

Programación en Python: Resolviendo Problemas en Línea

Aquí utilizamos un sistema de evaluación en línea para resolver problemas. Si tienes buen nivel de inglés, puedes usar Codeforces y LeetCode. En chino, puedes usar JiSuanKe y LiKou. Aquí usamos LeetCode. He resuelto 10 problemas. Además, para el último problema, utilicé varios métodos y optimicé la eficiencia del programa, pasando de superar el 10% de las entregas a superar el 99%.

The screenshot shows the Codeforces homepage with the following details:

- Header:** CODEFORCES, Sponsored by Telegram. User lzwjava is logged in.
- Navigation Bar:** HOME (highlighted), TOP, CONTESTS, GYM, PROBLEMSET, GROUPS, RATING, EDU, API, CALENDAR, HELP. A search bar is also present.
- Message Bar:** "Polygon and Codeforces will be possibly unavailable in the period between Mar. 16, 16:00 (UTC) and Mar. 17, 08:00 (UTC) because of maintenance."
- Contest Information:** "Codeforces Round #707 (Div.1, Div.2, based on Moscow Open Olympiad in Informatics, rated)" by ch_egor, 43 hours ago, translation, UK flag.
- Content Area:** Hello! Right now happens the first tour of the Open Olympiad in Informatics, and tomorrow will be the second one. This contest is prepared by Moscow Olympiad Scientific Committee that you may know by Moscow Team Olympiad, Moscow Olympiad for Young Students and Metropolises Olympiad (rounds 327, 342, 345, 376, 401, 433, 441, 466, 469, 507, 516, 541, 545, 567, 583, 594, 622, 626, 657, 680, 704). Open Olympiad consists of the most interesting and hard problems that are proposed by a wide community of authors, so we decided to conduct a Codeforces regular round based on it, which will happen on Saturday, March 13, 2021 at 17:05 UTC+8 and will be based on both days of the Olympiad. Each division will have 6 problems and 2 and a half hours to solve them. We kindly ask all the community members that are going to participate in the competition to show sportsmanship by not trying to cheat in any manner, in particular, by trying to figure out problem statements from the onsite participants. If you end up knowing some of the problems of Moscow Open Olympiad (by participating in it, from some of the onsite contestants or in any other way), please do not participate in the round. We also ask onsite contestants to not discuss problems in public. Failure to comply with any of the rules above may result in a disqualification.
- Footer:** Problems of this competition were prepared by Akulyat, KiKoS, wrg0ababd, Nebuchadnezzar, biexiong, alexX512 isaf27, ismagilov.code, DebNatkh. Siberian. NiceClock guided by cdkrot, vintage, Vlad Makeev, GlebsHP, Zlobober, meshanva, ch_eor.
- User Profile:** lzwjava (Rating: 1495, Contribution: 0). Includes links to Settings, Blog, Teams, Submissions, Favourites, Talks, Contests.
- Ranking:** Top rated users table.

Figure 1: cf

1480. Suma Acumulada de un Array Unidimensional

Dado un array `nums`, definimos la suma acumulada como un array `runningSum` donde `runningSum[i] = sum(nums[0]...nums[i])`. Es decir, cada elemento en `runningSum` es la suma de todos los elementos de `nums` desde el inicio hasta el índice `i`.

Ejemplo 1:

Entrada: `nums = [1,2,3,4]`

Salida: `[1,3,6,10]`

Explicación: La suma acumulada se obtiene de la siguiente manera: `[1, 1+2, 1+2+3, 1+2+3+4]`.

