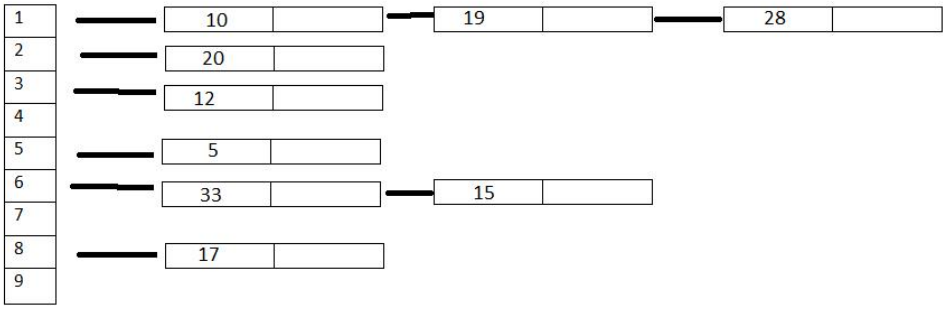
TP HASHTABLE

Tomás Rando - LCC - 14004

Ejercicio 1)



Ejercicio 2)

```
#Declara un diccionario (Array) de longitud m
def dictionary(m):
    dict = Array(m, linkedlist.LinkedList())
    return dict

#Hash por división k mod m
def hash(k, m):
    return (k % m)
```

Insert(D, key, value)

```
def insert(D, key, value):
    k = hash(key, len(D))
    element = (key, value)
    if D[k] == None:
        newList = linkedlist.LinkedList()
        linkedlist.add(newList, element)
        D[k] = newList
    else:
        linkedlist.add(D[k], element)
    return D
```

search(D, key)

```
def search(D, key):
    k = hash(key, len(D))
    if D[k] == None:
        return None

    currentNode = D[k].head
    while currentNode != None:
        if currentNode.value[0] == key:
            return currentNode.value[1]
        currentNode = currentNode.nextNode
    return None
```

delete(D, key)

```
def delete(D, key):
    k = hash(key, len(D))
    if D[k] == None:
        return D
    if D[k].head.value[0] == key:
        D[k].head = D[k].head.nextNode
        if D[k].head == None:
            D[k] = None
        return D

    currentNode = D[k].head
    while currentNode.nextNode != None:
        if currentNode.nextNode.value[0] == key:
            currentNode.nextNode = currentNode.nextNode.nextNode
            return D
        currentNode = currentNode.nextNode
    return D
```

Ejercicio 3)

Función hash = m * ((k * (math.sqrt(5) -1) / 2) % 1)

Las claves luego de utilizar el hash son:

hash(61) = 700
hash(62) = 318
hash(63) = 936
hash(64) = 554
hash(65) = 172

Ejercicio 4)

```
#Verifica si dos strings son permutaciones.
#Es O(n) siendo n la longitud de la string más larga o la longitud del alfabeto
#Esto es ya que se recorren las dos string y el diccionario cuyo tamaño es el del alfabeto
def permutation(S, P):
    if len(S) != len(P):
        return False

    D = dictionary(ord("z") - ord("a"))

    for i in range (0, len(S)):
        key = ord(S[i]) - ord("a")
        insert(D, key, S[i])
    for j in range (0, len(P)):
        key = ord(P[j]) - ord("a")
        delete(D, key)
    for i in range(0, len(D)):
        if D[i] != None:
            return False
    return True
```

Ejercicio 5)

```
#Verifica que todos los elementos de una lista sean unicos
#Es O(n) ya que lo máximo que se recorre es la longitud de la lista
#Además, el search y el insert son de O(1), por lo que no afectan
def unicos(L):
    D = dictionary(linkedlist.length(L))
    currentNode = L.head
    while currentNode != None:
        if search(D, currentNode.value) == None:
            insert(D, currentNode.value, currentNode.value)
        else:
            return False
        currentNode = currentNode.nextNode
    return True
```

Ejercicio 6)

```
def hashPostal(c, m):
    k = 1
    for i in range(1, 10):
        k = k * ord(c[i-1]) * i
    return k % m
```

Recorre todo el código postal y por cada elemento de la string se toma su código ASCII y se lo multiplica por la posición en la que se encuentra. Finalmente, se reduce al tamaño de la tabla mediante k mod m.

Ejercicio 7)
Usando hashtable:

```
#Comprime una cadena básica de cadenas contando los repetidos
#Es O(n) ya que se recorre la longitud de la string como peor caso
def compression(string):
    dict = dictionary(len(string))
    contador = 0
    for i in range(0, len(string)):
        if i == 0:
            insert(dict, contador, string[i])
        else:
            if string[i] == string[i-1]:
                insert(dict, contador, string[i])
            else:
                contador += 1
                insert(dict, contador, string[i])
    contadorFinal = contador
    contador = 0
    newString = ""
    for i in range(0, contadorFinal + 1):
        newString = newString + dict[i].head.value[1]
        newString = newString + str(linkedlist.length(dict[i]))
    if len(newString) >= len(string):
        return string
    else:
        return newString
```

Sin usar hashtable:

```
def compression2(string):
    newString = ""
    newString = newString + string[0]
    contador = 1
    for i in range(1, len(string)):
        if string[i] == string[i-1]:
            contador += 1
        else:
            newString = newString + str(contador)
            newString = newString + string[i]
            contador = 1
    newString = newString + str(contador)
    if len(newString) >= len(string):
        return string
    else:
        return newString
```

Ejercicio 8)

```
#Verifica que la string p este en la string a y devuelve el índice de la primera ocurrencia
#Es O(a - p + 1) o O(n) ya que se recorre la longitud de a - la de p + 1 como peor caso.
def insideString(p, a):
    length = len(p)
    lengthA = len(a)
    lengthDict = (lengthA - length) + 1
    dict = dictionary(lengthDict)
    for i in range(0, lengthDict):
        insert(dict, i, a[i: i + 4])
    for i in range(0, lengthDict):
        if dict[i].head.value[1] == p:
            return i
    return None
```

Ejercicio 9)

```
#Dados dos conjuntos de enteros verifica que S sea subconjunto de T
#Es O(n) ya que en el peor de los casos se recorre la lista más grande
def isSubset(S, T):
    D = dictionary(linkedlist.length(T))
    if T == None:
        return False
    if S == None:
        return True
    currentNode = T.head
    while currentNode != None:
        insert(D, currentNode.value, currentNode.value)
        currentNode = currentNode.nextNode

    currentNode = S.head
    while currentNode != None:
        if search(D, currentNode.value) == None:
            return False
        currentNode = currentNode.nextNode
    return True
```

Ejercicio 10)

1)

22
88
4
15
28
17
59
31
10

2)

22
88
17
4
28
59
15
31
10

3)

22
59
17
4
15
28
88
31
10

Ejercicio 12)

La correcta sería la tabla C. Las dos primeras se pueden descartar porque no tienen todos los elementos insertados, y, entre la D y la C podemos ver que es la C ya que utiliza la técnica pedida, es decir, calcula la posición mediante k % 10 y se fija si hay un elemento en ese lugar. Si existe, calcula una nueva posición mediante h'(x) = (h(x) + 1) % 10, hasta que encuentra una posición

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

(C)

Ejercicio 13)

La opción correcta es la C, ya que se utiliza la misma técnica que en el ejercicio anterior. Es decir, calcula la posición mediante k % 10 y se fija si hay un elemento en ese lugar. Si existe, calcula una nueva posición mediante h'(x) = (h(x) + 1) % 10, hasta que encuentra una posición. Si seguimos el patrón del C, podemos observar que queda como resultado la tabla brindada.