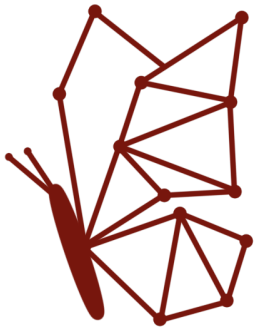

Camp. de Otoño 2025 - Segundo concurso (básico)

12 de noviembre, 2025

Libro de soluciones

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

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

Brandom Galder Hernández

Nicole Abigail Chow-Flores

Eduardo Soria

Ashley Torres

Ulises Refugio

Christopher Gael Contla

Carlos Alberto Lara

Sandro Martínez

Ángel David Franco

D21O25. Contraseña flotante

Por: Brandom Galder Hernández

Pista: La división es la operación que divide en partes iguales una cantidad.

Solución:
Para repartir equitativamente la cantidad de poder P que cada catrina debe utilizar para construir la ofrenda, debemos dividir el poder requerido T entre la cantidad de catrinas N.

$$P = \frac{T}{N}$$

Complejidad: Realizar una división tiene complejidad $O(1)$, y hacer una impresión también, entonces la complejidad es $O(2)$, prácticamente $O(1)$.

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 $O(2) \approx O(1)$.

D23O25. Marcador Final

Por: Nicole Abigail Chow-Flores

Pista 1: El perdedor te da una pista sobre el ganador de la ronda.
Pista 2: Revisa el orden de las rondas e identifica quien *sí* juega en esa ronda.

Solución:
Para resolver este problema, analicemos cada ronda.

1. Determinar el ganador de la Ronda 1
   Como solo compiten George y Aremi.
   - Si S1 = George, entonces gana Aremi.
   - Si S1 = Aremi, entonces gana George.

   Sea G1 el ganador de la ronda 1.
2. Determinar el ganador de la Ronda 2
   En esta ronda compiten: G1 (ganador de la ronda 1) contra Valeria.

   Entonces:
   - Si S2 = G1, significa que G1 perdío, por lo tanto gana Valeria.
   - Si S2 = Valeria, entonces gana G1.

Finalmente debe imprimir:

1. G1 (el ganador de la Ronda 1)
2. G2 (el ganador de la Ronda 2)

Complejidad: Todas las comparaciones se realizan con un número constante de nombres y condiciones, por lo tanto, la complejidad del algoritmo es: $O(1)$.

D24O25. Pibble el programador

Por: Hector Ulises Refugio Becerril

Pista 1: Calcula la respuesta para los valores desde 1 hasta 5 para N y observa si existe algún patrón.
Pista 2: Analiza cuánto crece un resultado para un valor N con respecto al resultado de $N-1$.

Solución:
Para resolver este problema analicemos primero casos pequeños.

- Si $N = 1$ sólo hay un problema, entonces Pibble tiene únicamente una manera de resolver el concurso.
- Si $N = 2$ Pibble tiene 2 opciones para elegir cuál resolver primero, y luego sólo le queda 1. Por lo tanto hay 2 maneras.
- Si $N = 3$ Pibble tiene 3 opciones para elegir cuál resolver primero, luego tiene 2 opciones, y al final, sólo le queda 1. Por lo tanto hay 6 maneras.

Siguiendo este razonamiento nos damos cuenta que en el primer paso Pibble puede elegir cualquiera de los N problemas. Después de resolver uno, le quedan $N-1$ problemas por resolver, en el siguiente paso podrá elegir entre $N-2$, y así sucesivamente hasta que le quede uno por elegir.
Por lo tanto la cantidad total de maneras se obtiene multiplicando:

$$N \times (N-1) \times (N-2) \times ... \times 1$$

Esta operación matemática se conoce como factorial y se denota como $N!$

Complejidad: Debemos iterar de 1 hasta N por lo que la complejidad total es $O(N)$.

D25O25. Coincidencias de otoño

Por: Ashley Torres

Pista 1: Convierte ambos caracteres a minúsculas o mayúsculas antes de compararlos.
Pista 2: Existen funciones para invertir una cadena de texto.

Solución:
El problema pide contar cuántos caracteres coinciden entre una cadena S y su reverso R. Dos caracteres coinciden si son iguales sin importar si están en mayúscula o minúscula. Primero, se construye R invirtiendo la cadena S.

Para cada posición i, se hacen minúsculas (o mayúsculas) $S[i]$ y $R[i]$, y se compara si son iguales. Cada vez que coinciden, se incrementa un contador.

Dado que solo recorremos la cadena una vez y las operaciones de comparación son constantes, el procedimiento es eficiente incluso para el límite máximo de longitud.

Complejidad: Tanto invertir la cadena como compararla con su reverso tienen complejidad $O(|S|)$, dando una complejidad total de $O(|S| + |S|) = O(2 \cdot |S|) \approx O(|S|)$.

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 $O(N \cdot\ log(N) + N)$.

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 $O(N + Q)$.