T1001 计算A+B (新手教程)
通过率: 55.0% 正确提交: 14493 总提交: 26372

T1002 输出马里奥
通过率: 28.8% 正确提交: 5633 总提交: 19569

T1003 输出字符菱形
通过率: 36.4% 正确提交: 5526 总提交: 15170

T1004 输出Hello, World!
通过率: 36.2% 正确提交: 8277 总提交: 22889

T1005 输出字符三角形
通过率: 52.0% 正确提交: 4768 总提交: 9163

T1006 对齐输出
通过率: 28.4% 正确提交: 3536 总提交: 12111

Figure 2: jsk

#	Title	Solution	Acceptance	Difficulty	Frequency
1757	Recyclable and Low Fat Products		96.1%	Easy	锁
1741	Find Total Time Spent by Each Employee		90.9%	Easy	锁
1693	Daily Leads and Partners		90.9%	Easy	锁
1683	Invalid Tweets		90.8%	Easy	锁
1119	Remove Vowels from a String		90.5%	Easy	锁
1350	Students With Invalid Departments		90.3%	Easy	锁
1378	Replace Employee ID With The Unique Identifier		90.1%	Easy	锁
1587	Bank Account Summary II		89.8%	Easy	锁
1571	Warehouse Manager		89.8%	Easy	锁
1303	Find the Team Size		89.6%	Easy	锁
1581	Customer Who Visited but Did Not Make Any Transactions		89.6%	Easy	锁

Figure 3: leetcode

Ejemplo 2:

Entrada: `nums = [1,1,1,1,1]`

Salida: `[1,2,3,4,5]`

Explicación: La suma acumulada se obtiene de la siguiente manera: `[1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1]`.

Ejemplo 3:

Entrada: `nums = [3,1,2,10,1]`

Salida: `[3,4,6,16,17]`

Restricciones:

- `1 <= nums.length <= 1000`
- `-10^6 <= nums[i] <= 10^6`

Solución

Para resolver este problema, podemos iterar a través del array `nums` y calcular la suma acumulada en cada paso. Aquí está el código en Python que implementa esta lógica:

```
def runningSum(nums):  
    for i in range(1, len(nums)):  
        nums[i] += nums[i-1]  
    return nums
```

Explicación:

1. Comenzamos iterando desde el segundo elemento del array (índice 1) hasta el final.
2. En cada paso, sumamos el valor del elemento anterior al elemento actual.
3. Finalmente, devolvemos el array modificado con las sumas acumuladas.

Complejidad:

- **Tiempo:** $O(n)$, donde n es la longitud del array `nums`. Solo necesitamos recorrer el array una vez.
- **Espacio:** $O(1)$, ya que estamos modificando el array original y no utilizamos espacio adicional.

Este enfoque es eficiente y fácil de entender, lo que lo convierte en una solución óptima para este problema.

Dado un arreglo `nums`. Definimos una suma acumulativa de un arreglo como

`runningSum[i] = suma(nums[0]...nums[i]).`

Devuelve la suma acumulativa de `nums`.

```
class Solution:

    def runningSum(self, nums: [int]) -> [int]:
        acumulado = []
        suma = 0
        for num in nums:
            suma += num
            acumulado.append(suma)

        return acumulado

print(Solution().runningSum([1,2,3,4]))
```

The screenshot shows a LeetCode problem page for the 'Running Sum of 1d Array' problem. The code has been submitted in Python3 and is displayed in the editor. The code is identical to the one shown above. The submission was successful, with a runtime of 80 ms and memory usage of 14.5 MB. The user can see their acceptance status, social sharing options, and a table of their recent submissions.

Time Submitted	Status	Runtime	Memory	Language
03/14/2021 00:47	Accepted	80 ms	14.5 MB	python3
03/14/2021 00:43	Runtime Error	N/A	N/A	python3

Figure 4: ac

La primera pregunta está aprobada.

1108. Desactivar una Dirección IP

Dada una dirección IP (Internet Protocol) válida (IPv4), devuelve una versión desactivada de esa dirección IP.

Una dirección IP desactivada reemplaza cada punto "." con "[.]".

Ejemplo 1:

Entrada: address = "1.1.1.1"

Salida: "1[.]1[.]1[.]1"

Ejemplo 2:

Entrada: address = "255.100.50.0"

Salida: "255[.]100[.]50[.]0"

Restricciones:

- La `address` dada es una dirección IPv4 válida.

