

SOLUCIÓN EVALUACIÓN CONTINUA 7

Autores: Ana Silvia Cordero Ricaldi, Leonel Leodolfo Campuzano Diestra, Yulinio Zavala Mariño

Implement queue using stacks

Solución de alto nivel:

Para implementar una cola mediante pilas, usaremos dos pilas (s1 y s2) para lograr FIFO . La idea clave es usar una pila para las operaciones de encolado y la otra para invertir el orden de las operaciones de desencolado.

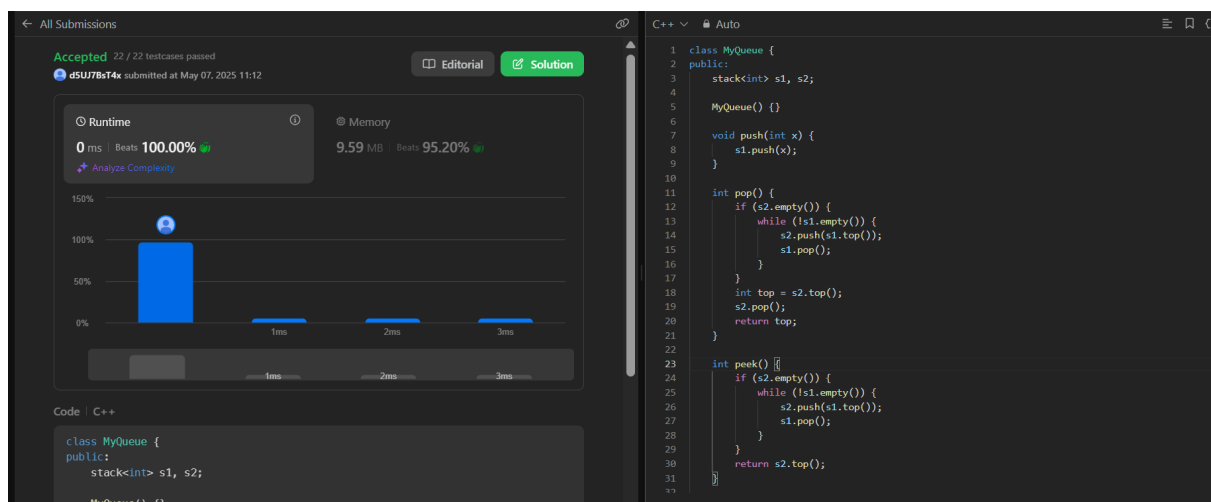
Complejidad Temporal:

O(n): O(n): Es O(n) ya que dependiendo de la cantidad hasta el elementos iteramos solo una vez para moverlo entre los stacks y las subsiguientes veces la complejidad es de O(1) para remover. En caso de las otras operaciones es O(1)

Complejidad Espacial:

O(n): Es O(n) ya que los elementos están en s1 o s2, nunca se excede de O(n)

Captura de LeetCode:



Daily Temperatures

Solución de alto Nivel:

Para obtener el número de días que tienes que esperar después del i -ésimo día para una temperatura mayor utilizaremos una pila para almacenar el índice de los días que no se encuentra una temperatura mayor. En caso de encontrar un día con una temperatura mayor, calcularemos cuántos días se llegó a esperar actualizando nuestro nuevo vector.

Complejidad Temporal:

Es $O(n)$, ya que necesitamos recorrer todas las temperaturas del vector para poder agregar un día con una temperatura más alta que el anterior y eliminar los elementos para el número de días de espera. Cada proceso de agregar o eliminar es $O(1)$, pero al recorrer todo un vector, se vuelve un bucle con una complejidad $O(n)$.

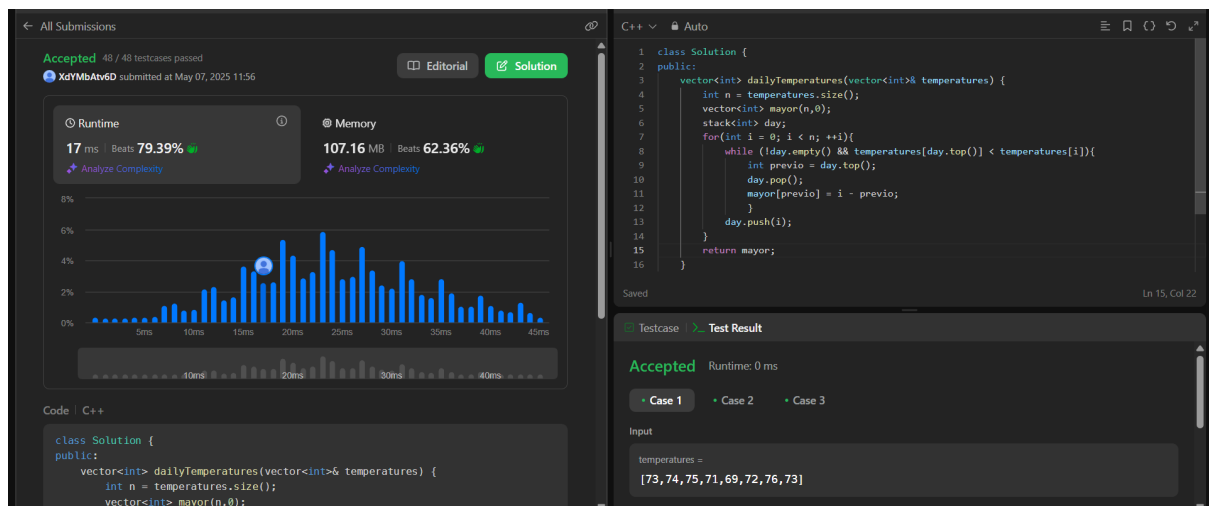
Complejidad Espacial:

Tenemos un vector que tendrá un tamaño $n \rightarrow O(n)$

Tenemos un stack day, en el peor de los casos, tendrá que almacenar todos los índices de los días $\rightarrow O(n)$

Por lo tanto, la complejidad espacial será $O(n)$

Captura de LeetCode:



Find the Winner of the Circular Game

Solución de alto nivel:

Para resolver el problema de encontrar al ganador en un juego circular de eliminación, modelamos el conjunto de jugadores usando una cola donde cada jugador representa un número del 1 al n . En cada ronda, simulamos el conteo hasta el k -ésimo jugador rotando la cola: movemos los primeros $k-1$ jugadores al final y luego eliminamos al jugador que queda al frente, que es el k -ésimo. Este proceso se repite hasta que quede un solo jugador en la cola, quien será el ganador del juego.

Usamos cola (FIFO) porque modela bien el tipo de rotación el problema planteado, es decir, podemos sacar el primer jugador con `pop()` y moverlo al final con `push`.

Complejidad Temporal:

- Mover $k-1$ jugadores al final \rightarrow Cada movimiento es una combinación de `pop()` y `push()`, ambos $O(1)$.
- Eliminar al jugador $k \rightarrow \text{pop}()$ ($O(1)$).

Se ejecuta $n - 1$ veces.

Por tanto:

$$(k-1+1) \times (n-1) = k \times (n-1)$$

Finalmente la complejidad temporal:

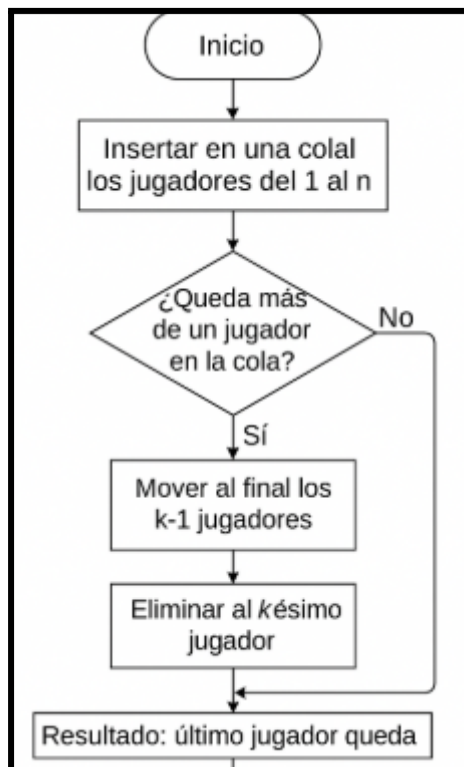
$$O(n \cdot k)$$

Complejidad Espacial:

Al inicio, la cola tiene " n " elementos y va disminuyendo hasta 1.

Por lo tanto, el espacio utilizado es lineal en función de n : **$O(n)$** .

Diagrama:



Captura de Leetcode:

Problem List

Run

Submit

Premium

Description

Accepted

Editorial

Solutions

Submissions

All Submissions

Accepted 95 / 95 testcases passed

slcVmeuB4M submitted at May 07, 2025 10:34

Editorial

Solution

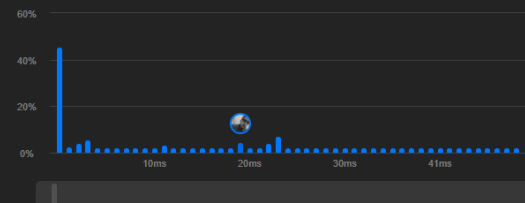
Runtime

19 ms | Beats 28.52%

Analyze Complexity

Memory

26.43 MB | Beats 14.14%



Code | C++

Code

C++

Auto

Ln 10, Col 42

```
2 public:
3     int findTheWinner(int n, int k) {
4         queue<int> q;
5         for (int i = 1; i <= n; ++i) {
6             q.push(i);
7         }
8
9         while (q.size() > 1) {
10            for (int i = 1; i < k; ++i) {
11                q.push(q.front());
12                q.pop(); // mover al final los k-1 jugadores
13            }
14            q.pop(); // eliminar al k-ésimo jugador
15        }
16        return q.front(); // el último que queda
17    }
```

Saved

Testcase

Test Result

Accepted Runtime: 0 ms

Case 1

Case 2