Ingeniería social

Problem Name	Social Engineering
Input file	Interactive task
Output file	Interactive task
Time limit	5 seconds
Memory limit	256 megabytes

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

María es una miembro de esta red social. A ella le gusta retar a sus amigos a hacer varias cosas. Esto significa que primero ella realiza una tarea simple, y después reta a uno de sus amigos a que haga la misma. Este amigo retará entonces a uno de sus amigos, el cual retará a otro de sus amigos, y así sucesivamente. Podría pasar que la misma persona sea retada más de una vez; sin embargo, cada pareja (no ordenada) de amigos puede solo pertenecer a lo más a un reto. (Una vez que una persona A haya retado a una persona B, ni la persona A ni la persona B pueden retar al otro otra vez). En otras palabras, los retos formarán un camino en el grafo que nunca usa una arista más de una vez.

Una persona pierde el reto si es su turno y ya no puede retar a ninguno de sus amigos. Los retos siempre comienzan con María y ella pierde muy rara vez. Ahora mismo, las $n-1$ personas restantes han decidido colaborar para hacer que María pierda del siguiente reto, y es tu trabajo coordinar esta causa.

Implementación

Debes implementar la función:

`void SocialEngineering(int n, int m, vector<pair<int,int>> edges);` que juega el juego sobre un grafo con n vértices y m aristas. Esta función será llamada una vez por el evaluador. La lista `edges` contendrá exactamente m parejas de enteros (u, v), y significa que una arista conecta los vértices u y v. Los vértices están numerados de 1 a n. María siempre es el vértice 1.

Tu función puede llamar a las siguientes funciones:

`int GetMove();` Esta función debe ser llamada cada que sea el turno de María, así como en el principio del juego. Si se llama a esta función cuando no sea el turno de María, obtendrás `Wrong Answer` como veredicto. La función puede regresar alguno de los siguientes valores:

- un entero v, donde $2 \leq v \leq n$. Esto significa que María reta a la persona con índice v. Este siempre será un movimiento válido.
- 0, si María se rinde. María siempre va a rendirse si ya no tiene movimientos válidos. Cuando esto pase, tu programa debe salir de la función `SocialEngineering` y obtendrás `Accepted` como veredicto.

`void MakeMove(int v);`. Esta función debe ser llamado cada que no sea el turno de María. Esto significa que la persona en turno reta a la persona v. Si no es un movimiento válido o si es el turno de María en el momento de la llamada, entonces obtendrás `Wrong Answer` como veredicto.

Si María tiene una estrategia ganadora al principio del juego, tu programa debe salir de la función `SocialEngineering` *antes* de la primera llamada a `GetMove()`. Así obtendrás `Accepted` como veredicto.

Límites

- $2 \leq n \leq 2 \cdot 10^5$.
- $1 \leq m \leq 4 \cdot 10^5$.
- El grafo está conectado. Cada pareja no ordenada de vértices aparecerá a lo más una vez como arista, y cada arista conectará dos vértices distintos.

Subtareas

María siempre jugará perfecto en el sentido de que hará movimientos ganadores siempre que tenga estrategia ganadora. Si ella no tiene una estrategia ganadora, entonces intentará provocar que tu programa cometa un error, de varias maneras ingeniosas. Ella solo se va a rendir si no tiene movimientos válidos, a excepción de la subtarea 3.

1. (15 puntos) $n, m \leq 10$.
2. (15 puntos) Todos excepto María tienen a lo más 2 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) No hay ninguna consideración adicional.

Interacción de ejemplo

Acción del participante	Acción del evaluador	Explicación
-	Se llamó a `SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering` con un grafo con 5 vértices y 6 aristas.
`GetMove()`	`Regresa 4`	María reta a la persona 4.
`MakeMove(2)`	-	La persona 4 reta a la persona 2.
`MakeMove(5)`	-	La persona 2 reta a la persona 5.
`MakeMove(1)`	-	La persona 5 reta a María.
`GetMove()`	`Regresa 0`	María no tiene movimientos válidos, por lo que se rinde.
`Returns`	-	El participante ganó el juego y debe salir de la función `SocialEngineering`.

Acción del participante	Acción del evaluador	Explicación
-	`SocialEngineering(2, 1, {{1,2}})`	Se llamó a `SocialEngineering` con un grafo con 2 vértices y 1 arista.
`Returns`	-	María tiene una estrategia ganadora en este grafo, por lo que el participante debe salir de la función sin hacer ninguna llamada a `GetMove()` para rendirse.

Evaluador de ejemplo

El evaluador de ejemplo provisto, `grader.cpp`, en el archivo adjunto `SocialEngineering.zip` lee la entrada de la entrada estándar en el siguiente formato:

- La primera línea contiene el núnmero de vértices *n* y el número de aristas *m* del grafo.
- Las siguientes *m* líneas contienen dos enteros *u* y *v*, indicando que hay una arista entre *u* y *v*.

El evaluador de ejemplo lee de la entrada y llama a la función `SocialEngineering` de la solución del participante. Nota que el evaluador de ejemplo no implementa la estrategia ganadora de María y solo se proporciona para una interacción de ejemplo.

Para compilar el evaluador de ejemplo con tu solución, puedes usar el siguiente comando en la terminal:

```
g++ -std=gnu++11 -O2 -o solution grader.cpp solution.cpp
```

donde `solution.cpp` es el archivo de tu solución que subirás a CMS. Para correr el programa con la entrada de ejemplo proporcionada en el archivo adjunto, escribe el siguiente comando desde la terminal:

```
./solution < input.txt
```