Solución

Para resolver este problema, simplemente necesitamos reemplazar cada punto "." en la dirección IP con "[.]". Esto se puede hacer fácilmente utilizando la función `replace` en la mayoría de los lenguajes de programación.

Aquí está la solución en Python:

```
def defangIPAddr(address: str) -> str:  
    return address.replace('.','[.]')
```

Explicación:

- La función `defangIPAddr` toma una cadena `address` como entrada.
- Utiliza el método `replace` para reemplazar cada ocurrencia de "." con "[.]".
- Devuelve la cadena modificada.

Complejidad:

- **Complejidad de tiempo:** O(n), donde n es la longitud de la cadena address. Esto se debe a que el método replace recorre la cadena una vez.
- **Complejidad de espacio:** O(n), ya que se crea una nueva cadena con los reemplazos.

Este enfoque es simple y eficiente para el problema dado.

Dada una dirección IP (IPv4) válida address, devuelve una versión “defang” de esa dirección IP.

Una dirección IP defang reemplaza cada punto “.” con “[.]”.

```
class Solution:
    def defangIPaddr(self, address: str) -> str:
        return address.replace('.','[.]')
```

print(Solución().defangIPaddr('1.1.1.1'))

1431. Niños con el mayor número de caramelos

> Dado el arreglo `candies` y el entero `extraCandies`, donde `candies[i]` representa la cantidad de dulces que tiene el niño i.
>
> Para cada niño, verifica si hay una manera de distribuir `extraCandies` entre los niños de tal forma que cada niño tenga al menos tanto como el niño con más caramelos.

```python

```
class Solution:
 def kidsWithCandies(self, candies: [int], extraCandies: int) -> [bool]:
 max = 0
 for candy in candies:
 if candy > max:
 max = candy
 greatests = []
 for candy in candies:
 if candy + extraCandies >= max:
 greatests.append(True)
 else:
 greatests.append(False)
 return greatests
```

El código anterior define una clase `Solution` con un método `kidsWithCandies` que toma una lista de enteros `candies` y un entero `extraCandies`. El método determina si cada niño, después de recibir los `extraCandies`, tendrá la mayor cantidad de dulces entre todos los niños. Devuelve una lista de valores booleanos donde `True` indica que el niño tendrá la mayor cantidad de dulces y `False` indica lo contrario.

## **print(Solución().kidsWithCandies([2,3,5,1,3], 3))**

```
1672. La Riqueza del Cliente Más Rico
```

```
> Se te proporciona una cuadrícula de enteros `accounts` de tamaño `m x n`, donde `accounts[i][j]` repr
>
> La **riqueza** de un cliente es la cantidad de dinero que tiene en todas sus cuentas bancarias. El cl
```

```
```python  
class Solution:  
  
    def maximumWealth(self, accounts: [[int]]) -> int:  
        max = 0  
        for account in accounts:  
            s = sum(account)  
            if max < s:  
                max = s  
        return max
```

En este código, la clase `Solution` contiene un método llamado `maximumWealth` que toma una lista de listas de enteros (`accounts`) como entrada y devuelve un entero. El objetivo es encontrar la riqueza máxima entre todas las cuentas. La riqueza de una cuenta se calcula sumando todos los valores en la lista correspondiente. El código recorre cada cuenta, calcula su suma y compara esta suma con el valor máximo actual. Si la suma es mayor que el valor máximo actual, se actualiza el valor máximo. Finalmente, se devuelve el valor máximo encontrado.

```
print(Solution().maximumWealth([[1,2,3],[3,2,1]]))
```

1470. Barajar el Array

Dado el arreglo `nums` que consiste en $2n$ elementos en la forma $[x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n]$.

