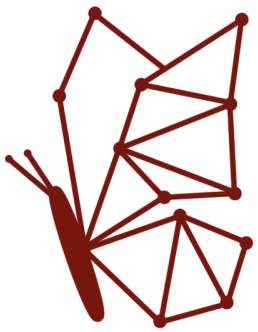

Campamento de Otoño 2025 - Cierre

18 de noviembre, 2025

Libro de soluciones

Este documento contiene las soluciones esperadas para los 10 problemas usados en el concurso de cierre.

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

Eduardo Soria

Jazmin Virgilio

Ulises Refugio

Christopher Gael Contla

Carlos Alberto Lara

Sandro Martínez

Ángel David Franco

D31O25. ¿Cuatro qué?

Por: Ignacio Benitez Salvador

Pista 1: Recuerda que un número es múltiplo de otro si la división no deja residuo.
Pista 2: Usa el operador módulo para determinar si un número es divisible entre 4.

Solución:
El problema únicamente pide verificar si el número N es divisible entre 4. Para ello, utilizamos la operación $N \bmod 4$:

- Si $N \bmod 4 = 0$, entonces N es múltiplo de 4 y debemos imprimir `CUATRO`.
- Si el residuo es distinto de 0, entonces imprimimos `NO`.

Complejidad: La verificación mediante el operador módulo y la condición se realiza en tiempo constante, por lo que la complejidad total es $O(1)$.

D32O25. El tamaño

Por: Ignacio Benitez Salvador

Pista 1: Por los límites, tienes un número de hasta cien dígitos. ¿Qué tipo de variable crees conveniente usar para poder leerlo e imprimirlo correctamente en c++?.
Pista 2: No lo manejes como un número.

Solución:
Ingresa la entrada como una variable tipo `string`, de esta forma puedes tratarla como un arreglo de 100 caracteres en el peor de los casos, listo para imprimirse de vuelta sin mayor complicación.

Complejidad: Dado que solo se realiza una lectura y una impresión de una cadena de tamaño fijo, la complejidad es constante, $O(1)$.

D33O25. Cuenta regresiva

Por: Ignacio Benitez Salvador

Pista 1: Basta con comenzar un ciclo en N y decrementar hasta llegar a 0.

Solución:
El problema pide imprimir una cuenta regresiva desde el número N hasta 0, inclusive. Para lograrlo, solo necesitamos un ciclo que inicie en N y vaya disminuyendo de uno en uno. En cada iteración imprimimos el valor actual del contador.

Un ciclo for sería:

- Iniciar en $i = N$
- Mientras $i \geq 0$, imprimir i y disminuir i en 1

Esto genera exactamente la secuencia pedida: $N, N-1, N-2, \ldots, 1, 0$.

Complejidad: Como solo recorremos los valores desde N hasta 0 una vez, la complejidad es $O(N)$.

D34O25. Humildad

Por: Ignacio Benitez Salvador

Pista 1: En un ciclo `for`, en cada iteración solo necesitas multiplicar N por el contador e imprimir el formato solicitado.
Pista 2: Revisa los límites del problema y elige el tipo de variable adecuado para evitar desbordamientos (overflow).

Solución:
El problema requiere imprimir la tabla de multiplicar del número N desde 1 hasta 10. Esto se logra con un ciclo que recorra los valores del 1 al 10. En cada paso del ciclo calculamos:

$$X = N \times i$$

y luego imprimimos exactamente el formato:

```
N * i = X
```

donde `N`, `i` y `X` son sustituidos por sus valores reales.

Es importante notar que el valor de N puede ser tan grande como 10^9. Al multiplicarlo por los valores de 1 a 10, el resultado puede llegar hasta 10^{10}.

Si usas una variable de tipo `int`, esta puede desbordarse, ya que su rango típico es aproximadamente hasta 2×10^9. Para evitar un desbordamiento (overflow), la multiplicación debe almacenarse en una variable de tipo `long long int`, que soporta valores mucho más grandes.

Complejidad: El ciclo siempre se ejecuta 10 veces sin importar el valor de N, por lo que la complejidad total es constante: $O(1)$.

C31O25. Wicked for stairs

Por: Brandom Galder Hernández

Pista: Dado que no hay repetidos y el orden debe ser descendiente, el ordenamiento de Dorothy debe cumplir $A_i > A_{i+1}$ para toda $1 \leq i < N$.

Solución:
El ordenamiento de Dorothy debe cumplir $A_i > A_{i+1}$ para toda $1 \leq i < N$. Como no hay repetidos, solo debemos ordenar los escalones de mayor a menor e imprimir los escalones.

