Concurso Final 2025/2026

Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	Categoría
A - El torneo más corto	Ad-hoc.
B - Buscando oro	grafos, BFS/DFS
C - La isla de los programadores	voraz, bucles, condicionales
D - La peor entrevista	bucles, cadenas
E - Números de Lychrel	pensar, matemáticas, long
F - Evitando Lag	grafos, floyd-warshall
G - Plan de Movilidad	grafos, dijkstra

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - El torneo más corto	10	88 KB
B - Buscando oro	7	2,85 MB
C - La isla de los programadores	20	428 KB
D - La peor entrevista	3	2,88 KB
E - Números de Lychrel	6	87 KB
F - Evitando Lag	5	65,3 KB
G - Plan de Movilidad	9	523 KB
- **Total**	60	4 MB

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - El torneo más corto	j.barrios.2025	19
B - Buscando oro	-	-
C - La isla de los programadores	jj.flores.2025	38
D - La peor entrevista	-	-
E - Números de Lychrel	-	-
F - Evitando Lag	-	-
G - Plan de Movilidad	-	-

*Antes de congelar el marcador.

Estadísticas*

Problema	Envíos	Válidos	% éxito
A - El torneo más corto	11	5	45.45 %
B - Buscando oro	-	-	- %
C - La isla de los programadores	3	2	66 %
D - La peor entrevista	-	-	- %
E - Números de Lychrel	-	-	- %
F - Evitando Lag	-	-	- %
G - Plan de Movilidad	-	-	- %

*Antes de congelar el marcador.

A. El torneo más corto

Envíos	Válidos	% éxito
11	5	45.45 %

A. El torneo más corto

La idea era que fuera el problema fácil del concurso.

Pensad en que todos los equipos deben ser eliminados en algún partido menos uno, el ganador.

No hace falta simular el torneo, se deben jugar partidos hasta que solo quede uno, es decir $N - 1$ partidos.

A. El torneo más corto

Esta es la solución en Python:

```
from math import dist

if __name__ == '__main__':
    c = int(input())
    for _ in range(n):
        sol = int(input()) -1
        print(sol)
```

B. Buscando oro

Envíos	Válidos	% éxito
-	-	- %

B. Buscando oro

Menos texto por favor... Dado una matriz de caracteres, que terreno tiene más oro para explotar. Cada local que se da es parte de una componente conexa, calcular el oro que se tiene en la componente conexa y quedarse con el máximo. En caso de empate, quedarse con el primero que venga.

B. Buscando oro

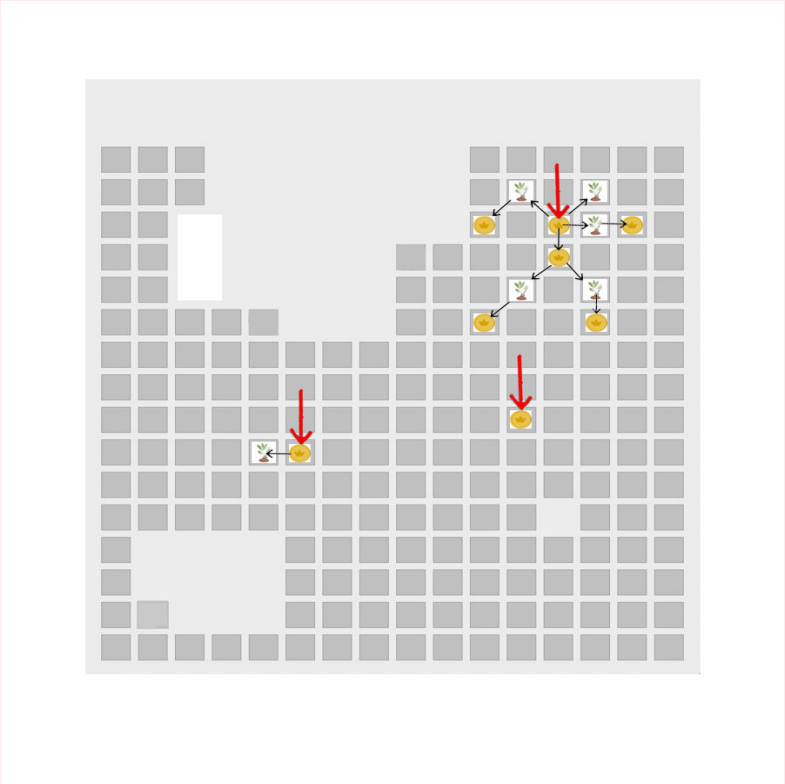


Figura: Imagen de un terreno de ejemplo para evaluar.