Devuelve el arreglo en la forma [x₁,y₁,x₂,y₂,...,x_n,y_n].

```
class Solution:

    def shuffle(self, nums: [int], n: int) -> [int]:
        ns1 = nums[:n]
        ns2 = nums[n:]
        ns = []
        for i in range(n):
            ns.append(ns1[i])
            ns.append(ns2[i])
        return ns
```

print(Solución().mezclar([2,5,1,3,4,7], 3))

1512. Número de Pares Buenos

```
> Dado un arreglo de enteros `nums`.
>
> Un par `(i,j)` se llama *bueno* si `nums[i]` == `nums[j]` y `i` < `j`.
>
> Devuelve el número de pares *buenos*.
```

```
```python
class Solution:

 def numIdenticalPairs(self, nums: [int]) -> int:
 j = 1
 n = len(nums)
 p = 0
 while j < n:
 for i in range(j):
 if nums[i] == nums[j]:
 p += 1
 j += 1
 return p
```

El código anterior define una clase Solution con un método numIdenticalPairs que cuenta el

número de pares idénticos en una lista de números enteros. Aquí está la explicación en español:

**1. Inicialización de variables:**

- `j` se inicializa en 1, que se utilizará como índice para recorrer la lista.
- `n` almacena la longitud de la lista `nums`.
- `p` se inicializa en 0 y se utilizará para contar el número de pares idénticos.

**2. Bucle while:**

- El bucle `while` se ejecuta mientras `j` sea menor que `n` (es decir, mientras no se haya recorrido toda la lista).
- Dentro del bucle, se utiliza un bucle `for` para comparar el elemento en la posición `j` con todos los elementos anteriores (desde la posición 0 hasta `j-1`).

**3. Condición if:**

- Si se encuentra que `nums[i]` es igual a `nums[j]`, se incrementa el contador `p` en 1, indicando que se ha encontrado un par idéntico.

**4. Incremento de `j`:**

- Después de comparar el elemento en la posición `j` con todos los elementos anteriores, se incrementa `j` en 1 para pasar al siguiente elemento de la lista.

**5. Retorno del resultado:**

- Finalmente, el método devuelve el valor de `p`, que es el número total de pares idénticos encontrados en la lista.

Este algoritmo tiene una complejidad temporal de  $O(n^2)$  en el peor de los casos, ya que para cada elemento en la lista, se compara con todos los elementos anteriores.

**print(Solución().numIdenticalPairs([1,2,3,1,1,3]))**

## 771. Joyas y Piedras

```
> Se te proporcionan dos cadenas: `jewels`, que representa los tipos de piedras que son joyas, y `stones`
>
> Las letras son sensibles a mayúsculas y minúsculas, por lo que `"a"` se considera un tipo de piedra d
```

```

```python
class Solution:

    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        n = 0
        for i in range(len(jewels)):
            js = jewels[i:i+1]
            n += stones.count(js)
        return n
```

```

El código anterior es una solución en Python para contar cuántas piedras (`stones`) son joyas (`jewels`). No es necesario traducir el código, ya que los nombres de las variables y la lógica del programa son universales en el contexto de la programación. Sin embargo, si necesitas una explicación en español, aquí está:

Este código define una clase `Solution` con un método `numJewelsInStones` que toma dos cadenas de texto como entrada: `jewels` (que representa los tipos de joyas) y `stones` (que representa las piedras que tienes). El método cuenta cuántas de las piedras son joyas y devuelve ese número.

1. Se inicializa una variable `n` a 0 para almacenar el conteo de joyas.
2. Se recorre cada carácter en la cadena `jewels`.
3. Para cada carácter en `jewels`, se cuenta cuántas veces aparece en la cadena `stones` usando el método `count`.
4. El conteo se suma a `n`.
5. Finalmente, se devuelve el valor de `n`, que es el número total de piedras que son joyas.

Este código es una forma sencilla de resolver el problema utilizando operaciones básicas de cadenas en Python.

**print(Solución().numJewelsInStones("aA", "aAAbbb"))**

```
1603. Diseño de Sistema de Estacionamiento
```

```
> Diseña un sistema de estacionamiento para un estacionamiento. El estacionamiento tiene tres tipos de
>
> Implementa la clase `ParkingSystem`:
>
```