Complejidad: El ordenamiento mediante `sort` en C++ tiene complejidad $O(N \log N)$. El arreglo se recorre una sola vez para imprimirlo, lo cual tiene complejidad $O(N)$. La complejidad total es $O(N + [N \cdot \ log(N)])$.

C32O25. Una fábrica frog-blemática

Por: Brandom Galder Hernández

Pista 1: Cuando haya más de un juguete con el mismo ID T_i, ya no hay que procesar los demás iguales a este.
Pista 2: Counting sort es un algoritmo que permite registrar la frecuencia en que aparece un elemento en $O(1)$.

Solución:
Cuando decimos frecuencia, nos referimos a la cantidad de apariciones que tiene un objeto en algún espacio. El algoritmo Counting Sort (cubetas) almacena la frecuencia de un elemento en $O(1)$, esto es útil porque de esta forma almacenamos la frecuencia de los N IDs de cada juguete. *Véase aquí:* `https://www.geeksforgeeks.org/dsa/counting-sort/`

Para aplicar cubetas a esta solución, contaremos la frecuencia de cada ID. Como el ID toma valores entre 1 y 10^6, basta con declarar un arreglo de tamaño 10^6 inicializado en 0. En dicho arreglo, el índice T_i representa el ID T_i, y el valor que contiene, su frecuencia.

Si la frecuencia de T_i es > 1, el juguete con ID T_i ya se considera como un juguete repetido y debemos incrementar nuestro contador en 1. No hace falta ordenar los IDs porque las frecuencias fueron registradas en un arreglo cuyos índices ya están ordenados.

Si la frecuencia del juguete $T_i > 1$, incrementamos en un 1 el contador de juguetes repetidos; en caso contrario, no hacemos nada. Después, solo habrá que imprimir la cantidad de repetidos y los IDs T_i con frecuencia > 0.

Complejidad:
Insertar la frecuencia de un ID se hace en $O(1)$, entonces, insertar N IDs se hace en $O(N)$. La complejidad de iterar sobre los N datos está sujeta al ID T_i más grande del arreglo (porque después de este ya no existen IDs). La complejidad total es $O(N + T_{grande})$.

C33O25. Palitos

Por: Eduardo Soria

Pista: La función puede $f(N)$ puede verse como una función recursiva

Solución:
La función está definida como:

$$f(N) = \begin{cases} \text{si } N \leq 20, & 1 \\ \text{si } N > 20, & f(N-5) + 5 + N \end{cases}$$

La solución recursiva es directa: para valores mayores a 20, hacemos una llamada a $f(N-5)$ y luego sumamos $5 + N$. Debido a que la llamada recursiva siempre reduce en 5 a N, eventualmente llegará a una $N \leq 20$, el cual será nuestro caso base que retornará 1.

Complejidad: EL número de operaciones realizadas para llegar al caso base es igual la cantidad de veces que se le tiene que restar 5 a N hasta llegar a un valor ≤ 20, lo que equivale a $O\left(\left[\frac{N-20}{5}\right]\right)$.

C34O25. Maripositas

Por: Jazmin Virgilio

Pista 1: Piensa el árbol como un arreglo de tamaño T, donde cada posición representa un punto del árbol; los espacios que ocupan las familias ya instaladas corresponden a intervalos que marcan posiciones ocupadas dentro de dicho arreglo.

Pista 2: La nueva familia requiere K posiciones libres consecutivas; es decir, el problema se reduce a determinar si existe un subarreglo de longitud K formado exclusivamente por ceros (posiciones libres).

Solución:
Sea el árbol modelado como un arreglo de tamaño T. Cada familia instalada ocupa un intervalo $[L, R]$, el cual puede registrarse en un arreglo, indicando únicamente el inicio y el final del intervalo. Al aplicar posteriormente una suma acumulada, se obtiene para cada posición del arreglo cuántas familias la ocupan.
Una vez construido este arreglo acumulado, se considera libre toda posición cuyo valor sea cero. De esta forma, el problema se transforma en verificar si, dentro del arreglo completo, existe una secuencia de ceros consecutivos cuya longitud sea mayor o igual a K.
Si existe al menos un subarreglo de longitud K compuesto únicamente por ceros, entonces la nueva familia puede instalarse; en caso contrario, no es posible hacerlo.

