Algoritmo y Estructura de Datos II

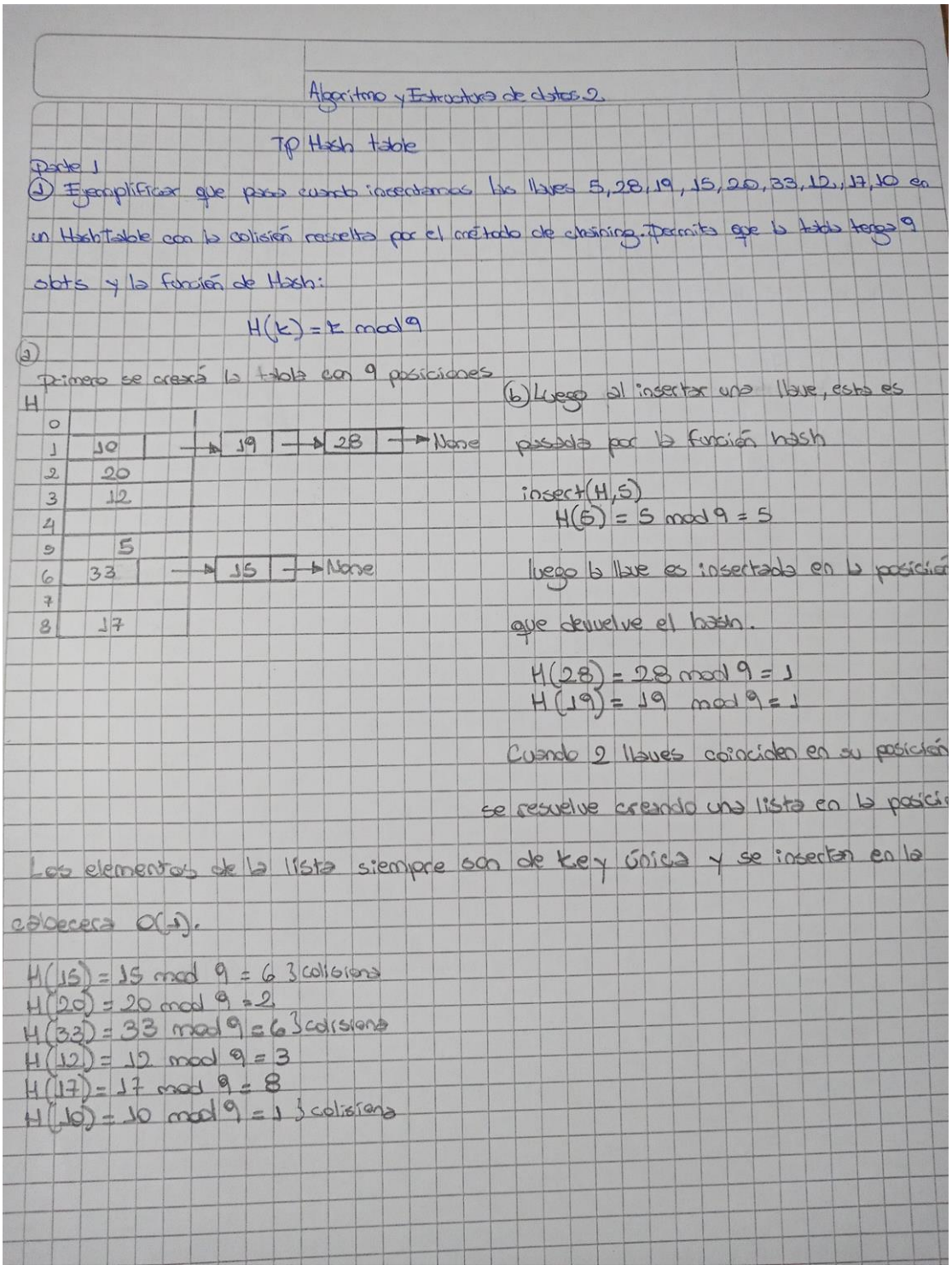
Alumno:Gonzalez Sanchez Gabriel
Legajo: 12007