```

> - `ParkingSystem(int big, int medium, int small)` Inicializa un objeto de la clase `ParkingSystem`. El constructor recibe tres enteros que representan el número de espacios disponibles para coches grandes, medianos y pequeños respectivamente.
> - `bool addCar(int carType)` Verifica si hay un espacio de estacionamiento disponible del tipo `carType` y lo agrega al sistema de estacionamiento. Si hay espacio, se resta uno del contador correspondiente y se retorna `True`. Si no hay espacio, se retorna `False`.

```python
class ParkingSystem:

    slots = [0, 0, 0]

    def __init__(self, big: int, medium: int, small: int):
        self.slots[0] = big
        self.slots[1] = medium
        self.slots[2] = small

    def addCar(self, carType: int) -> bool:
        if self.slots[carType - 1] > 0:
            self.slots[carType - 1] -=1
            return True
        else:
            return False

```

Traducción:

```

def addCar(self, carType: int) -> bool:
    if self.slots[carType - 1] > 0:
        self.slots[carType - 1] -=1
        return True
    else:
        return False

```

Nota: El código no necesita traducción, ya que es un bloque de código en Python y debe mantenerse en su idioma original para que funcione correctamente.

```

# parkingSystem = ParkingSystem(1, 1, 0)
# print(parkingSystem.addCar(1))
# print(parkingSystem.addCar(2))
# print(parkingSystem.addCar(3))
# print(parkingSystem.addCar(1))

```

En este código, se crea una instancia de `ParkingSystem` con 1 espacio para coches grandes, 1 espacio para coches medianos y 0 espacios para coches pequeños. Luego, se intenta agregar coches de diferentes tipos (1 para grande, 2 para mediano, 3 para pequeño) y se imprime el resultado de cada operación.

1773. Contar elementos que coinciden con una regla

Se te da un arreglo `items`, donde cada `items[i] = [typei, colori, namei]` describe el tipo, color y nombre del i -ésimo elemento. También se te da una regla representada por dos cadenas, `ruleKey` y `ruleValue`.

Se dice que el i -ésimo elemento coincide con la regla si **una** de las siguientes condiciones es verdadera:

- `ruleKey == "type"` y `ruleValue == typei`.
- `ruleKey == "color"` y `ruleValue == colori`.
- `ruleKey == "name"` y `ruleValue == namei`.

Devuelve *el número de elementos que coinciden con la regla dada*.

```
class Solution:

    def countMatches(self, items: [[str]], ruleKey: str, ruleValue: str) -> int:
        i = 0
        if ruleKey == "type":
            i = 0
        elif ruleKey == "color":
            i = 1
        else:
            i = 2
        n = 0
        for item in items:
            if item[i] == ruleValue:
                n += 1
        return n
```

El código anterior define una clase `Solution` con un método `countMatches` que cuenta cuántos elementos en una lista de ítems coinciden con una regla específica. La regla se define por una clave (`ruleKey`) y un valor (`ruleValue`). Dependiendo de la clave, se verifica un atributo

específico del ítem (tipo, color o nombre) y se cuenta cuántos ítems coinciden con el valor dado.

```
print(Solución().contarCoincidencias([["teléfono","azul","pixel"],[  
"color", "plateado"]])
```

```
## 1365. Cuántos números son menores que el número actual
```

```
> Dado el arreglo `nums` , para cada `nums[i]` encuentra cuántos números en el arreglo son menores que él.  
>  
> Devuelve la respuesta en un arreglo.
```

```
> ````
```

```
> Entrada: nums = [8,1,2,2,3]
```

```
> Salida: [4,0,1,1,3]
```

```
> Explicación:
```

```
> Para nums[0]=8 existen cuatro números menores que él (1, 2, 2 y 3).
```

```
> Para nums[1]=1 no existe ningún número menor que él.
```

```
> Para nums[2]=2 existe un número menor que él (1).
```

```
> Para nums[3]=2 existe un número menor que él (1).
```

```
> Para nums[4]=3 existen tres números menores que él (1, 2 y 2).
```

```
> ````
```

```
```python
```