La solución consiste en con un algoritmo tipo BFS o DFS recorrer la componente conexa del local daba, quedarse con la que más oro tenga e imprimir por pantalla.
Complejidad esperada $O|V| + |E|$

C. La isla de los programadores

Envíos	Válidos	% éxito
3	2	66 %

C. La isla de los programadores

- Partiendo de un número de participantes (donde una pareja cuenta como un solo individuo), y un alcance que tiene el tentador, el problema trata de maximizar el número de parejas que son tentadas.
- Solución: algoritmo voraz.
- Complejidad esperada $O(N)$.

C. La isla de los programadores

Primera aproximación: fuerza bruta.

- Probar todas las combinaciones factibles.
- Devolver la mejor de ellas.
- Complejidad: exponencial.

C. La isla de los programadores

Segunda aproximación: Algoritmo voraz.

- Primer criterio: comenzar por la izquierda y asociar a cada tentador la pareja más cercana dentro del alcance.

Figura: k=2

C. La isla de los programadores

Segunda aproximación: Algoritmo voraz.

- Segundo criterio: comenzar por la izquierda y asociar a cada tentador la pareja más lejana dentro del alcance.

Figura: k=3

C. La isla de los programadores

Segunda aproximación: Algoritmo voraz.

- Criterio correcto:
  1. Buscar primer tentador y primera pareja desde la izquierda.
  2. Si la pareja está en el alcance del tentador, se le asigna a este.
  3. Si no, movemos el que esté más a la izquierda al siguiente elemento del mismo tipo.
  4. Repetir los dos pasos anteriores hasta terminar de recorrer el array.

D. La peor entrevista

Envíos	Válidos	% éxito
-	-	- %

D. La peor entrevista

En este programa programa se pedía, como idea general:

1. Leer un número determinado de líneas que contienen texto.
2. Leer una línea que indica el orden en el que las líneas anteriores se mostrarán como salida del programa.
3. Imprimir las líneas en el orden correcto.

D. La peor entrevista

Para resolver de manera correcta este programa se propone:

1. Leer línea a línea la entrada, almacenándola en una estructura de datos similar a un *array* o vector.
2. Una vez leídas todas las líneas, se lee número a número la última línea de la entrada.
3. El número leído, que es una referencia directa al orden en el que se leyeron las líneas en la entrada, permite acceder en $\mathcal{O}(1)$ a la estructura de datos y recuperar la línea deseada.
4. La línea deseada se imprime como salida del programa.

Teniendo esto en cuenta, y dado que en el caso peor será necesario realizar N accesos a la estructura de datos, la complejidad esperada es $\mathcal{O}(N)$

E. Números de Lychrel

Envíos	Válidos	% éxito
-	-	- %

E. Números de Lychrel

Existe un maravilloso truco para que un número se pueda convertir en capicúa: Sumarlo consigo mismo del revés. En el enunciado ya nos daban un ejemplo: 18 + 81 es 99, con lo que conseguimos un capicúa.
Sin embargo, es necesario hacer este truco varias veces hasta llegar. En este problema teníamos que detectar cuántas veces hacía falta repetir el truco, eso sí, teniendo en cuenta que a veces es imposible, lo que da lugar a un Número de Lychrel.
Álex nos da la pista de los números 196, 879, 1997 y 7059. Estos son llamados "Semillas de Lychrel" Un nombre un tanto... Importante.

E. Números de Lychrel

Para resolver este problema, podíamos optar por dos opciones distintas:

1. Realizar para cada número todas las iteraciones hasta alcanzar un capicúa o superar la iteración 25. (Lo que todos habéis hecho y yo no quería que hicierais :))
2. Precomputar todos los números de Lychrel pensando en las propiedades de las semillas de Lychrel. (Lo que nadie ha hecho y mi solución inicial)

E. Números de Lychrel

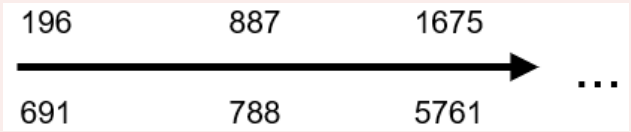
Optando por la primera opción, esta es, saber que todos los números que superen 25 iteraciones serán Lychrel, lo único que necesitamos es darle la vuelta a un número. Esto se puede hacer de dos formas:

1. Pasar el número a String, invertir la String y convertir la String inversa a número.
2. Descomponer el número en dígitos a través de operaciones división y módulo.

