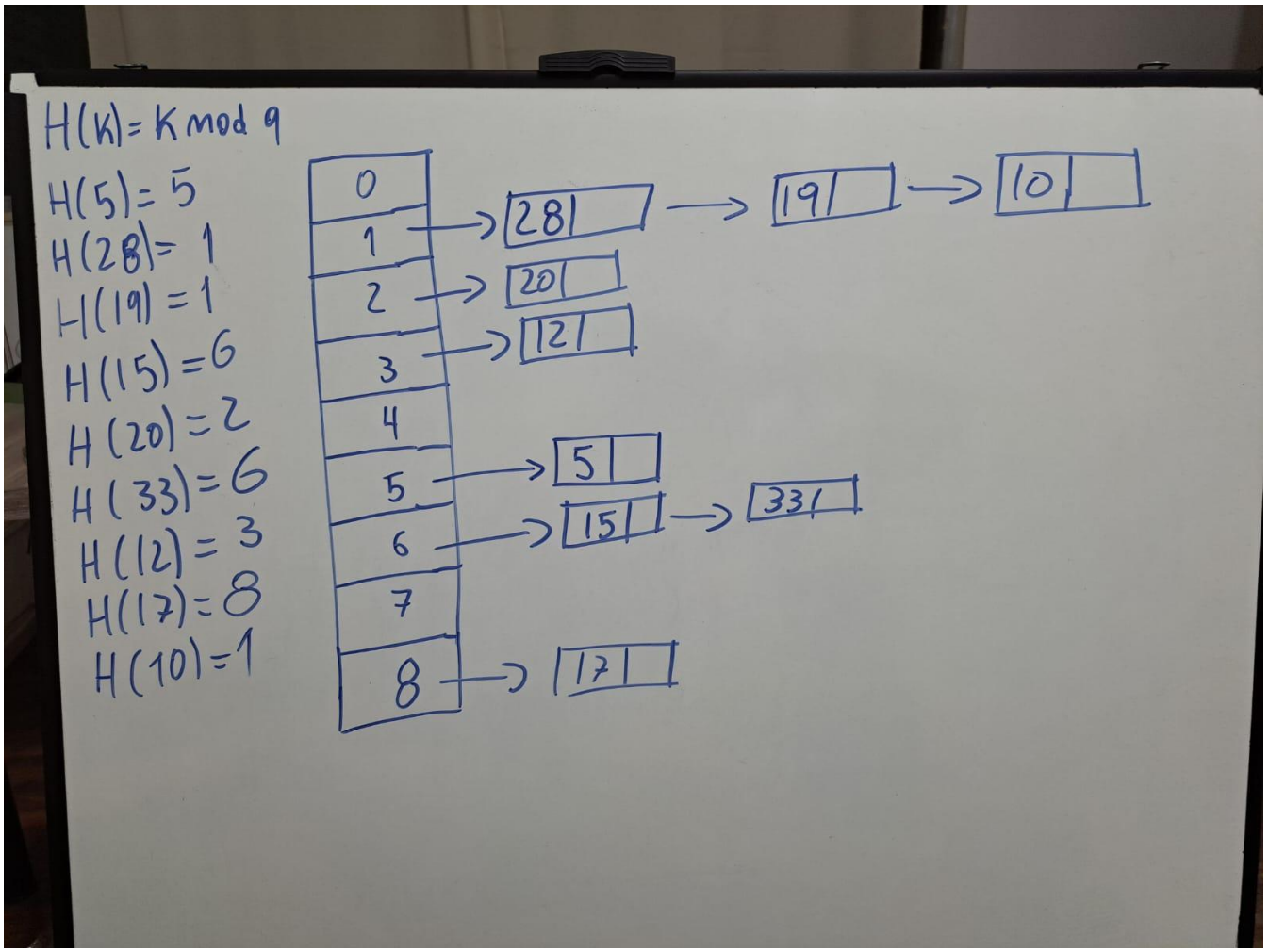
PARTE 1

Ejercicio 1

Ejemplificar que pasa cuando insertamos las llaves 5, 28, 19, 15, 20, 33, 12, 17, 10 en un **HashTable** con la colisión resulta por el método de chaining. Permita que la tabla tenga 9 slots y la función de hash:

$$H(k) = k \bmod 9 \qquad (1)$$



Ejercicio 2

A partir de una definición de diccionario como la siguiente:

```
dictionary = Array(m,0)
```

Crear un módulo de nombre **`dictionary.py`** que **implemente** las siguientes especificaciones de las operaciones elementales para el **TAD diccionario** .

