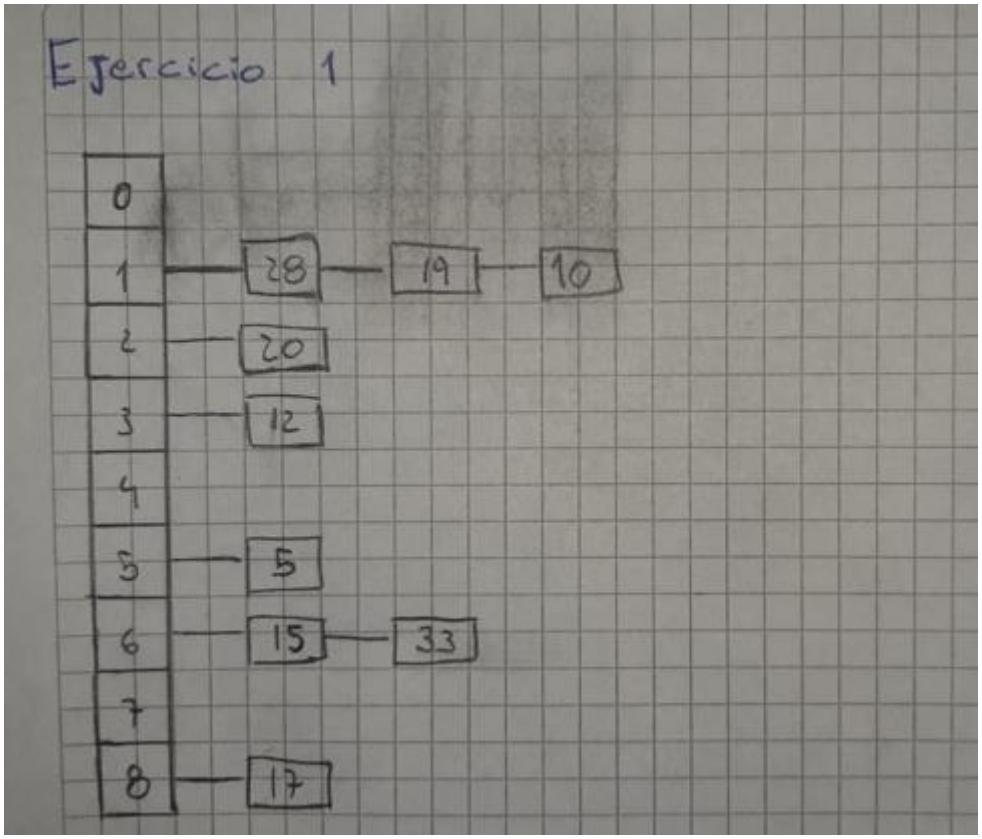
Trabajo Práctico N°4

Nombre: Victor Ramirez

Ejercicio 1



Ejercicio 2

```python
class DictionaryNode:
    def __init__(self, key= None, value = None):
        self.key = key
        self.value = value

class Dictionary:

    def __init__(self , hash_function = None , longitud = 10):
        self.D = [None] * longitud

        if hash_function != None:
            self.hash_function = hash_function
        else:
            self.hash_function = lambda x: x % longitud
```

```
16      def insert(self , D , key , value):
17
18          new_key = self.hash_function(key)
19
20          if D[new_key] == None:
21              new_node = DictionaryNode(key , value)
22              D[new_key] = []
23              D[new_key].append(new_node)
24          else:
25              new_node = DictionaryNode(key , value)
26              if search_list(value , D[new_key]) == False:
27                  D[new_key].append(new_node)
28          return D
29
30      def search(self , D , key):
31
32          new_key = self.hash_function(key)
33          if D[new_key] == None:
34              return None
35          else:
36              list = D[new_key]
37              pos = search_key(key , list)
38              if pos == None:
39                  return None
40              else:
41                  return list[pos].value
```

```
43      def delete(self , D , key):
44
45          new_key = self.hash_function(key)
46
47          if D[new_key] == None:
48              return D
49          else:
50              lista = D[new_key]
51              pos = search_key(key , lista)
52              if pos == None:
53                  return D
54              else:
55                  lista.pop(pos)
56                  return D
57
58
59
60  def search_list(element , lista):
61
62      for i in range(len(lista)):
63          if lista[i] != None:
64              if element == lista[i].value:
65                  return True
66      return False
```

```python
def search_key(key , lista):

    for i in range(len(lista)):
        if lista[i] != None:
            if key == lista[i].key:
                return i
    return None
```