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:

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



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**

```
#Defino el diccionario como un Array de nodos dictionary
def Dictionary(m):
    return Array(m,dictionary())

#Insert
#Inserta un value a través de una key en un diccionario D
#Devuelve el diccionario con la insercion
#Las colisiones se resuelven por encadenamiento
def insert(D,key,value):
    h = hash(key,len(D))
    if D[h]==None:
        L = dictionary()
        D[h] = L

    newNode = dictionaryNode()
    newNode.key = key
    newNode.value = value
    newNode.nextNode = D[h].head
    D[h].head = newNode
    return D

#Utilizo una llave y un modulo para aplicar metodo de la division
#Segun el tipo de dato(numerico,caracter o string) aplica diferentes calculos
def hash(key,module):
    if type(key)==int:
        return key % module
    elif type(key)==str:
        n = len(key)
        for i in range(0,n):
            if i==0:
                j = n-1
                newkey = ord(key[i])*(255**j)
            else:
                j -= 1
                newkey += ord(key[i])*(255**j)
        return newkey % module
```

```
#Search
#Busca la key en el diccionario D
#Devuelve el value asociado a la key o None si no lo encuentra
def search(D,key):
    h = hash(key,len(D))
    if D[h] == None:
        return None
    currentNode = D[h].head
    while currentNode!=None:
        if currentNode.key==key:
            return currentNode.value
        currentNode = currentNode.nextNode
    return None
```

```python
#Delete
#Elimina una Key en un diccionario D
#Devuelve el diccionario
def delete(D,key):
    h = hash(key,len(D))
    #Caso 1: Posicion vacia
    if D[h] == None:
        return D
    else:
        currentNode = D[h].head
        #Caso 2: El key esta en la cabeza de la lista
        #Si sa vacia una lista se busca que el slot de el diccionario quede vacio
        if currentNode.key == key and currentNode.nextNode == None:
            D[h] = None
        #Caso 3: Hay colisiones en la posicion
        else:
            currentNode=D[h].head
            #Caso elemento en el primer nodo
            if currentNode.key==key:

                D[h].head=currentNode.nextNode
            elif currentNode.nextNode!=None:
                flag = True
                while currentNode!=None and flag:
                    if currentNode.nextNode==None:
                        flag = False
                    elif currentNode.nextNode.key==key:
                        currentNode.nextNode=currentNode.nextNode.nextNode
                        flag = False
                    currentNode=currentNode.nextNode
        return D
```

