Engenharia Social

Nome do Problema	Engenharia Social
Arquivo de entrada	Tarefa interativa
Aquivo de saída	Tarefa interativa
Limite de tempo	5 segundos
Limite de memória	256 megabytes

Uma rede social consiste em um grafo não-direcionado conexo com *n* vértices e *m* arestas, onde cada vértice é uma pessoa, e duas pessoas são amigas se houver uma aresta entre elas.
Maria é membro desta rede social. Ela gosta de desafiar seus amigos a fazerem várias coisas. Isto significa que ela primeiro realiza uma tarefa simples, e depois desafia um de seus amigos a fazer o mesmo. Este amigo então desafiará um de seus amigos, que desafiará um de seus amigos, e assim por diante. É possível que uma mesma pessoa seja desafiada mais de uma vez, mas cada par não ordenado de amigos só pode participar do desafio no máximo uma vez. (Uma vez que uma pessoa *A* desafia uma pessoa *B*, então nem a pessoa *A* nem a pessoa *B* pode desafiar o outro novamente). Em outras palavras, o desafio será um passeio no grafo que nunca usa uma aresta mais de uma vez.
Uma pessoa perde o desafio se for a sua vez e ela não puder desafiar nenhum de seus amigos. Os desafios são sempre iniciados por Maria, e ela raramente perde. Agora as demais *n* - 1 pessoas decidiram colaborar a fim de fazer Maria perder o próximo desafio, e é seu trabalho coordenar este esforço.

Implementação

Você deve implementar uma função:

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

que simula o jogo em um grafo com *n* vértices e *m* arestas. Esta função será chamada uma vez pelo corretor. A lista `edges` (arestas) contém exatamente *m* pares de inteiros (*u*, *v*), o que significa que há uma aresta entre o vértice *u* e o vértice *v*. Os vértices são numerados de 1 a *n*. Maria é sempre o vértice 1. Sua função pode chamar as seguintes funções:

```
int GetMove();
```

Este método deve ser chamado sempre que for a vez de Maria, por exemplo no início do jogo. Se

você chamar este método quando não é a vez de Maria, você terá o veredito `Wrong Answer`. O método pode retornar um dos seguintes valores:

- um número inteiro v, onde $2 \le v \le n$. Isto significa que Maria desafia a pessoa com índice v. Este será sempre um movimento válido.
- 0, se Maria desiste do jogo. Maria sempre desiste se ela não tem movimentos válidos. Quando isso acontecer, seu programa deve deixar a função `SocialEngineering` retornar, e você terá o veredito `Accepted`.

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

Este método deve ser chamado sempre que não for a vez de Maria. Isto significa que o jogador da vez desafia a pessoa v. Se este não for um movimento válido ou se for a vez de Maria no momento da chamada, você receberá o veredito `Wrong Answer`.
Se Maria tem uma estratégia vencedora no início do jogo, seu programa deve deixar `SocialEngineering` retornar *antes* da primeira chamada a `GetMove()`. Você receberá então o veredito `Accepted`.

Restrições

- $2 \le n \le 2 \cdot 10^5$.
- $1 \le m \le 4 \cdot 10^5$.
- O grafo é conexo. Cada par não-ordenado de vértices aparecerá no máximo uma vez como uma aresta, e cada aresta é entre dois vértices distintos.

Pontuação

Maria jogará sempre de forma perfeita no sentido de que ela fará movimentos vencedores sempre que ela tiver uma estratégia vencedora. Se ela não tiver uma estratégia vencedora, então ela tentará induzir seu programa a fazer um erro, de várias maneiras espertas. Ela só desistirá se não tiver movimentos válidos, exceto na Subtarefa 3.

Subtarefa 1 (15 pontos) $n, m \le 10$.

Subtarefa 2 (15 pontos) Todos exceto Maria têm no máximo 2 amigos.

Subtarefa 3 (20 pontos) Maria desitirá imediatamente, exceto se tiver uma estratégia vencedora.

Subtarefa 4 (25 pontos) $n, m \le 100$.

Subtarefa 5 (25 pontos) Nenhuma restrição adicional.

Exemplo de Interação

Sua ação	Ação do corretor	Explicação
-	`SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering` é chamada em um grafo com 5 vértices e 6 arestas.
`GetMove()`	`Returns 4`	Maria desafia a pessoa 4.
`MakeMove(2)`	-	A pessoa 4 desafia a pessoa 2.
`MakeMove(5)`	-	A pessoa 2 desafia a pessoa 5.
`MakeMove(1)`	-	A pessoa 5 desafia Maria.
`GetMove()`	`Returns 0`	Maria não tem nenhum movimento válido, então ela desiste.
`Returns`	-	Você venceu o jogo e deve deixar `SocialEngineering` retornar.

Sua ação	Ação do corretor	Explicação
-	`SocialEngineering(2, 1, {{1,2}})`	`SocialEngineering` é chamada em um grafo com 2 vértices e 1 aresta.
`Returns`	-	Maria tem uma estratégia vencedora neste grafo, portanto, você deve retornar sem fazer nenhuma chamada de `GetMove()` para desistir.

Corretor Exemplo

O corretor exemplo `grader.cpp`, fornecido no anexo de tarefa `SocialEngineering.zip`, lê a entrada da entrada padrão no seguinte formato:

- A primeira linha contém o número de vértices, *n*, e o número de arestas, *m*, do grafo.
- As próximas *m* linhas contêm dois números inteiros *u* e *v*, indicando que existe uma aresta entre *u* e *v*.

O corretor exemplo lê a entrada e chama a função `SocialEngineering` na solução do usuário. Note que o corretor exemplo não implementa a estratégia vencedora de Maria e fornece apenas um exemplo de interação.

Para compilar o corretor exemplo com sua solução, você pode usar o seguindo o comando no terminal de comando:

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

onde `solution.cpp` é seu arquivo de solução a ser submetido para o CMS. Para executar o programa com o exemplo de entrada fornecido no anexo, digite o seguinte comando no terminal de comando:

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