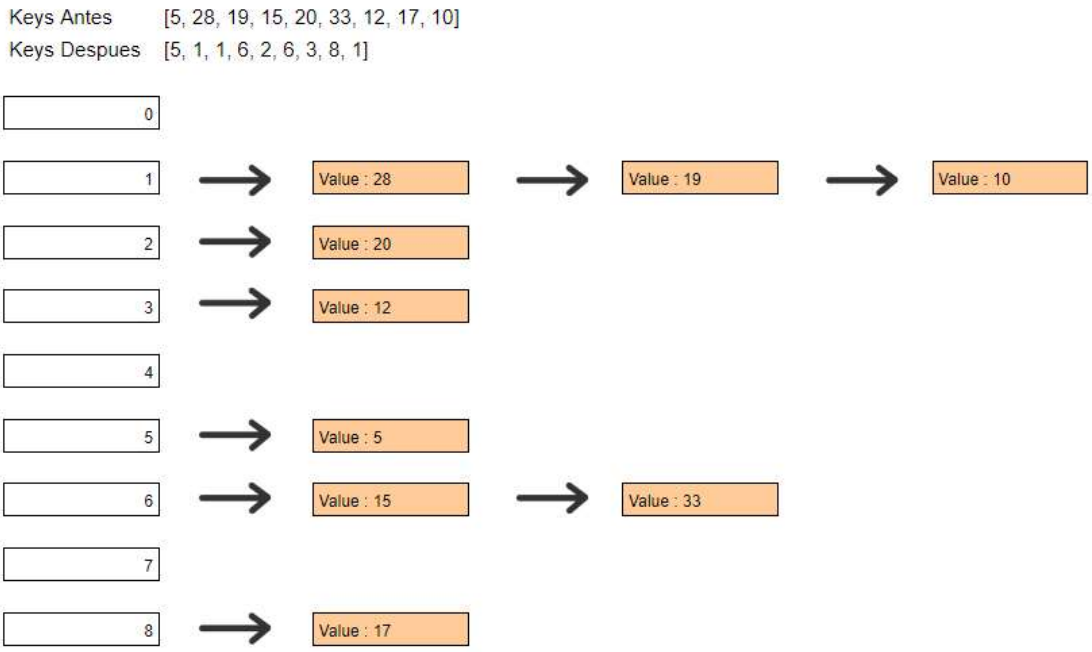
Giovanni Azurduy

1)



2)

```python
def crear_diccionario(size):
    dictionary = []
    i = 0
    while i < size:
        dictionary.append(None)
        i += 1
    return dictionary
```

```python
def insert(D,key,value):
    """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."""
    if len(D) < key:
        return "Failed"

    if D[key] == None:
        L = linkedlist.LinkedList()
        linkedlist.add(L,value)
        D[key] = L
    else:
        L = D[key]
        linkedlist.add(L,value)
    return D

def search(D,key):
    """Busca un key en el diccionario"""
    if len(D) < key:
        return None
    return D[key]

def delete(D,key):
    """Elimina un key en la posición determinada por la función de hash (1) del diccionario (dictionary)"""
    if len(D) < key:
        return None
    D[key] = None
    return D
```

3)

Keys	HashKey
61	700
62	318
63	936
64	554
65	172

4)

```
def search_element(D,element):
    key = hashing(len(D)-1,ord(element))
    linklist = search(D,key)
    if linklist == None:
        return
    index = linkedlist.search(linklist,element)
    result = linkedlist.access(linklist,index)
    return result

def ispermutation(string_s,string_p):
    largo_caracteres = 128
    list_abecedario = crear_diccionario(largo_caracteres)

    for s in string_s:
        key = hashing(largo_caracteres-1,ord(s))
        insert(list_abecedario,key,s)

    count = 0
    for p in string_p:
        element = search_element(list_abecedario,p)
        if element == p:
            count += 1

    return (count == len(string_s))
```

Peor Caso es (K^2) y caso Promedio (K)