Nota: puede **`dictionary`** puede ser redefinido para lidiar con las colisiones por encadenamiento

insert(D,key, value)

Descripción: 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.
Entrada: el diccionario sobre el cual se quiere realizar la inserción y el valor del key a insertar
Salida: Devuelve D

search(D,key)

Descripción: Busca un key en el diccionario
Entrada: El diccionario sobre el cual se quiere realizar la búsqueda (dictionary) y el valor del key a buscar.
Salida: Devuelve el value de la key. Devuelve **None** si el key no se encuentra.

delete(D,key)

Descripción: Elimina un key en la posición determinada por la función de hash (1) del diccionario (dictionary)
Poscondición: Se debe marcar como nulo el **key** a eliminar.
Entrada: El diccionario sobre el se quiere realizar la eliminación y el valor del key que se va a eliminar.
Salida: Devuelve **D**

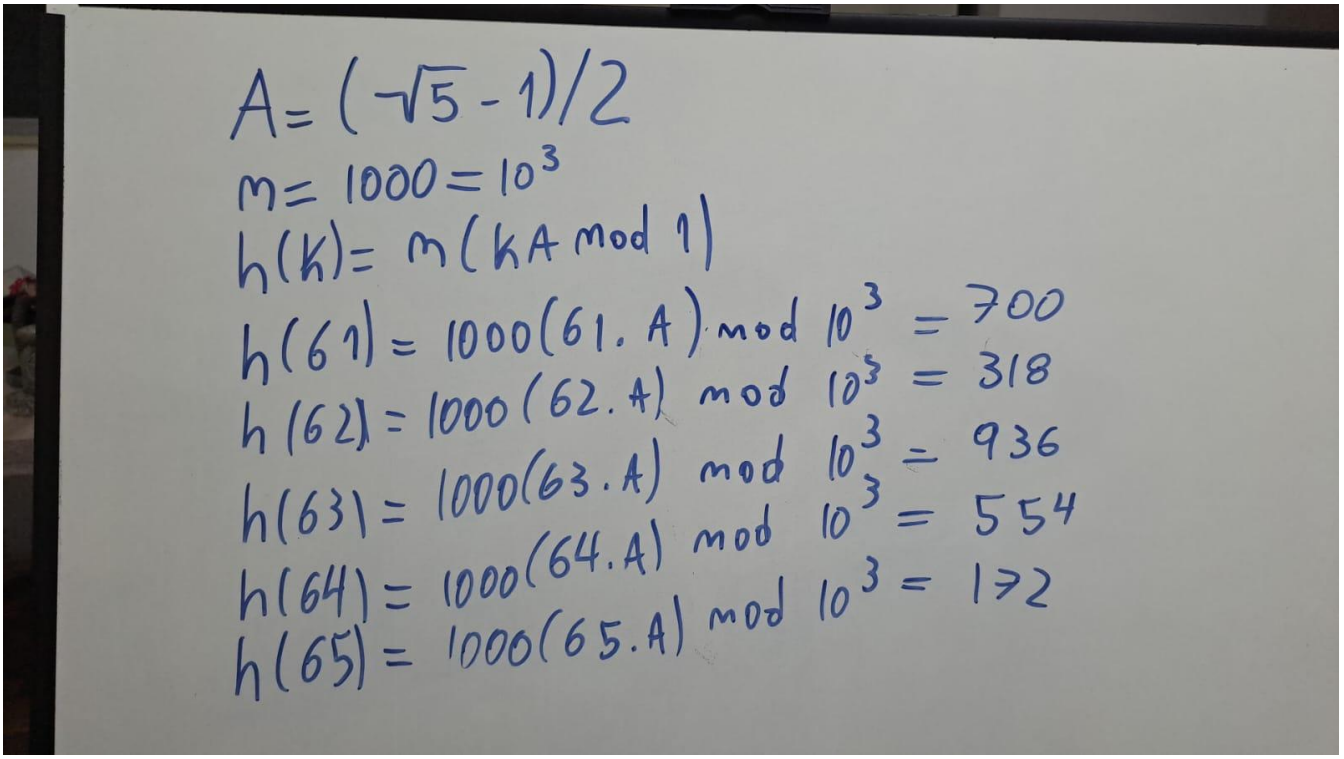
```python
import linkedlist
import array

class Dictionary:

    slots = None

    def __init__(self, length):
        self.slots = [None for i in range(length)]



class DictionaryItem:
    key = None
    value = None

    def __init__(self, key, value):
        self.key = key
        self.value = value


#Método división
def hashFunction(k, m):
    return k % m

