SocialEngineering

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

A social network consists of an undirected connected graph with n vertices and m edges, where each vertex is a person, and two people are friends, if there is an edge between them.
Maria is a member of this social network. She likes challenging her friends to do various things. This means that she first performs some simple task, and then challenges one of her friends to do the same. This challenge will then travel around the network. It could happen that the same person gets challenged more than once, but each unordered pair of friends can only take part in the challenge at most once. (Once a person A challenges a person B, then neither person A nor person B can challenge the other again.) In other words, the challenge will be a walk in the graph that never uses an edge more than once.
A person loses the challenge if it is their turn and they cannot challenge any of their friends. The challenges are always started by Maria, and interestingly, she has not lost any challenges so far. Now the remaining $n - 1$ people have decided to collaborate in order to make Maria lose the next challenge, and it is your job to coordinate this effort.

Implementation

You must implement a function:

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

that plays the game on a graph with n vertices and m edges. This function will be called once by the grader. The list `edges` will contain exactly m pairs of integers (u, v), meaning that an edge goes between vertex u and vertex v. Vertices are numbered from 1 to n. Maria is always vertex 1. Your function can make calls to the following methods:

```
int GetMove();
```

This method should be called whenever it is Maria's turn, such as in the beginning of the game. If

you call this method when it is not Maria's turn, you will get the verdict `Wrong Answer`. The method can return one of the following values:

- an integer v, where $2 \leq v \leq n$. This means that Maria challenges the person with index v. This will always be a legal move.
- 0, if Maria resigns the game. Maria will always resign, if she has no legal moves. When this happens, your program should let the function `SocialEngineering()` return, and you will get the verdict `Accepted`.

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

This method should be called whenever it is not Maria's turn. This means that the person with the turn challenges the person v. If this is not a legal move or if it is Maria's turn at the time of the call, then you will get the verdict `Wrong Answer`.

If Maria has a winning strategy at the start of the game, your program should let `SocialEngineering()` return *before* the first call to `GetMove()`. You will then get the verdict `Accepted`.

Constraints

- $2 \leq n \leq 2 \cdot 10^5$.
- $1 \leq m \leq 4 \cdot 10^5$.
- The graph is connected. Every unordered pair of vertices will appear at most once as an edge, and every edge will go between two different vertices.

Subtasks

Maria will always play perfectly in the sense that she will make winning moves whenever she has a winning strategy. If she does not have a winning strategy, then she will try to lure your program into making a mistake, in various clever ways. She will only resign if she has no legal moves, except in Subtask 3.

1. (15 points) $n, m \leq 10$.
2. (15 points) Everyone except Maria has at most 2 friends.
3. (20 points) Maria will resign immediately unless she has a winning strategy.
4. (25 points) $n, m \leq 100$.
5. (25 points) No further constraints.

Sample Interaction

Contestant action	Grader action	Explanation
-	`SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering()` is called with a graph with 5 vertices and 6 edges.
`GetMove()`	`Returns 4`	Maria challenges person 4.
`MakeMove(2)`	-	Person 4 challenges person 2.
`MakeMove(5)`	-	Person 2 challenges person 5.
`MakeMove(1)`	-	Person 2 challenges Maria.
`GetMove()`	`Returns 0`	Maria has no legal moves, so she resigns.
`Returns`	-	The contestant has won the game, and should let `SocialEngineering()` return.

Contestant action	Grader action	Explanation
-	`SocialEngineering(2, 1, {{1,2}})`	`SocialEngineering` is called with a graph with 2 vertices and 1 edge.
`Returns`	-	Maria has a winning strategy on this graph, so the contestant should return immediately to resign.

Sample Grader

The provided sample grader, `grader.cpp`, in the task attachment `SocialEngineering.zip`, reads the input from the standard input in the following format:

- The first line contains the number of nodes, *n*, and the number of edges, *m* in the graph.
- The following *m* lines contain edges as node tuples.

Sample grader reads the input and calls the `SocialEngineering()` function in user's solution. Note that the sample grader does not implement Maria's winning strategy and provided only for a sample interaction.
To compile the sample grader with your solution, you may use the following command at the terminal prompt:

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

where `solution.cpp` is your solution file to be submitted to CMS. To run the program with the sample input provided in the attachment, type the following command from at the terminal prompt:

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