Complejidad: Cada uno de los N intervalos $[L, R]$ se procesa mediante dos operaciones en un arreglo, lo cual es una complejidad $O(N)$. Posteriormente, la solución construye el arreglo acumulado recorriendo todas las posiciones del árbol, lo que representa una complejidad $O(T)$. Finalmente, para verificar si existe un segmento libre, se realiza un recorrido lineal del arreglo acumulado contando cuántas posiciones consecutivas son iguales a cero; este paso también es $O(T)$. Por lo tanto, la complejidad total del algoritmo es $O(N + T)$.

C35O25. Nada

Por: Eduardo Soria

Pista 1: Nótese que si un valor X es valido para la solución, todo valor $Y < X$ también será valido, lo mismo para el caso contrario
Pista 2: Verificar si una X es valida puede realizarse con un solo recorrido de A.

Solución:
El problema plantea la idea de buscar un máximo valor X que cumpla que con la condición de que no existe ningún subarreglo continuo de longitud mayor a K de valores menores que X. Esta estructura de buscar el máximo o mínimo valor que satisface una condición es una señal clásica para considerar una búsqueda binaria sobre la respuesta.

En este caso, es posible resolver el problema usando una búsqueda binaria debido a que, si una X es valida, todo valor $Y < X$ lo será. La razón es simple: al disminuir X, el conjunto de tramos que quedan sumergidos (los tramos con altura menor que Y) es un subconjunto del conjunto de los que quedarían sumergidos con nivel X. Por lo tanto, al bajar el nivel del agua, nunca aparecerá un segmento continuo más grande que antes.

De forma análoga, si un valor X no es válido, entonces cualquier $Y > X$ tampoco podrá serlo, pues al aumentar el nivel del agua aumentará el número de tramos sumergidos, posiblemente formando segmentos aún más largos.

Definamos $f(X)$ como una función booleana que indica si con nivel de agua X Coco puede completar la carrera. Esta función puede ser evaluada haciendo un recorrido de todo el arreglo A, donde llevaremos un contador C de los tramos menores a X, el cual será reiniciado cada vez que encontremos un valor mayor, finalmente solo debemos retornar si en algún momento C fue mayor que K.

Debido a lo mencionado anteriormente, el dominio de $f(X)$ se divide en dos regiones:

$$\underbrace{\text{true, true, true, } \ldots}_{\text{valores válidos}}, \underbrace{\text{false, false, false, } \ldots}_{\text{valores no válidos}}$$

Nuestro objetivo es encontrar el máximo X para el cual $f(X)$ sigue siendo verdadero, esto es posible haciendo una búsqueda binaria sobre el dominio de la función.

Complejidad: La solución necesita de realizar una búsqueda binaria sobre el dominio de $f(X)$ el cual es el intervalo $[1, 10^9]$, y cada evaluación de la función necesitará de un recorrido del arreglo desde 1 hasta N. Por lo tanto la complejidad final es: $O(N \cdot log(10^9))$

C36O25. Washington

Por: Hector Ulises Refugio Becerril

Pista 1: Piensa en el algoritmo que se usa para encontrar la mínima distancia entre dos puntos. Ahora analiza, ¿conviene realizar este algoritmo por cada cliente o tienda que hay en el centro comercial, cuál sería la complejidad de esta solución?

O‘ **Pista 2:** La solución propuesta en la pista anterior es ineficiente, en el peor de los casos tenemos 10^6 clientes o tiendas en total, pero recordemos que tenemos un número de pasos K, que puede ser mayor a 1, lo que nos permite explorar más celdas y justamente hace que la solución previa supere el límite de tiempo. Piensa de qué manera podrías implementar el algoritmo una sola vez.

Solución:
La solución consiste en implementar una BFS multifuente. Seleccionamos las tiendas como las fuentes originales, de esta manera garantizamos la distancia mínima entre cada cliente a su tienda más cercana y que ningún camino válido quede omitido. A partir de esta BFS multifuente podemos construir una matriz de distancias y de esta manera contar cuántos clientes tienen una distancia a alguna tienda no mayor a K o contar directamente el número de clientes que cumplen esta condición durante el recorrido.

Complejidad: La BFS multifuente recorre la cuadrícula una sola vez, la BFS tiene una complejidad de $O(V + E)$, donde V es el número de nodos y E el número de aristas, en el caso de este problema tenemos $(N \times M)$ nodos y cada nodo tiene a lo mucho 4 vecinos por lo que la complejidad total es $O((N \times M) + 4 \times (N \times M))$, lo que se simplifica en una complejidad total de $O(N \times M)$.