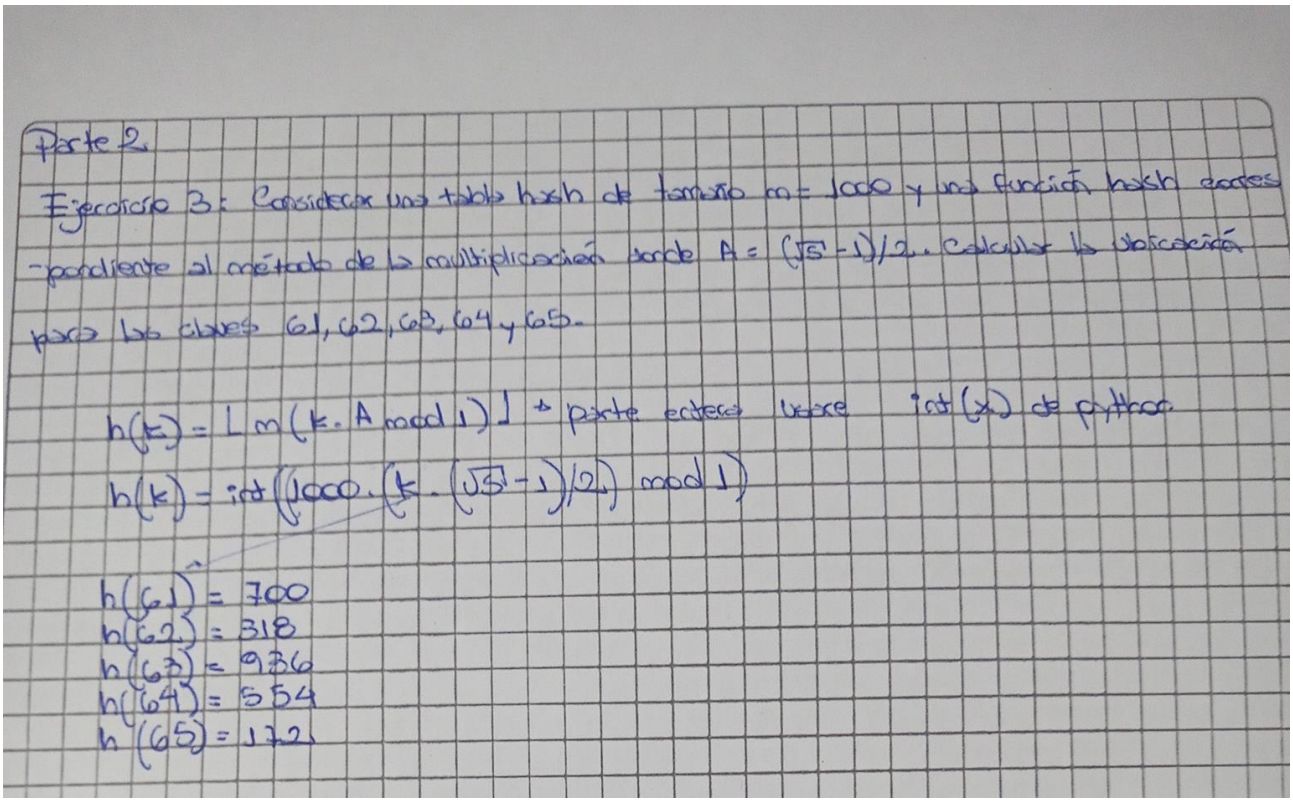
```
#Actualiza el value de un nodo en un dictionary
def update(D,key,value):
    m = len(D)
    currentNode = D[hash(key,m)]
    if currentNode==None:
        return None
    else:
        currentNode = currentNode.head
        while currentNode!=None:
            if currentNode.key == key:
                currentNode.value = value
                return value
            currentNode = currentNode.nextNode
        return None
```

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 3
def ex3():
    i = 61
    while i<=65:
        A = ((1000**(1/2))-1)/2
        print(int(6*((i*A)%1)))
        i+=1
```



Ejercicio 4

Implemente un algoritmo lo más eficiente posible que devuelva **True** o **False** a la siguiente proposición: dado dos strings s1...sk y p1...pk, se quiere encontrar si los caracteres de p1...pk corresponden a una permutación de s1...sk. Justificar el coste en tiempo de la solución propuesta.

```
#Ejercicio 4
#Determina si un string es permutacion de otro
#Su tiempo de ejecucion es O(m**2) siendo m la longitud de cadena
#Seria más optimo si delete retornara True o False en lugar de D
#De esta forma podriamos detener la ejecucion si no logra eleminar un caracter de
string2 en D
def isPermutation(string1,string2):
    n = len(string1)
    m = len(string2)

    #Check de ancho de palabra
    if m == n:
        D = Dictionary(m)
        #inserto string 1 en D
        for i in range(0,len(string1)):
            insert(D,string1[i],string1[i])

        #Elimino string 2 en D
        for i in range(0,len(string2)):
            delete(D,string2[i])

        flag = True
        for i in range(0,len(D)):
            if D[i]!=None:
                flag = False
        if flag:
            return flag
    return False
```

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

```
#Ejercicio 5
#Determina si una lista L posee elementos unicos
#Busca el elemento en D, si existe previamente devuelve False
def Unique(D,L):
    if L == None:
        return
    else:
        for i in range(0,len(L)):
            flag = search(D,L[i])
            if flag == None:
                insert(D,L[i],L[i])
            else:
                return False
        return True
```

Complejidad:

Las operaciones search y insert son O(1), la lista es recorrida una única vez por lo que su complejidad es O(n).

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.

