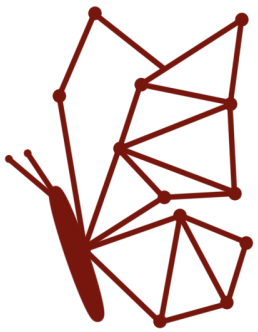

Camp. de Otoño 2025 - Segundo concurso (intermedio)

12 de noviembre, 2025

Libro de soluciones

Este documento contiene las soluciones esperadas para los 7 problemas usados en el segundo concurso de nivel intermedio.

Las siguientes personas apoyaron desarrollando el set de problemas ya sea creando o mejorando enunciados, soluciones, casos de prueba, verificadores de entradas y salidas:

Ignacio Benitez Salvador

Brandom Galder Hernández

Nicole Abigail Chow-Flores

Eduardo Soria

Ashley Torres

Ulises Refugio

Christopher Gael Contla

Carlos Alberto Lara

Sandro Martínez

Ángel David Franco

D22O25. Todo sea por los minions

Por: Brandom Galder Hernández

Pista 1: Gru puede ir y venir a la misma plataforma las veces que sea.
Pista 2: Cuando Gru empieza en una plataforma cuyo número es divisible entre 5, no puede hacer nada.

Solución:
Para este problema existen dos casos: cuando la posición X de Gru es divisble entre 5, y cuando no lo es.
Cuando X es divisble entre 5, Gru no puede hacer nada porque, en el segundo cero, Vector disparará a dicha plataforma y caerá con los minions morados. Caso contrario, Gru siempre puede evitar las casillas divisibles entre 5. Por ejemplo, si $X = 4$, Gru puede moverse únicamente entre las posiciones 4 y 3, evitando así caer en la posición 5 (la posición más cercana que es divisble entre 5).

Complejidad: Los condicionales y las impresiones por consola tienen complejidad $O(1)$, por lo tanto, la complejidad total es $O(2) \approx O(1)$.

C21O25. Basta

Por: Eduardo Soria

Pista: Todo elemento S_i con un índice $i \in [l, r]$ no deberá ser impresos.

Solución:
Para cada travesura de Flaco, deberás de recorrer cada uno de los caracteres de tu cadena de texto $S_1, S_2, S_3, ..., S_N$ (siendo $N = |S|$), imprimiendo los caracteres S_i cuyo subíndice $i \notin [l, r]$, es decir, si el subíndice de S_i cumple $i < l$ o cumple $i > r$.

Complejidad: Debido a que habrá un total de Q travesuras y recorrer la cadena de texto tiene complejidad $O(N)$ por cada una, la complejidad final es de $O(Q \cdot N)$.

C22O25. Gracias Jason Funderburker

Por: Brandom Galder Hernández

Pista 1: Ordenar el arreglo para identificar los pares de tumbas consecutivas.
Pista 2: La distancia entre dos tumbas se obtiene con una resta entre sus posiciones.

Solución:
Un par de tumbas consecutivas $\{P_i, P_j\}$ es aquel que, entre estas, no existen tumbas intermedias, cumpliendo $(i < j)$. Para encontrar rápidamente los pares $\{P_i, P_j\}$ en el arreglo, es conveniente ordenar el arreglo P, para que los pares $\{P_i, P_{i+1}\}$ sean pares de tumbas consecutivas. La distancia entre dos tumbas P_i y P_j es igual a

$$d(P_i, P_j) = P_j - P_i$$

Entonces, la solución consiste en calcular $d(P_i, P_{i+1})$ para cada i tal que $(1 \leq i < N)$ y, si por lo menos una $d(P_i, P_{i+1}) = D$, Jason Funderburker cumplió. Caso contrario, Jason Funderburker no cumplió.

Complejidad: La función $sort()$ de C++ tiene complejidad $O(N\ log(N))$ y la complejidad de recorrer el arreglo es $O(N)$, resultando una complejidad total de $O(N \cdot\ log(N) + N)$.

C23O25. Geeble y las calabazas

Por: Hector Ulises Refugio Becerril

Pista 1: Debemos contar cuántas calabazas hay en cada fila.
Pista 2: Por cada fila, sabiendo el número de calabazas, es conveniente usar la segunda máquina únicamente cuando el número de calabazas $> M$.

Solución:
Siendo A_i una fila, en esta debemos guardar la cantidad de calabazas que hay en esta. Por cada fila A_i, de acuerdo al número de calabazas que tiene, debemos de realizar la siguiente comparación comparación:

$$\begin{cases} \text{cantidad de calabazas en } A \leq M & \text{conviene usar la primera máquina} \\ \text{cantidad de calabazas en } A > M & \text{conviene usar la segunda máquina} \end{cases}$$

de esta forma decidimos cuanto aumentar a nuestro contador de monedas.

Por ejemplo, se tiene una segunda máquina que elimina todas las calabazas por un costo $M = 5$. Si una fila A_i tiene 3 calabazas, conviene quitar una por una por un costo de 1, dando un total de 3 monedas. Caso contrario, tenemos otra fila A_i que tiene 10 calabazas, donde conviene quitar todas de una por un costo total de 5, porque quitar una por una tendría un costo total de 10.

Complejidad: Necesitamos iterar sobre todas las filas que contengan alguna calabaza, por lo que las complejidades podrían ser $O(N)$ u $O(NlogN)$ dependiendo la estructura de datos que se use.

