PRÁCTICA I

Mariajosé Chinchilla Morán

Dos sumas

Dado un arreglo de enteros y un entero *target*, devolver los índices de los números en el arreglo tales que su suma es igual a *target*.

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
            for i in range(0,len(nums)-1):
                list = nums[i+1:len(nums)]
                if target - nums[i] in list:
                    answer = [i,list.index(target - nums[i])+i+1]
                    break
            return answer
```

Análisis del algoritmo

Nota. Se tomará $n = len(nums)$. Los costos también están tomados en el peor caso.

Algorithm 1 Two Sum	*costo*	*veces*
class Solution:		
def twoSum(self, nums, target):		
for i in **range**(0, len(nums)-1):	c_1	$n-1$
list = nums[i+1:len(nums)]	c_2	$n-1$
if target - nums[i] in list:	n	$n-1$
answer = [i,list.index(target - nums[i])+i+1]	n	$n-1$
break		
return answer	c_5	1

TOTAL.

$$c_1(n-1) + c_2(n-1) + n(n-1) + n(n-1) + c_5 = (n-1)(c_1 + c_2 + 2n) + c_5$$
$$= 2n^2 + K_1 n + K_2 \in \mathcal{O}(n^2).$$

Primer caracter único en cadena

Dada una cadena, retornar el índice del primer caracter que no se repite. Si no existe, retornar -1.

```python
class Solution:
    def firstUniqChar(self,s):
        chars = set(s)
        copystring = s
        for char in chars:
            s = s.replace(char,'',1)
        else:
            for letter in copystring:
                if letter not in s:
                    return copystring.index(letter)
            for sym in s:
                if sym in copystring:
                    return -1
```

Análisis del algoritmo

Nota. Se tomará $n = |set(s)|$, es decir, el número de caracteres distintos en la cadena y $m = len(s)$.

Algorithm 2 Primer caracter único	*costo*	*veces*
class Solution:		
def firstUniqChar(self,s):		
chars = set(s)	c_1	1
copystring = s	c_2	1
for char **in** chars:	c_3	n
s = s.replace(char,”,1)	$m-n$	n
else:		
for letter **in** copystring:	c_4	m
if letter **not in** s:	$m-n$	m
return copystring.index(letter)	m	1
for sym **in** s:	c_5	$m-n$
if sym **in** copystring:	m	$m-n$
return -1	c_6	1

TOTAL

$$\begin{aligned} c_1 + c_2 + c_3 n + (m+n)^2 + c_4 m + m + (m+n)(c_5 + m) + c_6 \leq \\ c_1 + c_2 + m(c_3 + c_4 + 1) + 2m(3m + c_5) + c_6 = \\ K_1 + K_2 m + 6m^2 \in \mathcal{O}(m^2), \end{aligned}$$

donde la desigualdad se da porque $n \leq m$.

Búsqueda en matriz 2D

Escribir un algoritmo eficiente para buscar un valor en una matriz de enteros de mxn. Los valores en la matriz satisfacen

- Enteros en cada fila están ordenados de izquierda a derecha.
- El primer entero en cada fila es mayor al último de la fila anterior.

Retornar verdadero si el valor está en la matriz y falso de otro modo.

```python
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) ->
     bool:
        cols = len(matrix[0])
        rows = len(matrix)
        margins = []
        for i in range(rows):
            margins.append(matrix[i][cols-1])
        for num in margins:
            if target <= num:
                if target in matrix[margins.index(num)]:
                    return True
                break
        return False
```

Análisis del algoritmo

Nota. Se definen $n = cools, m = rows$ y $N = nxm$.

Algorithm 3 Búsqueda en matriz 2D	*costo*	*veces*
class Solution:		
def searchMatrix(self, matrix, target):		
cols = len(matrix[0])	c_1	1
rows = len(matrix)	c_2	1
margins = []	c_3	1
for i **in range(rows)**:	c_4	m
margins.append(matrix[i][cols-1])	c_5	m
for num **in** margins:	c_6	m
if target <= num:	c_7	m
if target **in** matrix[margins.index(num)]:	máx(m, n)	m
return True	c_8	1
break	c_9	1
return False	c_{10}	0

TOTAL.

$$c_1 + c_2 + c_3 + m(c_4 + c_5 + c_6 + c_7 + \text{máx}(m, n)) + c_8 + c_9 \in \begin{cases} K_1 + K_2 N = \mathcal{O}(N) & si \quad \text{máx}(n, m) = n \\ K_1 + K_2 m + K_3 m^2 = \mathcal{O}(m^2) & si \quad \text{máx}(n, m) = m. \end{cases}$$

Reacomodar matriz

Se es dada una matriz de mxm y dos enteros r y c que representan el número de filas y columnas de una nueva matriz deseada respectivamente. Reacomodar la matriz original en una de rxc. Si no es posible, retornar la matriz original.

```python
class Solution:
    def matrixReshape(self, mat: List[List[int]], r: int, c: int)
    -> List[List[int]]:
        cols = len(mat[0])
        rows = len(mat)
        output = []
        nums = []
        if r * c != cols * rows:
            return mat
        else:
            for i in range(rows):
                for j in range(cols):
                    nums.append(mat[i][j])
            for i in range(r):
                output.append(nums[i*c:(i+1)*c])
            return output
```

Análisis del algoritmo

Nota. Se definen $n = rows, m = cols$ y $N = nm$. En el caso de una entrada válida para reacomodo de matriz, $N = nm = rc$.

Algorithm 4 Reacomodar matriz	*costo*	*veces*
class Solution:		
def matrixReshape(self, mat, r, c):		
cols = len(mat[0])	c_1	1
rows = len(mat)	c_2	1
output = []	c_3	1
nums = []	c_4	1
if r * c != cols * rows:	c_5	1
return mat	c_6	1
else:		
for i **in** range(rows):	c_7	n
for j **in range(cols)**:	c_8	nm
nums.append(mat[i][j])	c_9	nm
for i **in range(r)**:	c_{10}	r
output.append(nums[i*c:(i+1)*c])	c_{11}	r
return output	c_{12}	1

TOTAL.

$$c_1 + c_2 + \cdots + c_6 + nc_7 + mn(c_8 + c_9) + r(c_{10} + c_{11}) + c_{12} \leq K_1 + K_2 N \in \mathcal{O}(N).$$

Enlace a Github.
https://github.com/MariajoseChinchilla/Practica1AA