Ingeniería Social

Nombre del problema	Ingeniería Social
Archivo de entrada	Tarea interactiva
Archivo de salida	Tarea interactiva
Tiempo límite	5 segundos
Límite de memoria	256 megabytes

Una red social consiste de un grafo conexo no dirigido con n vértices y m aristas, donde cada vértice es una persona y dos personas son amigos si hay una arista entre ellos.

María es parte de esta red social y le gusta desafiar a sus amigos. Ella propone un desafío a uno de sus amigos, esta persona desafía a otro de sus amigos y así sucesivamente.

Una persona puede ser desafiada nuevamente, pero nunca se repite el desafío en un mismo par de personas (si A desafía a B, ni A puede desafiar a B ni B puede desafiar a A)

En otras palabras, *los desafíos forman un camino que no repite aristas*.

Si una persona no puede desafiar a nadie en su turno pierde el juego.

El juego siempre lo comienza María e interesantemente ella no ha perdido hasta ahora. Las demás $n-1$ personas decidieron ponerse de acuerdo para que Maria pierda el siguiente juego y es tu tarea implementar su estrategia.

Implementación

Debes implementar la función: `void SocialEngineering(int n, int m, vector<pair<int,int>> edges);`

que describe el próximo juego de n personas, m amistades y *edges* lista las amistades.

Si María tiene una estrategia ganadora, la función debe retornar sin haber realizado llamados a la función `GetMove` y tendrás `Accepted`.

Pero si las demás $n-1$ personas tienen una estrategia ganadora, debe describir los desafíos según esta estrategia.

Las amistades son pares de enteros (u, v) entre *1* y *n*

María es representada por el *1*

Tu función puede llamar dos métodos `int GetMove();` y `void MakeMove(int v);` que representan el desafío de María y de la persona *v* respectivamente. Recuerda que un desafío es válido si no repite aristas de la red social.

`int GetMove();` puede retornar dos tipos de valores:

- un entero *v*, donde $2 \le v \le n$. Esto significa que María desafía a la persona *v*. El evaluador implementa esta función y siempre retorna un desafío válido.
- *0* representa que Maria se rinde. María se rendirá por ejemplo si no puede desafiar a nadie.

Al ejecutar esta función la persona en turno debe ser María, sino tendrás `Wrong Answer`.

`void MakeMove(int v);` representa que la persona en turno desafía a la persona *v*.

Al ejecutar esta función la persona en turno no debe ser María, sino tendrás `Wrong Answer`. Al ejecutar esta función el desafío debe ser válido, sino tendrás `Wrong Answer`

Finalmente:

- si `GetMove` retorna 0, `SocialEngineering` debe retornar en la siguiente instrucción.
- el evaluador llama `SocialEngineering` una sola vez.

Restricciones

- $2 \le n \le 2 \cdot 10^5$.
- $1 \le m \le 4 \cdot 10^5$.
- El grafo es conexo, cada par de vértices aparece máximo una vez como arista y cada arista une dos vértices diferentes.

Subtareas

María siempre juega perfecto en el sentido que siempre realizará movimientos ganadores cuando tiene una estrategia ganadora.

Si ella no tiene una estrategia ganadora, entonces ella realizará movimientos inteligentes que busquen la victoria ante posibles errores de los demás jugadores, salvo en la subtarea 3.

1. (15 puntos) $n, m \leq 10$.
2. (15 puntos) Todos salvo María tienen máximo dos amigos.
3. (20 puntos) María se rendirá inmediatamente a menos que tenga una estrategia ganadora.
4. (25 puntos) $n, m \leq 100$.
5. (25 puntos) restricciones iniciales (ninguna extra).

Interacción de ejemplo

Acción del participante	Acción del evaluador	Explicación
-	`SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering()` es llamada con un grafo de 5 vértices y 6 aristas.
`GetMove()`	`Retorna 4`	Maria desafía a la persona 4.
`MakeMove(2)`	-	La persona 4 desafía a la persona 2.
`MakeMove(5)`	-	La persona 2 desafía a la persona 5.
`MakeMove(1)`	-	La persona 5 desafía a Maria.
`GetMove()`	`Retorna 0`	Maria no tiene movimientos válidos, se rinde.
`return`	-	El juego terminó, y `SocialEngineering()` debe retornar.

Acción del participante	Acción del evaluador	Explicación
-	`SocialEngineering(2, 1, {{1,2}})`	`SocialEngineering` es llamada con un grafo de 2 vertices y 1 arista.
`return`	-	María tiene una estrategia ganadora, la función debe retornar inmediatamente.

Evaluador de ejemplo

El evaluador de ejemplo `grader.cpp`, en el archivo `SocialEngineering.zip`, lee del input estandar en el siguiente formato:

- La cantidad de vértices, *n*, y la cantidad de aristas, *m* en el grafo.
- *m* aristas como pares de enteros entre *1* y *n*.

El evaluador de ejemplo lee el input y luego llama `SocialEngineering`. Notar que el evaluador de ejemplo no implementa la estrategia ganadora de María y provee solo un ejemplo de estrategia.

Para compilar los programas puedes usar el siguiente comando en una terminal:
`g++ -std=gnu++11 -O2 -o solution grader.cpp solution.cpp`
donde `solution.cpp` es tu solución a enviar al sistema (CMS). Para ejecutar solución con la entrada de ejemplo, puedes usar el siguiente comando en una terminal: `./solution < input.txt`