5)

```
#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.
"""
#hash

def lista_unica(lista):
    dictionary = crear_diccionario(len(lista))
    for i in range(lista):
        index = search_element(dictionary,lista[i])
        if linkedlist.access(lista[i],index) == i:
            return False
        insert(dictionary,hashing(i),lista[i])

    return True
```

6)

le asignamos un valor a cada letra según el orden del abecedario: A-1,B-2,C-3,D-4,E-5 y sumamos todos los caracteres junto el numero de la altura , y por ultimo utilizamos el metodo de la division

7)

```python
def compresion_cadena(cadena):
    count = 1
    nueva_cadena = ""
    if len(cadena) == 1:
        return cadena
    for i in range(1,len(cadena)):
        if len(cadena)-1 == i:
            nueva_cadena += f"{cadena[i-1]}{count+1}"
        elif cadena[i-1] != cadena[i]:
            nueva_cadena += f"{cadena[i-1]}{count}"
            count = 0
        count += 1
    if len(nueva_cadena) < len(cadena):
        return nueva_cadena
    return cadena
```

8)

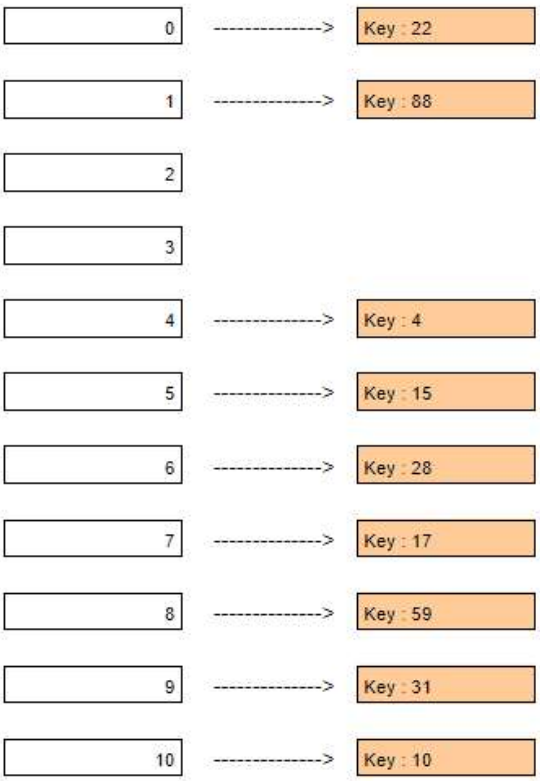
```python
def primera_ocurrencia(cadena_p , cadena_a):
    length_a = len(cadena_a)
    for i in range(len(cadena_p)):
        if cadena_p[i:i+length_a] == cadena_a:
            return i
```

9)

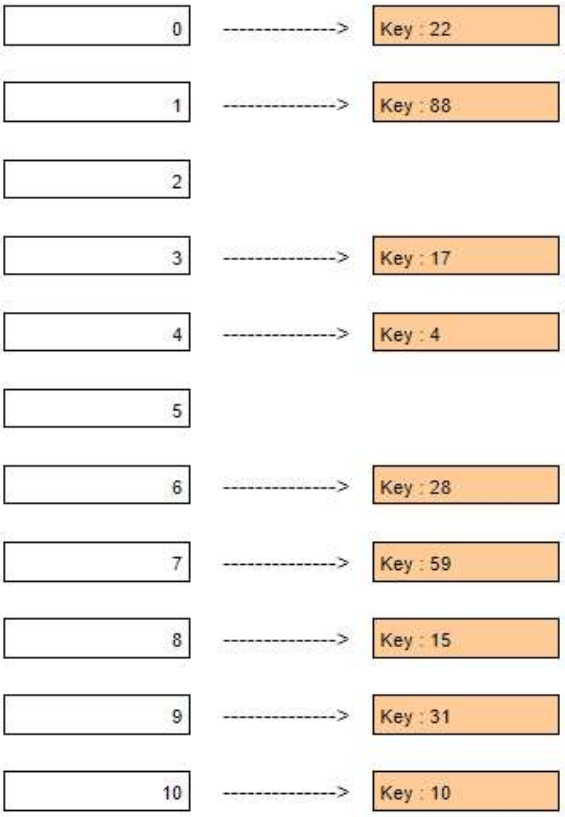
```python
def search_element_9(D,element):
    key = hashing(len(D)-1,element)
    linklist = search(D,key)
    if linklist == None:
        return
    index = linkedlist.search(linklist,element)
    result = linkedlist.access(linklist,index)
    return result

def subconjunto(S,T):
    largo_T = len(T)
    hashtable = crear_diccionario(largo_T)

    for t in range(largo_T):
        key = hashing(largo_T-1,T[t])
        insert(hashtable,key,T[t])
    for s in range(len(S)):
        search = search_element_9(hashtable,S[s])
        if search != S[s]:
            return False

    return True
```