```

```
26  def insert(D, key, value):
27      item = DictionaryItem(key, value)
28      slot = hashFunction(key, len(D.slots))
29      position = 0
30      if(D.slots[slot] == None):
31          D.slots[slot] = linkedlist.LinkedList()
32          linkedlist.add(D.slots[slot], item)
33      else:
34          currentNode = D.slots[slot].head
35          while currentNode != None:
36              if(currentNode.value.key < key):
37                  position += 1
38              elif(currentNode.value.key > key):
39                  linkedlist.insert(D.slots[slot], item, position)
40                  break
41              if(currentNode.nextNode == None):
42                  currentNode.nextNode = item
43                  break
44              currentNode = currentNode.nextNode
45      return D
46
47  def search(D, key):
48      slot = hashFunction(key, len(D.slots))
49      currentNode = D.slots[slot]
50      if (currentNode == None):
51          return None
52      else:
53          currentNode = currentNode.head
54      while currentNode != None:
55          if(currentNode.value.key == key):
56              return currentNode.value.value
57          currentNode = currentNode.nextNode
58      return
59
60  def delete(D, key):
61      slot = hashFunction(key, len(D.slots))
62      currentNode = D.slots[slot]
63      if (currentNode == None):
64          return None
65      else:
66          currentNode = currentNode.head
67      while currentNode != None:
68          if(currentNode.value.key == key):
69              linkedlist.delete(D.slots[slot], currentNode.value)
70          currentNode = currentNode.nextNode
71      return
```

PARTE 2

Ejercicio 3

Considerar una tabla hash de tamaño m = 1000 y una función de hash correspondiente al método de la multiplicación donde A = (sqrt(5)-1)/2). Calcular las ubicaciones para las claves 61,62,63,64 y 65.



Ejercicio 4

Implemente un algoritmo lo más eficiente posible que devuelva **True** o **False** a la siguiente proposición: dado dos strings $s_1...s_k$ y $p_1...p_k$, se quiere encontrar si los caracteres de $p_1...p_k$ corresponden a una permutación de $s_1...s_k$. Justificar el coste en tiempo de la solución propuesta.

Ejemplo 1:
Entrada: S = ‘hola’ , P = ‘ahlo’
Salida: True, ya que P es una permutación de S

Ejemplo 2:
Entrada: S = ‘hola’ , P = ‘ahdo’
Salida: Falso, ya que P tiene al carácter ‘d’ que no se encuentra en S por lo que no es una permutación de S

```
136  def isPermutation(s, p):
137      if(len(s) != len(p)):
138          return False
139
140      for i in range(0, len(s)):
141          if(s[i].upper() != p[len(p) - i - 1].upper()):
142              return False
143
144      return True
```

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.

Ejemplo 1:
Entrada: L = [1,5,12,1,2]
Salida: Falso, L no tiene todos sus elementos únicos, el 1 se repite en la 1ra y 4ta posición

```
87  def hasUniqueElements(L):
88      d = Dictionary(linkedlist.length(L))
89      currentNode = L.head
90      while currentNode != None:
91          if(search(d, currentNode.value) != None):
92              return False
93          insert(d, currentNode.value, currentNode.value)
94          currentNode = currentNode.nextNode
95      return True
```

