# TP HASH

Nombre : Palau Enzo

Ejercicio 1:

[0] → []
[1] → [28,19,10]
[2] → [20]
[3] → [12]
[4] → []
[5] → [5]
[6] → [15,33]
[7] → []
[8] → [17]

Ejercicio 2:

insert:

```
10  def insert(D,key, value):
11      #inserta un elemento en la HashTable
12      if D==None or len(D)==0:
13          print("Please first create a HashTable with CreateHashTable(Dim) ")
14      else:
15          #busco la posc donde insertar
16          HashIndex=F_hash_Div(key,len(D))
17          Tupla=(key,value)
18          #inserto por encadenamineto
19          D[HashIndex].append(Tupla)
20      return D
```

Search:

```
22    def search(D,key):
23        #busca un key en un HashTable
24        HashIndex=F_hash_Div(key,len(D))
25        for each in D[HashIndex]:
26            if each[0] is key:
27                return each[1]
28        return None
```

Delete:

```
30    def delete(D,key):
31        if search(D,key) is None:
32            print("Not found")
33            return None
34        else:
35            HashIndex=F_hash_Div(key,len(D))
36            for i in range(0,len(D[HashIndex])):
37                if D[HashIndex][i][0] is key:
38                    D[HashIndex].pop(i)
39                    return D
```

Otros:

```
46    """Funciones Hash"""
47
48    def F_hash_Div(K,M): return K % M
49
50    """Extras"""
51
52    def printHashTable(D):
53        count=0
54        for each in D:
55            print("[",count,"]","--->",end="")
56            print(each)
57            print("----")
58            count+=1
59
```

```
2    def CreateHashTable(Dim):
3        Hash=[]
4        #crea un Hash de M posciones
5        for i in range (0,Dim):
6            L=[]
7            Hash.append(L)
8        return Hash
```

Ejercicio 3:

61 → 700

62 → 318

63 → 936

64 → 554

65 → 172

Ejercicio 4:

```
3   """Ejercicio 4"""
4
5   def IsPermutation(S1,S2):
6       #cuidado que mayusculas sera otra cadena distinta
7       #dado dos cadenas vemos si una es permutacion de la otra
8       if S1 == S2 or len(S1) != len(S2): return False #no pueden ser permutaciones
9       else:
10          D=CreateHashTable(len(S1))
11          for letter in S1:
12              Hindex=ord(letter)-ord("A")#Func hash uno a uno
13              insert(D,Hindex,letter)
14          printHashTable(D)
15          for letter in S2:
16              Hindex=ord(letter)-ord("A")#Func hash uno a uno
17              Cond=search(D,Hindex)
18              if Cond==None: return False
19      return True
```

La complejidad es O(N) ya que recorro las palabras

## Ejercicio 5:

```
22  """Ejercicio 5"""
23
24  def Isunique(List):
25      #verifica que los elementos de la lista son unicos
26      D=CreateHashTable(len(List))
27      for i in range(0,len(List)):
28          #busco la key y veo si su value esta y coincide
29          HashIndex=F_hash_Div(List[i],len(D))
30          Aux=search(D,HashIndex)
31          insert(D,HashIndex,List[i])
32          if Aux is not None and Aux==List[i]:
33              return False
34      return True
```

La complejidad es $O(N^2)$ ya que tengo un un bucle y un search, es posible hacer el ejercicio en N.

## Ejercicio 6:

```
"""Ejercicio 6"""

def C_PostalesHash(key,M):
    #una funcion de hash para los codigos cddddccc c=carc d=int
    return ((ord(key[0])*10**8+ord(key[1])*10**7+ord(key[2])*10**6+ord(key[3])*10**5+o
```

```
+ord(key[4])*10**4+ord(key[5])*10**3+ord(key[6])*10**2+ord(key[7])*10) % M)
```

## Ejercicio 7:

En este no hace falta usar un hash ya que la complejidad es de O(N)

```python
def CompressedString(string):
    #comprime cadenas con el numero de veces del caracter repetido
    #no es necesario un hash table
    cont=1
    Sresult=""
    #recorro y cuando el sig es otro reinicio el contador
    for i in range(0,len(string)):
        #final de cadena
        if i+1 is len(string):
            Sresult+=string[i]
            Sresult+=str(cont)
            return Sresult
        if string[i+1] is not string[i]:
            Sresult+=string[i]
            Sresult+=str(cont)
            cont=1
        else:
            cont+=1
```

## Ejercicio 8:

```python
def FirstOccurrence(S,P):
    #muestra el indice de la primera ocurrencia de p en s
    #hacer sin hash o poner un comentariop
    if S==P: return 0
    if len(S)<len(P): return False
    for i in range(0,len(S)):
        print(S[i:i+len(P)])
        if S[i:i+len(P)]==P:
            return True
    return False
```

La complejidad es O(n) recorro la lista con un for.

No encontré mejor solución usando un hashtable; pero algo que se podría hacer seria insertar las subcadenas de la long de p (con una funcion de hash que use los ascii y los pondere) y luego hacer un search.

# Ejercicio 9:

```
85  def subset(S,T):
86      #verifica si S es subconjunto de T
87      #insertar y luego en otro bucle poner search y no es o de n^2 porque no recorres toda la lista
88      D=CreateHashTable(len(T))
89      for each in T:
90          insert(D,each,each)
91      printHashTable(D)
92      for each in S:
93          if search(D,each) is None:
94              return False
95      return True
96
```

La complejidad seria de O(n) en el caso promedio.

# Ejercicio 10:

Keys :10; 22; 31; 4; 15; 28; 17; 88; 59

## Linear probing:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | 88 | | | 4 | 15 | 28 | 17 | 59 | 31 | 10 |

## Quadratic probing con c1 = 1 y c2 = 3

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | | 88 | 17 | 4 | | 28 | 59 | 15 | 31 | 10 |

Double hashing

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | | 59 | 17 | 4 | 15 | 28 | 88 | | 31 | 10 |

## Ejercicio 12:

La respuesta correcta es la c), te das cuanta cuando llegas a insertar la key=2

## Ejercicio 13:

La respuesta es la c) ya que quedaría como la tabla

Link de replit: https://replit.com/@EnzoPalau/HashTables