```
#El formato de las direcciones es cddddccc
#Hay 26 letras desde la A-Z para c y 10 digitos desde 0-9 para d
#Genero una clave unica
def postalHash(k,m):
    cityreference = 0
    streetreference = ''
    count1 = 3
    for i in range(0,8):
        if (i<=0 or i>=5):
            print(ord(k[i])-ord('A'))
            print(10**count1)
            cityreference = cityreference + (ord(k[i])-ord('A'))*(100**count1)
            count1 -= 1
        else:
            streetreference += k[i]

    keystr = str(cityreference) + streetreference
    key = int(keystr)
    #Utilizo metodo de la multiplicaion con A = ((5**(1/2))-1)/2
    A = ((5**(1/2))-1)/2
    return int(m*((key*A)%1))
```

Ejercicio 7

Implemente un algoritmo para realizar la compresión básica de cadenas utilizando el recuento de caracteres repetidos. Por ejemplo, la cadena ‘*aabcccccaaa’* 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.

```
#Comprime una palabra
#Recibe un diccionario y una palabra
#Devuelve la palabra o la compresion segun cual sea más corta
#Coste O(n)
def compresion(D,word):
    n = len(word)
    newString = ''
    newStringlen = 0
    lastInsert = word[0]
    for i in range(0,n):
        exist = search(D,word[i])
        if exist == None:
            insert(D,word[i],1)
        else:
            update(D,word[i],exist+1)
        if lastInsert != word[i] or i == n-1:
            newString += lastInsert+str(search(D,lastInsert))
            newStringlen += 2
            delete(D,lastInsert)
            lastInsert = word[i]
    if n>=newStringlen:
        return newString
    else:
        return word
```

Complejidad:

El algoritmo recorre la cadena una única vez insertando los caracteres distintos la primera vez que los encuentra, si un carácter se repite entonces actualiza su valor. Todas las operaciones, la de inserción, búsqueda y actualización se realizan en tiempo O(1). Además, entre ese proceso de búsqueda, si el carácter que busco es distinto al insertado en la iteración anterior entonces elimina la inserción anterior del hash en un operación O(1) y agrega esos caracteres eliminados al nuevo String. Por ello la complejidad del algoritmo es O(n).

Ejercicio 8

Se requiere encontrar la primera ocurrencia de un string p1...pk en uno más largo a1...aL. 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.

```
#Determina si una cadena S se encuentra como subcadena dentro de P
#Devuelte True al encontrar la primera instancia de S en P
#Devuelve False si no existe en P
def isInside(D,S,P):
    m = len(S)
    n = len(P)
    if m>=n:
        for i in range(0,m-n+1):
            insert(D,S[i:i+n],i)
        exist = search(D,P)
        if exist!=None:
            return True
        else:
            return False
```

Complejidad:

La complejidad en la primera parte del algoritmo es de m-n debido a que inserta subcadenas de tamaño n de S en el hash a través de insert que es O(1). Luego la búsqueda de P en el hash es una operación O(1). Así que podríamos decir que el algoritmo posee una complejidad O(m).

Ejercicio 9

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

```
#Determina si un S de enteros en subconjuto de T
#Devuelve True o False segun sea
def subSet(D,S,T):
    m = len(S)
    n = len(T)
    if n>=m:
        for i in range(0,n):
            exist = search(D,T[i])
            if exist!=None:
                update(D,T[i],exist+1)
            else:
                insert(D,T[i],1)

        for i in range(0,m):
            exist = search(D,S[i])
            if exist!=None:
                if exist == 0:
                    return False
                else:
                    update(D,S[i],exist-1)
            else:
                return False
        return True
```

Complejidad:
La complejidad promedio de este algoritmo es de m+n debido a que inserta en el hash los elementos de T y luego busca en el hash los elementos de S.