10)

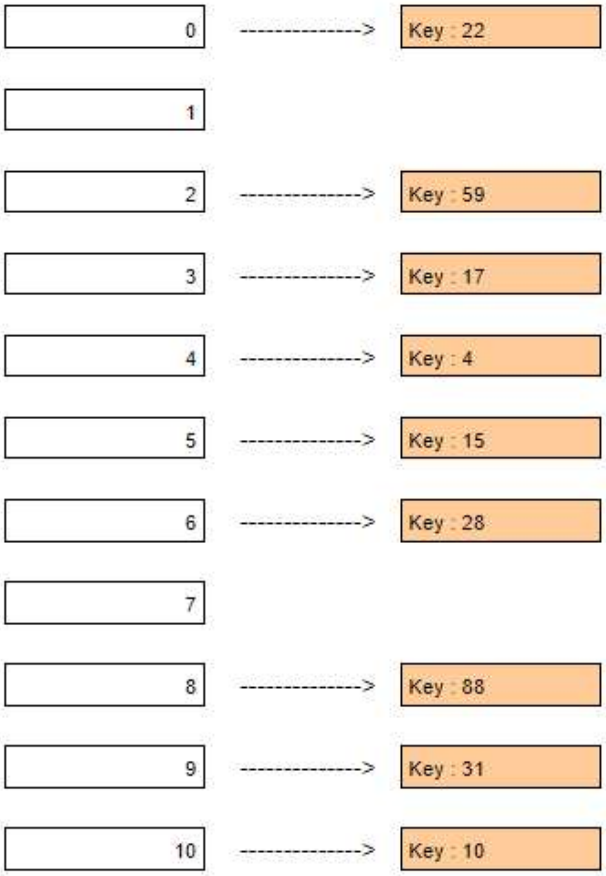
Linear Probing



Quadratic Probing



Double Hashing



12)
La tabla hash resultante del orden [12, 18, 13, 2, 3, 23, 5 y 15] es la c)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(C)

Porque

key 12 – slot 2
key 18 – slot 8
key 13 – slot 3
key 2 – slot 4 Porque ya esta ocupado el slot 2 y busca el próximo más cercano vacío
key 3 – slot 5 Porque ya esta ocupado el slot 3 y busca el próximo más cercano vacío
key 23 – slot 6 Porque ya esta ocupado el slot 3 y busca el próximo más cercano vacío
key 5 – slot 7 Porque ya esta ocupado el slot 5 y busca el próximo más cercano vacío
key 15 – slot 9 Porque ya esta ocupado el slot 5 y busca el próximo más cercano vacío
13)
El orden de llave que da el siguiente hashtable con direccionamiento abierto

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

es : (C) [46, 34, 42, 23, 52, 33]
Porque el 46 ocupa el slot 6 luego el 34 ocupa el slot 4 , el 42 ocupa el slot 2 , 23 ocupa el slot 3, 52 ocuparía el slot 2 pero como está ocupado busca uno vacio hacia abajo, ocupando el slot 5, 33 ocuparía el slot 3 pero como está ocupado busca el slot vacío más cercano siendo el 7