Ejercicio 3

```python
def ejercicio3(lista):
    m = 1000
    A = (math.sqrt(5)-1)/2
    hash_function = lambda k : int(m*((k*A) % 1))
    dict = d.Dictionary(hash_function , m)
    for key in lista:
        dict.insert(dict.D , key , str(key))
    return dict.D

```

Ejercicio 4

```python
def is_permutation(element1 , element2):

    """
    La complejidad temporal de insert y search en un hash_table es O(1) , sin embargo
    la complejidad del algoritmo viene dada por la longitud de los elementos ya que
    se itera en un bucle .Por lo tanto la complejidad es O(n)
    """
    is_permutation = True
    if len(element1) != len(element2):
        return False

    m = 25  #Longitud del universo (los caracteres de a-z)
    hash_function = lambda x: ord(x) % m   #ord(x) es el codigo ASCII del caracter que ponga
    dict = d.Dictionary(hash_function , m)

    #Agregamos al diccionario todos los caracteres de el primer elemento
    for i in element1:
        dict.insert(dict.D , i , str(i))

    for j in element2:
        pos = dict.search(dict.D , j)
        if pos == None:
            is_permutation = False

    return is_permutation
```

Ejercicio 5

```python
def isSet(lista):
    """
    El costo de iterar la lista para agregar los elementos al diccionario es  O(n),
    las funciones insert,search,delete son O(1). Por lo tanto la complejidad del algoritmo
    viene dada por la longitud de la lista O(n).
    """
    m = len(lista)
    A = (math.sqrt(5)-1)/2
    hash_function = lambda k : int(m*((k*A) % 1))
    dict = d.Dictionary(hash_function , m)

    for key in lista:
        dict.insert(dict.D , key ,str(key))

    count = 0
    for key in lista:
        aux = dict.search(dict.D , key)
        if aux != None:
            dict.delete(dict.D , key)
            count += 1

    #Si despues de buscar los elementos, el contador es igual a la longitud de la lista
    #entonces significa que no hay elementos repetidos
    if count == m:
        return True
    else:
        return False
```

Ejercicio 6

```python
def codigo_postal(dict , codigo):
    #dict = d.Dictionary(hash_postal, 100003)
    print(f"El codigo postal se va a insertar en la posicion {hash_postal(codigo)}")
    dict.insert(dict.D ,codigo , str(codigo))
    return dict.D

def hash_postal(codigo):
    #La cantidad total de combinaciones del codigo postal que se puede
    #obtener con cddddccc son: 4569760000
    #Entonces vamos a tener que elegir un numero primo muy grande para que se distribuyan
    #correctamente las keys sin que se colisionen tanto

    m = 100003 #Numero primo grande
    num = int(codigo[1:5])
    char_ascii = ord(codigo[0])*10^4+ord(codigo[5])*10^3+ord(codigo[6])*10^2+ord(codigo[7])*10
    value = num + char_ascii
    hash_function = value % m
    return hash_function
```

Ejercicio 7

```python
def compress(element):
    #No supe como implementar diccionarios de manera optima en este ejercicio. La complejidad de
    #este algoritmo es O(n) porque itero una lista de tamaño n.

    lista = ""
    lista2 = list(element)
    cont = 0

    for i in range(len(element)):
        if i == 0:
            lista += element[i]
            cont += 1
            continue
        if element[i] == lista[-1]:
            cont += 1
        else:
            lista += str(cont) + element[i]
            cont = 1
    lista += str(cont)
    count = lista.count("1")
    if count == len(lista2):
        return element
    else:
        return lista
```