Ejercicio 6

Los nuevos códigos postales argentinos tienen la forma cddddccc, donde c indica un carácter (A - Z) y d indica un dígito 0, . . . , 9. Por ejemplo, C1024CWN es el código postal que representa a la calle XXXX a la altura 1024 en la Ciudad de Mendoza. Encontrar e implementar una función de hash apropiada para los códigos postales argentinos.

```
97  def postalCodeHashFunction(char, k, m):
98      return ord(char * (k + 1)) % m
```

Ejercicio 7

Implemente un algoritmo para realizar la compresión básica de cadenas utilizando el recuento de caracteres repetidos. Por ejemplo, la cadena '*aabccccccaaa'* se convertiría en 'a2blc5a3'. Si la cadena "comprimida" no se vuelve más pequeña que la cadena original, su método debería devolver la cadena original. Puedes asumir que la cadena sólo tiene letras mayúsculas y minúsculas (a - z, A - Z). Justificar el coste en tiempo de la solución propuesta.

```python
def stringCompression(str):
    d = Dictionary(26)
    for i, char in enumerate(str):
        if(char != char.upper()):
            key = ord(char) - 71
        else:
            key = ord(char) - 65
        if(search(d, key) == None):
            insert(d, key, 1)
        else:
            cantidad = search(d, key) + 1
            delete(d, key)
            insert(d, key, cantidad)
    return compressedString(d, str)

def compressedString(D, str):
    string = ""
    for i in range(0, len(D.slots)):
        currentNode = D.slots[i]
        if currentNode != None:
            currentNode = currentNode.head
        while currentNode != None:
            key = currentNode.value.key
            if(key >= 26):
                key = key + 71
            else:
                key = key + 65
            string = string + chr(key) + repr(currentNode.value.value)
            currentNode = currentNode.nextNode
    if(len(string)/2 == len(str)):
        return str
    return string
```

Ejercicio 8

Se requiere encontrar la primera ocurrencia de un string $p_1...p_k$ en uno más largo $a_1...a_L$. Implementar esta estrategia de la forma más eficiente posible con un costo computacional menor a $O(K*L)$ (solución por fuerza bruta). Justificar el coste en tiempo de la solución propuesta.

Ejemplo 1:
Entrada: S = ‘abracadabra’ , P = ‘cada’
Salida: 4, índice de la primera ocurrencia de P dentro de S (abracadabra)

Ejercicio 9

Considerar los conjuntos de enteros $S = \{s1, \ldots, sn\}$ y $T = \{t1, \ldots, tm\}$. Implemente un algoritmo que utilice una tabla de hash para determinar si $S \subseteq T$ (S subconjunto de T). ¿Cuál es la complejidad temporal del caso promedio del algoritmo propuesto?

```
146  def isSubset(S, T):
147      if(linkedlist.length(S) > linkedlist.length(T)):
148          return False
149
150      ds = Dictionary(linkedlist.length(S))
151      dt = Dictionary(linkedlist.length(T))
152
153      currentNodeT = T.head
154      while currentNodeT != None:
155          insert(dt, currentNodeT.value, currentNodeT. value)
156          currentNodeT = currentNodeT.nextNode
157
158      currentNodeS = currentNodeS.head
159
160      while currentNodeS != None:
161          if(search(dt, currentNodeS.value) == None):
162              return False
163          currentNodeS = currentNodeS.nextNode
164
165      return True
```

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 $h2(k) = 1 + (k \bmod (m - 1))$
   [22| |59|17|4|15|28|88| |31|10]

Ejercicio 11 (opcional)

Implementar las operaciones de **`insert()`** y **`delete()`** dentro de una tabla hash vinculando todos los nodos libres en una lista. Se asume que un slot de la tabla puede almacenar un indicador (flag), un valor, junto a una o dos referencias (punteros). Todas las operaciones de

diccionario y manejo de la lista enlazada deben ejecutarse en O(1). La lista debe estar doblemente enlazada o con una simplemente enlazada alcanza?

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 sería la “C”, ya que en las demás tablas no se encuentran todos los elementos que se querían insertar y, además, se llenaron los slots donde estaban vacíos antes de llenar el mismo slot.