¡Cuidado! Los números crecen en dígitos muy rápido. No usar long (En Java) o long long int (En C++) nos dará overflow.

E. Números de Lychrel

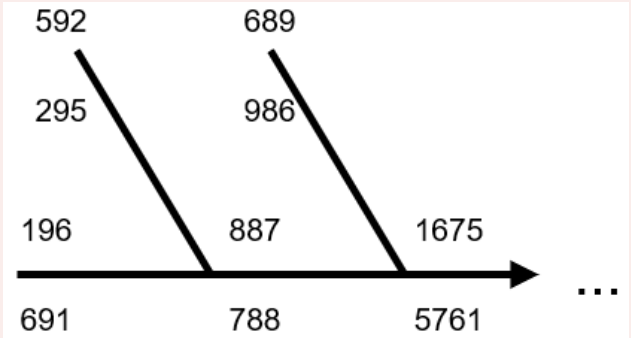
Optando por la seguna opción, podemos seguir el siguiente razonamiento para precomputar los números de Lychrel: Hay infinitos Números de Lychrel, y lo complicado de este problema era encontrarlos. Pensemos un poco: Si 196 es Lychrel, sabemos que nunca podrá hacer un capicúa, por mucho que lo sumemos.
El primer paso sería sumar 196 + 691... ¡Pero si ya sabemos que 196 no va a dar capicúas! 691 debe ser Lychrel también. ¿Y su suma? ¡También tendrá que serlo! 196 + 691 = 887. ¡Y ya tenemos nuestro tercer Lychrel!



Esto produce lo que se llama una "Cadena de Lychrel": Una cadena de números, los cuales no producirán nunca un capicúa. Un primer acercamiento puede ser preprocesar estos números, y si nos dan alguno de ellos, sacar -1 directamente.

E. Números de Lychrel

Pero esto no es suficiente: ¿Qué pasa si nos dan el 295? En la lista anterior no estaba, pero comenzando el proceso... ¡Obtenemos 887!.
Hay que tener cuidado, porque estas Cadenas de Lychrel pueden tener más de una rama:



Para resolver correctamente el problema necesitamos preprocesar las cadenas completas de las semillas de Lychrel (196, 879, 1997 y 7059), al menos hasta 6 iteraciones (Por las ramas laterales). Si nos piden un número de esa lista, lo podemos denotar como “Lychrel” inmediatamente.
En caso contrario, comenzamos el proceso de búsqueda del capicúa. Si durante el proceso nos encontramos con alguno de los números preprocesados, es que habremos entrado a una Cadena de Lychrel, por lo que podemos parar también.

F. Arreglando el lag

Envíos	Válidos	% éxito
-	-	- %

F. Arreglando el lag

Tenemos un grafo en el que cada nodo es un ordenador y cada arista la latencia entre los nodos que conecta. Objetivo: Contar cuántas rutas entre cualquier par de nodos superan la latencia máxima.

F. Arreglando el lag

Complejidad máxima esperada: $O(n^3)$ (por el tamaño de la entrada) Idea: 1. Calcular distancias entre cada par de nodos usando: - N Dijkstras - Floyd-Warshall 2. Crear un segundo grafo, mismos nodos, arista entre dos nodos si la ruta más corta entre 3. Solución: Aristas máximas - Aristas del grafo Donde aristas máximas: $N * (N-1)/2$

G. Plan de movilidad

Envíos	Válidos	% éxito
-	-	- %

G. Plan de movilidad

Tenemos una situación en la que dos personas deben encontrar el camino más corto de forma que se encuentren en un punto.
PROBLEMA: El mapa no es igual para los dos.

G. Plan de movilidad

Existe una red de caminos para motos y otra para coches. Como cada uno de nuestros personajes lleva un tipo de vehículo, debemos encontrar dos caminos válidos que minimicen el coste de encontrarse en un punto.

G. Plan de movilidad

Idea principal 1: mapear las letras a números para armar la matriz es el primer paso (puede hacerse directamente restando ord(character) - ord('A'). Idea principal 2: Leer las aristas para cada vehículo, si son bidireccionales tenerlo en cuenta. Idea principal 3: Aplicar el algoritmo de Floyd sobre las dos redes de caminos Idea principal 4: Aprovechando el DP que nos dejan los dos recorridos, podemos sacar de forma simple el camino mínimo para ambos y dónde se encuentran si es posible el encuentro, además de saber cuándo un encuentro será imposible.

G. Plan de movilidad

Otra opción: Dijkstra desde el origen de Thous y Ada para encontrar los caminos más cortos hacia todos los nodos + encontrar los nodos comunes de longitud mínima.