```
class Solution:
```

```
 def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
 ns = []
 l = len(nums)
 for i in range(l):
 n = 0
 for j in range(l):
 if i != j:
 if nums[j] < nums[i]:
 n += 1
 ns.append(n)
```

```
 return ns
```

El código anterior define una clase `Solution` con un método `smallerNumbersThanCurrent` que toma una lista de números enteros `nums` y devuelve una lista `ns` donde cada elemento representa la cantidad de números en `nums` que son menores que el número en la posición correspondiente.

El código no necesita traducción, ya que es un fragmento de código en Python y los nombres de las variables y métodos son universales en el contexto de programación.

## **print(Solución().númerosMenoresQueActual([8,1,2,2,3]))**

Tiempo de ejecución: 528ms, superando al 11.81% de los programas. Vamos a optimizarlo.

```
```python
class Solution:

    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
```

En este fragmento de código, se define una clase `Solution` con un método `smallerNumbersThanCurrent`. Este método toma una lista de números enteros (`nums`) como entrada y devuelve una lista de enteros. La variable `l` se utiliza para almacenar la longitud de la lista `nums`.

```
sort_nums = nums.copy()

ins = list(range(l))
for i in range(l):
    for j in range(i+1, l):
        if sort_nums[i] > sort_nums[j]:
            a = sort_nums[i]
            sort_nums[i] = sort_nums[j]
            sort_nums[j] = a

        a = ins[i]
        ins[i] = ins[j]
        ins[j] = a
```

```

smalls = [0]
for i in range(1, l):
    if sort_nums[i-1] == sort_nums[i]:
        smalls.append(smalls[i-1])
    else:
        smalls.append(i)

# print(sort_nums)
# print(smalls)

r_is = list(range(l))
for i in ins:
    r_is[ins[i]] = i

ns = []
for i in range(l):
    ns.append(smalls[r_is[i]])
return ns

```

print(Solución().númerosMenoresQueElActual([8,1,2,2,3]))

Esta prueba tomó `284ms`, menos que los `528ms` anteriores.

Aquí tienes la traducción al español:

Usa las funciones abreviadas del sistema para escribirlo.

```

```python
class Solution:

 def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
 sort_nums = nums.copy()
 sort_nums.sort()

 ns = []
 for num in nums:
 ns.append(sort_nums.index(num))

```

```
 return ns
```

Traducción al español:

```
class Solution:

 def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
 sort_nums = nums.copy()
 sort_nums.sort()

 ns = []
 for num in nums:
 ns.append(sort_nums.index(num))
 return ns
```

El código permanece igual, ya que los nombres de las variables y las funciones no se traducen. La funcionalidad del código es calcular cuántos números son menores que cada elemento en la lista `nums`.

## **print(Solución().númerosMenoresQueActual([8,1,2,2,3]))**

Esto solo toma `64ms`, superando al `71%` de las entregas.

```
```python
class Solution:

    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
        ns = [0] * l
        for i in range(l):
            for j in range(i+1, l):
                if nums[i] > nums[j]:
                    ns[i] +=1
                elif nums[i] < nums[j]:
                    ns[j] +=1
                else:
                    pass
        return ns
```

```
print(Solución().númerosMenoresQueActual([8,1,2,2,3]))
```

He encontrado otra solución. Tiempo de ejecución: `400ms`.

```
```python
class Solution:

 def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
 ss = sorted((e,i) for i,e in enumerate(nums))

 l = len(nums)
 smalls = [0]
 for i in range(1, l):
 (e0, j0) = ss[i-1]
 (e1, j1) = ss[i]
 if e0 == e1:
 smalls.append(smalls[i-1])
 else:
 smalls.append(i)

 ns = [0]*l
 for i in range(l):
 (e, j) = ss[i]
 ns[j] = smalls[i]
 return ns
```

```
print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))
```

```
> Tiempo de ejecución: 52 ms, más rápido que el 91.45% de las soluciones en Python3 en línea para "How Many Numbers Are Smaller Than the Current Number"
>
> Uso de memoria: 14.6 MB, menos que el 15.18% de las soluciones en Python3 en línea para "How Many Numbers Are Smaller Than the Current Number"
```

;Finalmente lo logré! Este método es aún más rápido, superando al `91.45%` de las presentaciones.

Continúa simplificando.

