Examen Final

Mariajosé Chinchilla Morán

Subiendo las escaleras

Estás subiendo una escalera. Se necesitan n pasos para llegar a la cima. Cada vez puedes subir 1 o 2 escalones. ¿De cuántas formas distintas puedes subir a la cima?

```python
class Solution:
    def climbStairs(self, n: int) -> int:
        a, b = 1, 1
        for i in range(n - 1):
            temp = a
            a = a + b
            b = temp

        return a
```

Análisis del algoritmo

Algorithm 1 Subir las escaleras

class Solution:		
def climbStairs(self, n: int) − > int:		
a, b = 1, 1	c	1
for i **in range(n - 1):**	c	n
temp = a	c	n
a = a + b	c	n
b = temp	c	n
return a		

Concluimos que el algoritmo es $\mathcal{O}(n)$, donde n la altura de la escalera.

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 2 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

Concluimos que el algoritmo es

$$\begin{cases} K_1 + K_2 N = \mathcal{O}(N) & si \quad \text{máx}(n, m) = n \\ K_1 + K_2 m + K_3 m^2 = \mathcal{O}(m^2) & si \quad \text{máx}(n, m) = m. \end{cases}$$

Número solitario

Dada una matriz no vacía de números enteros, cada elemento aparece dos veces excepto uno. Encuentra ese único.

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        numbers = set(nums)
        for i in numbers:
            nums.remove(i)
        for num in numbers:
            if num not in nums:
                return num
```

Análisis del algoritmo.

Algorithm 3 Número solitario

class Solution:		
def singleNumber(self, nums: List[int]) − > int:		
numbers = **set(nums)**	c	1
for i **in** numbers:		
nums.**remove(i)**	c	n
for num **in** numbers:		
if num **not in** nums:	c	n
return num		

Concluimos que el algoritmo es $\mathcal{O}(n)$, donde n es la longitud de la lista.

Enlace a Git

`https://github.com/MariajoseChinchilla/Examen-Final-AA`