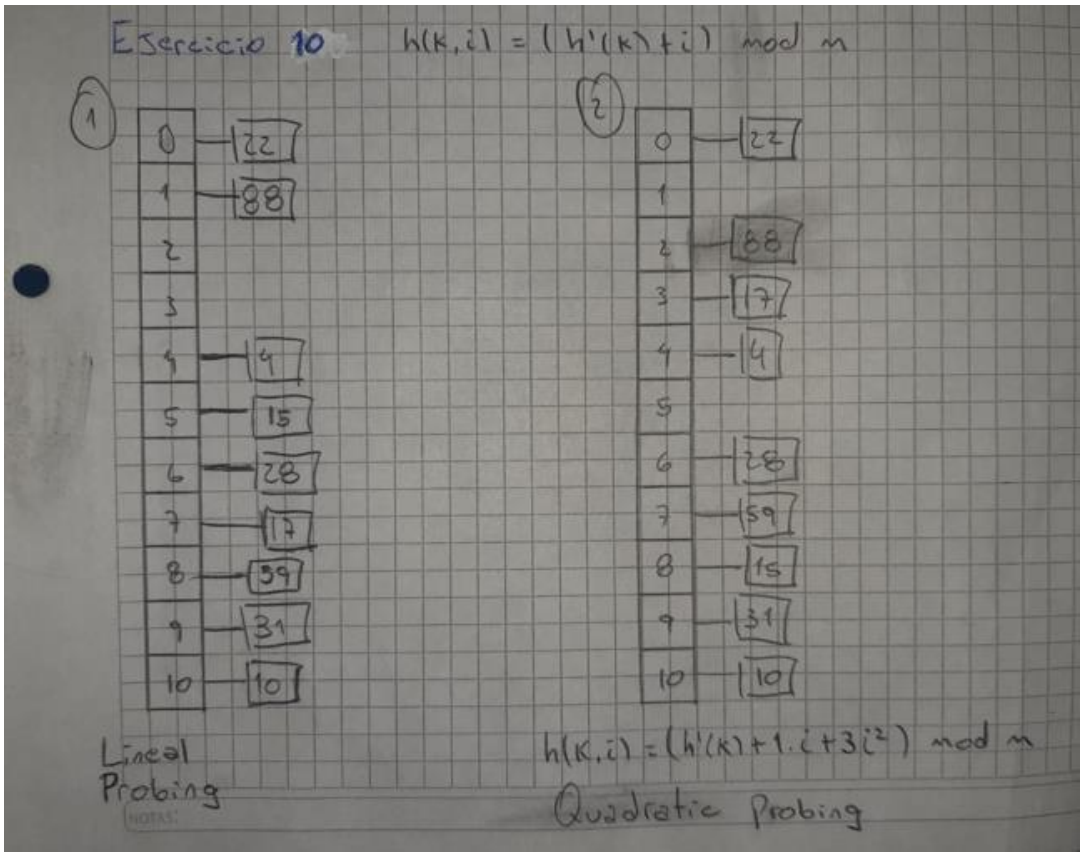
Ejercicio 8

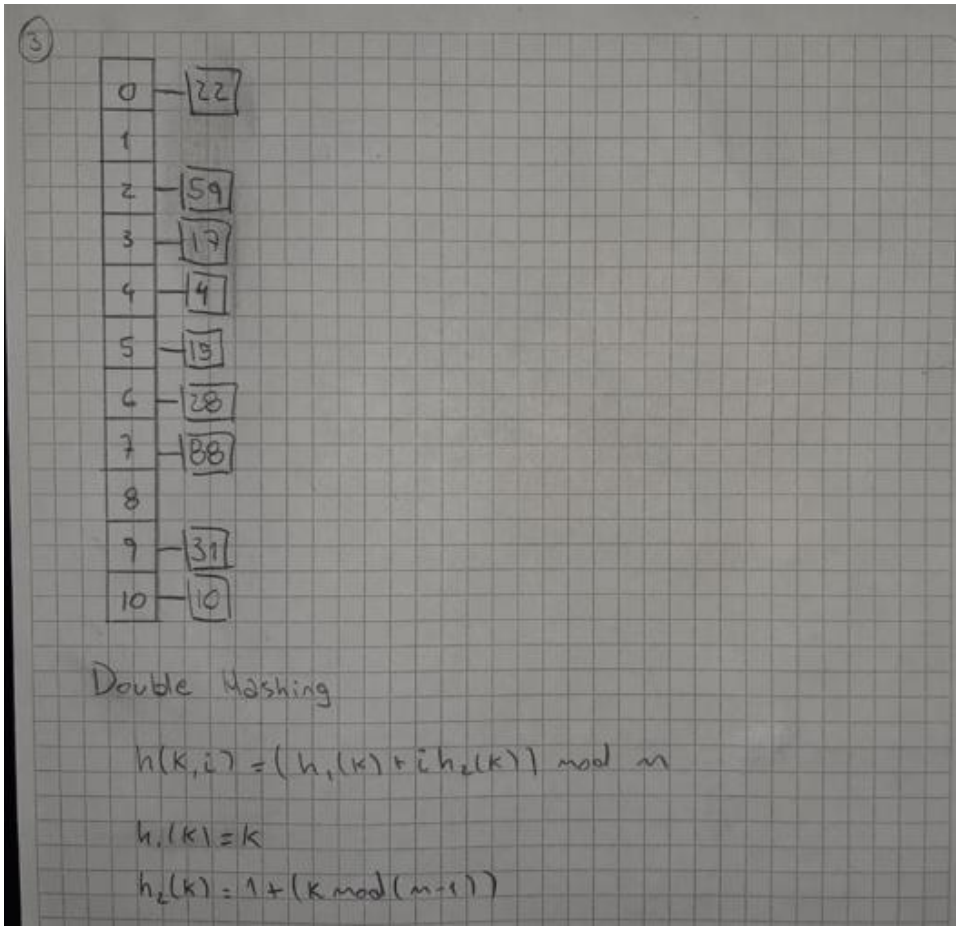
```
126  def inText(subtext , text):
127
128      aux = len(subtext)
129      dict = d.Dictionary(hash_text , 97)
130      for i in range(0, len(text)):
131          if i+3 <= len(text):
132              new_char = text[i:i+3]
133              dict.insert(dict.D , new_char , new_char)
134      found = dict.search(dict.D , subtext)
135      if found != None:
136          return True
137      else:
138          return False
139
140  def hash_text(text):
141
142      char_ascii = 0
143      for i in range(len(text)):
144          char_ascii += ord(text[i])*10^i
145
146      hash_function = char_ascii % 97
147      return hash_function
```