```
```python
```

```

class Solution:

    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ss = sorted((e,i) for i,e in enumerate(nums))

    l = len(nums)
    smalls = [0]
    ns = [0]*l

    for i in range(1, l):
        (e0, j0) = ss[i-1]
        (e1, j1) = ss[i]
        if e0 == e1:
            smalls.append(smalls[i-1])
        else:
            smalls.append(i)

    ns[j1] = smalls[i]
    return ns

```

print(Solución().númerosMenoresQueElActual([8,1,2,2,3]))

Continúa.

```

```python
class Solution:

 def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
 ss = sorted((e,i) for i,e in enumerate(nums))

 l = len(nums)
 last = 0
 ns = [0] * l

 for i in range(1, l):
 (e0, j0) = ss[i - 1]
 (e1, j1) = ss[i]
 if e0 == e1:
 pass
 else:
 last = i

```

```
 ns[j1] = last
 return ns
```

## **print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))**

En este momento, hemos alcanzado un tiempo de ejecución de `40ms`, superando al `99.81%` de los programadores.

```
> Tiempo de ejecución: 40 ms, más rápido que el 99.81% de las soluciones en Python3 en línea para "How Many Numbers Are Smaller Than the Current Number"
>
> Uso de memoria: 14.4 MB, menos que el 15.18% de las soluciones en Python3 en línea para "How Many Numbers Are Smaller Than the Current Number"
```

Aquí tienes otra solución.

```
```python
class Solution:

    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
        n = [0] * 101
        max_num = 0
        for num in nums:
            n[num] += 1
            if num > max_num:
                max_num = num

        sm = [0] * (max_num + 1)
        sum = 0
        for i in range(max_num + 1):
            sm[i] = sum
            sum += n[i]

        ns = [0] * l
        for i in range(l):
            ns[i] = sm[nums[i]]

        devolver ns
```

```
print(Solución().númerosMenoresQueActual([8,1,2,2,3]))
```

Aquí tienes uno un poco más complejo.

```
```python
class Solution:

 def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
 l = len(nums)
 n = [0] * 101
 max_num = 0
 for num in nums:
 n[num] += 1
 if num > max_num:
 max_num = num

 short_n = []
 short_num = [] * 1
 zn = [0] * 101
 j = 0
 for i in range(max_num+1):
 if n[i] > 0:
 zn[i] = j
 short_n.append(n[i])
 short_num.append(num)
 j += 1

 sm = [0] * j
 sum = 0
 for i in range(j):
 sm[i] = sum
 sum += short_n[i]

 ns = [0] * l
 for i in range(l):
 ns[i] = sm[zn[nums[i]]]
 return ns
```

```
print(Solución().númerosMenoresQueActual([8,1,2,2,3]))
```

```
```python
class Solution:

    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        max_num = max(nums)

        n = [0] * (max_num + 1)
        for num in nums:
            n[num] += 1

        sorted_ls = []
        for i in range(max_num + 1):
            if n[i] > 0:
                sorted_ls.append(i)

        sm = [0] * (max_num + 1)
        sum = 0
        for i in range(len(sorted_ls)):
            v = sorted_ls[i]
            sm[v] = sum
            sum += n[v]

        ns = []
        for i in range(len(nums)):
            ns.append(sm[nums[i]])
        return ns

# print(Solution().smallerNumbersThanCurrent([72,48,32,16,10,59,83,38,1,4,68,7,67,16,5,35,99,15,55,11,20]))
```

Práctica

- Los estudiantes resuelven problemas similares a los mencionados anteriormente.