SocialEngineering

Nombre del problema	Social Engineering
Fichero de entrada	Problema interactivo
Fichero de salida	Problema interactivo
Límite de tiempo	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 amigas si hay una arista que las conecta. Maria es un usuario de esta red social. Le gusta retar a sus amigos a hacer cosas varias. Esto consiste en que ella primero realiza una tarea sencilla y luego reta a uno de sus amigos a hacer lo mismo. Este reto viajará entonces a través de la red. Podría pasar que la misma persona fuera retada más de una vez, pero cada pareja desordenada de amigos puede participar en el reto como mucho una vez. (Una vez que una persona A rete a una persona B, ya no podrán ni A ni B retarse mutuamente de nuevo.) En otras palabras, el reto será un camino en el grafo que no use ninguna arista más de una vez. Una persona pierde el reto si es su turno y no puede retar a ninguno de sus amigos. Los retos siempre los empieza Maria y, curiosamente, no ha perdido ningún reto hasta la fecha. Ahora las n - 1 personas restantes han decidido colaborar para hacer que Maria pierda el próximo reto, y será tu trabajo coordinar este esfuerzo.

Implementación

Debes implementar una función:

```
void SocialEngineering(int n, int m, vector<pair<int,int>> edges);
```

que juegue al juego en un grafo con n vértices y m aristas. Esta función será llamada una vez por el corrector. La lista `edges` contendrá exactamente m parejas de enteros (u, v), que significan que una arista va del vértice u al vértice v. Los vértices tienen números del 1 al n. Maria es siempre el vértice 1. Tu función puede realizar llamadas a los siguientes métodos:

```
int GetMove();
```

Este método debería ser llamado siempre que sea el turno de Maria como, por ejemplo, al principio del juego. Si llamas a este método sin que sea el turno de Maria, recibirás el veredicto `Wrong Answer`. Este método devolverá uno de los siguientes valores:

- un entero v, donde $2 \le v \le n$. Esto significa que Maria reta a la persona con índice v. Este siempre será un movimiento legal.
- 0, si Maria se rinde en el juego. Maria siempre se rendirá cuando no tenga ningún movimiento legal. Cuando esto ocurra, tu programa debería dejar a la función `SocialEngineering()` retornar, y tú deberías obtener el veredicto `Accepted`.

```
void MakeMove(int v);
```

Este método debería ser llamado cuando no sea el turno de Maria. Esto significa que la persona de la que sea el turno reta a la persona v. Si este no es un movimiento legal o si es el turno de Maria cuando se hace la llamada, recibirás el veredicto `Wrong Answer`.
Si Maria tiene una estrategia ganadora al principio del juego, tu programa debería dejar a `SocialEngineering()` retornar *antes* de la primera llamada a `GetMove()`. Obtendrás entonces el veredicto `Accepted`.

Restricciones

- $2 \le n \le 2 \cdot 10^5$.
- $1 \le m \le 4 \cdot 10^5$.
- El grafo será conexo. Cada pareja desordenada de vértices aparecerá como mucho una vez como arista, y cada arista conectará dos vértices diferentes.

Subtareas

Maria siempre jugará de forma perfecta en el sentido de que hará jugadas que la lleven a la victoria siempre que exista una estrategia ganadora. Si no tiene ninguna estrategia ganadora, tratará de trucar a tu programa para que cometa un error, de varias formas inteligentes. Solo se rendirá si no tiene movimientos legales, excepto en la Subtarea 3.

1. (15 puntos) $n, m \le 10$.
2. (15 puntos) Todos excepto Maria tendrán como mucho dos amigos.
3. (20 puntos) Maria se rendirá de inmediato salvo que tenga una estrategia ganadora.
4. (25 puntos) $n, m \le 100$.
5. (25 puntos) Restricciones originales.

Ejemplo de interacción

Tu acción	Acción del corrector	Explicación
-	`SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering` es llamada con un grafo con 5 vértices y 6 aristas.
`GetMove()`	`Devuelve 4`	Maria 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 Maria.
`GetMove()`	`Devuelve 0`	Maria no tiene movimientos legales, así que se rinde.
`Retorna`	-	Has ganado la partida y deberías dejar a `SocialEngineering` retornar.

Acción del concursante	Acción del corrector	Explicación
-	`SocialEngineering(2, 1, {{1,2}})`	`SocialEngineering` es llamada con un grafo con 2 vértices y 1 arista.
`Retorna`	-	Maria tiene una estrategia ganadora en este grafo, así que el concursante debería retornar inmediatamente para rendirse.

Corrector de ejemplo

El corrector de ejemplo proporcionado, `grader.cpp`, en el adjunto de la tarea `SocialEngineering.zip`, lee la entrada de la entrada estándar en el siguiente formato:

- La primera línea contiene el número de nodos, *n*, y el número de aristas, *m* en el grafo.
- Las siguientes *m* líneas contienen aristas como parejas de nodos.

El corrector de ejemplo lee la entrada y llama a la función `SocialEngineering` en la solución del concursante. Ten en cuenta que el corrector de ejemplo no implementa la estrategia ganadora de Maria y se proporciona simplemente como un ejemplo de interacción. Para compilar el correcto 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` sea tu archivo a enviar a CMS. Para ejecutar tu programa con la entrada de

ejemplo proporcionada en el adjunto, escribe el siguiente comando en la terminal:

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