nextNode
        if D[newKey].head == None:
            D[newKey] = None
        return D

    currentNode = D[newKey].head
    while currentNode.nextNode != None:
        if currentNode.nextNode.value[0] == key:
            currentNode.nextNode = currentNode.nextNode.nextNode
            return D
        currentNode = currentNode.nextNode
    return D
```

PARTE 2

Ejercicio 3

h(k) = m * (((k * sqrt(5) -1) / 2) % 1)
h(61) = 700
h(62) = 318
h(63) = 936
h(64) = 554
h(65) = 172

Ejercicio 4

```python
def isPermutation(S, P):
    if len(S) != len(P):
        return False

    stringDict = dictionary(ord("z") - ord("a"))

    for i in range(len(S)):
        keyS = ord(S[i]) - ord("a")
        insert(stringDict, keyS, S[i])

    for j in range(len(P)):
        keyD = ord(P[j]) - ord("a")
        delete(stringDict, keyD)

    for k in range(len(stringDict)):
        if stringDict[k] != None:
            return False
    return True
```

La complejidad temporal del algoritmo es O(n), siendo n la longitud de la cadena S, sin embargo, en el peor de los casos, cuando todas las letras de S son diferentes, la complejidad temporal es O(n^2) debido a las colisiones en el diccionario

Ejercicio 5

```
def Unique(L):

    listDict = dictionary(len(L))

    for i in range(len(L)):
        key = hash(L[i], len(L))
        if search(listDict, key) != None:
            return False
        else:
            insert(listDict, key, L[i])
    return True
```

La complejidad temporal de este algoritmo es de O(n) siendo n la longitud de L

Ejercicio 6

```
def hashPostal(codigopostal, mod):
    sumatoria = 0
    i = 1
    for c in codigopostal:
        if c.isalpha():
            sumatoria = sumatoria + ord(c) * i
            i += 1
        elif c.isdigit():
            sumatoria = sumatoria + int(c) * i
            i += 1

    return sumatoria % mod
```

Ejercicio 7

```
def Compile(P):
    compileDict = dictionary(len(P))
    key = 0

    insert(compileDict, key, P[0])
    for i in range(1, len(P)):
        if P[i] == P[i-1]:
            insert(compileDict, key, P[i])
        else:
            key += 1
            insert(compileDict, key, P[i])

    stringFinal = ""
    contador = key

    for j in range(contador):
        stringFinal = stringFinal + compileDict[j].head.value[1]
        stringFinal = stringFinal + str(linkedlist.lengthList(compileDict[j]))

    if len(stringFinal) >= len(P):
        return P
    return stringFinal
```

La complejidad temporal de este algoritmo depende de la longitud de la cadena de entrada P. entonces es O(n), siendo n la longitud de P

Ejercicio 8

```python
def subString(S, P):
    longS = len(S)
    longP = len(P)

    if longS < longP:
        return None

    hashP = sum(ord(P[i]) for i in range(longP))
    hashS = sum(ord(S[i]) for i in range(longP))

    if hashP == hashS and P == S[:longP]:
        return 0

    for i in range(longP, longS):
        hashS += ord(S[i]) - ord(S[i-longP])

        if hashP == hashS and P == S[i-longP+1:i+1]:
            return i-longP+1
    return None
```

El costo computacional de este algoritmo es O(n + m), donde n es la longitud de la cadena de texto y m es la longitud de la cadena de patrón. Esto se debe a que se realiza una operación de suma para calcular el valor hash de la cadena de texto y la cadena de patrón, y el bucle for se ejecuta n - m + 1 veces.

Ejercicio 9

```python
def isSubset(S, T):
    subSetDict = dictionary(len(T))

    for i in range(len(T)):
        insert(subSetDict, hash(T[i], len(T)), T[i])

    for j in range(len(S)):
        if search(subSetDict, S[j]) == None:
            return False

    return True
```

La complejidad temporal de este algoritmo es de O(longitud de S + longitud de T), ya que realiza una inserción para cada elemento de T y se realiza una búsqueda por cada elemento de S

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

22	88			4	15	28	17	59	31	10

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

22		88	17	4		28	59	15	31	10

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

22		59	17	4	15	28	88		31	10

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 tabla resultante será la (C), ya que en las primeras 2 no se encuentran todos los elementos que se tienen que insertar y en la (D) todavía no se realiza la exploración lineal.

Ejercicio 13

La opción correcta es la (C), ya que utilizando el método de exploración lineal podemos incidir en el resultado mostrado en el ejemplo. Para demostrarlo, enseñaré lo que ocurre en cada uno de las opciones:

(A)

		42	52	34	23	46	33		

(B)

		42	23	34	52	33	46		

(C)

		42	23	34	52	46	33		

(D)

		42	33	23	34	46	52		