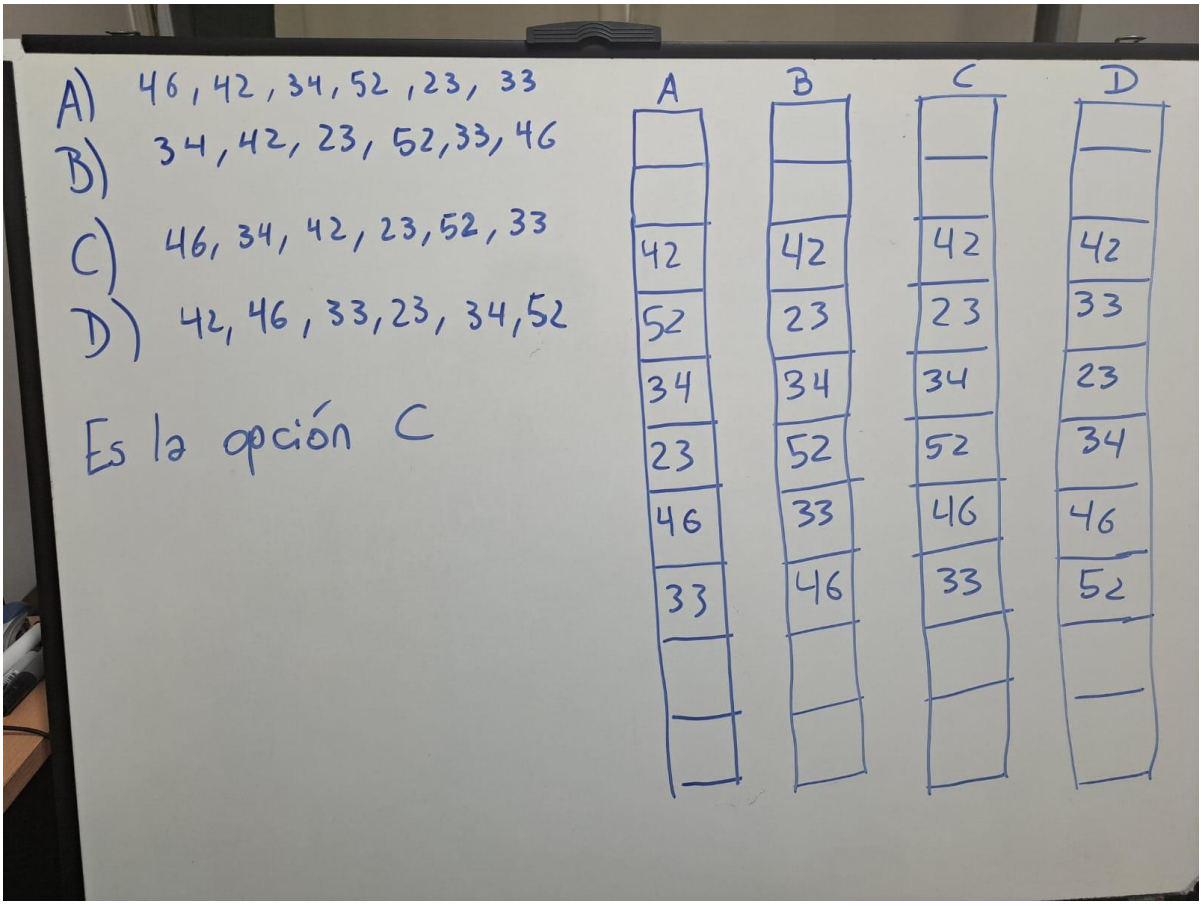
Ejercicio 13

Una tabla hash de longitud 10 utiliza direccionamiento abierto con función hash h(k)=k mod 10, y exploración lineal (linear probing). Después de insertar 6 valores en una tabla hash vacía, la tabla es como se muestra a continuación.

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

¿Cuál de las siguientes opciones da un posible orden en el que las llaves podrían haber sido insertadas en la tabla? Justifique
(A) 46, 42, 34, 52, 23, 33

(B) 34, 42, 23, 52, 33, 46
(C) 46, 34, 42, 23, 52, 33
(D) 42, 46, 33, 23, 34, 52



A tener en cuenta:

1. **Usen lápiz y papel primero**
2. ~~**No se puede utilizar otra Biblioteca mas allá de algo1.py y las bibliotecas desarrolladas durante Algoritmos y Estructuras de Datos I.**~~