Ejercicio 9

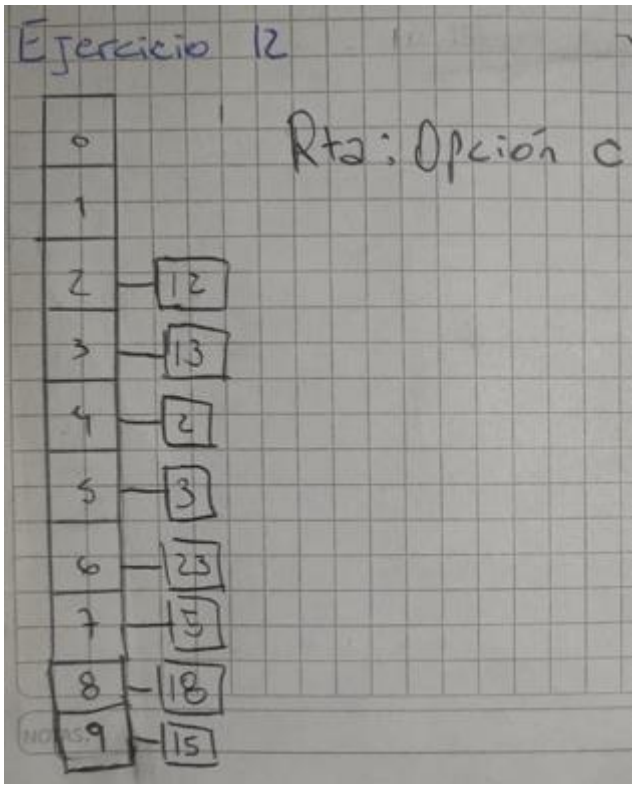
```
150  def isSubset(subconjunto , conjunto):
151      #La complejidad del algoritmo es O(n) y viene dada por la iteracion
152      #del conjunto al insertar, aunque el costo de insertar es O(1)
153
154      m = 67 #Numero primo
155      hash_function = lambda key : key % m
156      dict = d.Dictionary(hash_function , m)
157
158      #Agrego los elementos del conjunto a un diccionario
159      for key in conjunto:
160          dict.insert(dict.D , key , str(key))
161
162      #Itero el subconjunto y me fijo si sus elementos estan en el diccionario
163      #Si algun elemento no esta, entonces no es subconjunto
164      #Recordemos que por ser conjunto no hay elementos repetidos en el diccionario
165      for key in subconjunto:
166          found = dict.search(dict.D , key)
167          if found == None:
168              return False
169      return True
```

Ejercicio 10





Ejercicio 12



Ejercicio 13