C24O25. El pan más barato de Don Bartolomeo

Por: Ashley Torres

Pista 1: Observa que las ventanas de tamaño K se solapan entre sí.
Pista 2: No es necesario volver a sumar los K elementos cada vez. Es posible actualizar la suma anterior en tiempo constante, $O(1)$.

Solución:
Se utiliza la técnica de **sliding window**, la cual permite mantener la información de K posiciones consecutivas para cada $1 \leq i \leq n - K + 1$. Esta técnica consiste en procesar un bloque fijo de tamaño K mientras la ventana se desplaza una posición a la vez a lo largo del arreglo, véase aquí: `https://www.geeksforgeeks.org/dsa/window-sliding-technique/`.

Primero se calcula la suma de los primeros K panes. Luego, para cada nueva posición de la ventana, la suma se actualiza en tiempo $O(1)$: siendo i la nueva posición de la ventana, se resta el precio del pan $i-1$ que sale de la ventana y se agrega el precio del pan $i+K-1$ que entra. Gracias a esta actualización constante, no es necesario recalcular la suma completa del bloque en cada desplazamiento.

Durante todo el recorrido se mantiene un registro de la suma mínima obtenida. Con este enfoque se recorre el arreglo una sola vez en $O(N - K + 1)$ y se obtiene la suma mínima de cualquier bloque consecutivo de tamaño K de forma eficiente.

Complejidad: La suma inicial toma $O(K)$ y el recorrido completo $O(N - K + 1)$. Por lo tanto, la complejidad total es $O([N - K + 1] + K) = O(N + 1)$.

C25O25. ¿Difícil para quién?

Por: Ignacio Benitez Salvador

Pista 1: Piensa cómo usar suma de acumulados para optimizar las actualizaciones.
Pista 2: Intenta "marcar" en tu arreglo únicamente los cambios que ocurren al iniciar y al terminar cada uno de los rangos.

Solución:
Este problema pide aplicar muchas actualizaciones por rango: para cada par $[L, R]$, debemos aumentar en 1 todas las posiciones desde L hasta R.

Si intentamos actualizar cada rango directamente recorriendo desde L hasta R, el costo sería demasiado grande: en el peor caso, $Q = 10^6$ y cada rango puede ser tan grande como $N = 10^6$. Esto produciría una solución de $O(N \cdot Q)$ esto excede el tiempo límite.

La clave está en usar un **arreglo para marcar los rangos**, en lugar de sumar 1 a todas las posiciones del rango, solo realizamos dos marcas.:

- Marcamos el inicio del aumento:
  - $a[L] += 1$
- Marcamos el final del aumento:
  - $a[R + 1] -= 1$

Estas marcas indican "a partir de aquí empieza un $+1$" y "a partir de aquí deja de aplicarse el $+1$".

Después de procesar los Q rangos, el arreglo todavía no representa la respuesta final. Para obtener los valores verdaderos, aplicamos una **suma de acumulados**:

$$a[i] = a[i] + a[i - 1]$$

Es decir, acumulamos los cambios para construir los incrementos reales en cada posición.

Complejidad: La técnica de marcar los rangos permite procesar los Q rangos en $O(Q)$ y construir el arreglo final en $O(N)$, por lo que la complejidad total es $O(N + Q)$.

C26O25. CATastrofe

Por: Ignacio Benitez Salvador

Pista 1: Debido a que solo buscas saber si existe un camino posible, tanto DFS como BFS son útiles para solucionar el problema
Pista 2: Ten en cuenta que el grafo creado por el movimiento de Maullín en la biblioteca es bidireccional

Solución:
El problema nos pide saber, para distintas posiciones de Maullín dentro de la biblioteca, si puede escapar llegando a la celda (N, M). Para ello puede moverse en 8 direcciones, pero debe respetar ciertas reglas:
- Maullín solo puede situarse en celdas cuyo carácter sea un punto (.).
- Los movimientos diagonales únicamente son válidos si las dos celdas intermedias no contienen fuego ($*$).

Una forma directa de verificar si Maullín puede llegar a la salida desde una posición es hacer un recorrido BFS o DFS desde cada consulta. Sin embargo, esto sería muy ineficiente, ya que cada consulta costaría $O(N \cdot M)$ y existen Q consultas.

La idea clave para solucionar el problema de forma eficiente es notar que todos los movimientos permitidos son bidireccionales. Esto significa que si es posible ir del punto A al punto B, entonces también es posible ir del punto B al punto A.

Gracias a esto, en lugar de verificar cada consulta por separado, primero realizamos un único recorrido (BFS o DFS) empezando desde la salida (N, M), marcando todas las celdas desde las cuales se puede alcanzar la salida. Una forma de hacer esto es modificando la matriz en el proceso para indicar qué celdas fueron visitadas. Después, para cada consulta, únicamente debemos verificar si la celda consultada quedó marcada durante el recorrido inicial.

Complejidad: La precomputación con BFS o DFS desde la celda (N, M), en el peor caso, recorrerá toda la matriz, lo cual toma $O(N \cdot M)$. Cada consulta posterior se resuelve en $O(1)$, por lo que la complejidad total es $O(N \cdot M + Q) = O(N \cdot M)$.