Социјално Инженерство

Име на задачата	Social Engineering
Влезна датотека	Интерактивна задача
Излезна датотека	Интерактивна задача
Временско ограничување	5 секунди
Мемориско ограничување	256 мегабајти

Една социјална мрежа се состои од неориентиран сврзан граф со n темиња и m ребра, кадешто секое теме е една личност, и притоа две личности се пријатели ако постои ребро помеѓу нив.

Атина е член на оваа социјална мрежа. Таа обожува да ги предизвикува своите пријатели да прават разни нешта. Ова значи дека таа прво извршува некоја едноставна активност, и потоа предизвикува еден од своите пријатели да го направи истото. Овој пријател потоа ќе предизвика еден од своите пријатели, кој пак ќе предизвика еден од своите пријатели, и така натаму. Може да се случи истата личност да биде предизвикана повеќе од еднаш, но секој неподреден пар од пријатели може да учествува во предизвикот најмногу еднаш (Штом личноста A ќе ја предизвика еднаш личноста B, од тој момент ниту личноста A ниту личноста B не може веќе да ја предизвика другата личност). Со други зборови, предизвиците ќе претставуваат пат во графот кој не изминува ниту едно ребро повеќе од еднаш.

Една личност е губитник во предизвикот ако дошол ред на неа и притоа не може да предизвика ниту еден од своите пријатели. Предизвиците секогаш почнуваат од Атина и таа многу ретко е губитник . Сега преостанатите $n-1$ луѓе одлучиле да соработуваат за да направат Атина да биде губитник во следниот предизвик, а ваша задача е да го координирате овој обид.

Имплементација

Треба да имплементирате функции:

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

којашто ја игра играта на граф со n темиња и m ребра. Оваа функција ќе биде повикана еднаш од оценувачот. Листата `edges` ќе содржи точно m парови од цели броеви (u, v), со значење дека постои ребро кое го поврзува темето u со темето v. Темињата се нумерирани

со целите броеви од 1 до n. Атина е секогаш темето 1. Вашата функција може да прави повици до следните функции:

```
int GetMove();
```

Оваа функција треба да биде повикана секогаш кога Атина е на ред, како на пример - на почетокот на играта. Ако ја повикате оваа функција кога Атина не е на ред, ќе добиете резултат: `Wrong Answer`. Функцијата може да врати една од следните вредности:

- цел број v, каде $2 \le v \le n$. Ова означува дека Атина ја предизвикува личноста со реден број v. Ова секогаш ќе биде легален потег.
- 0, ако Атина се откажува од играта. Атина ќе се откаже секогаш кога ќе нема легални потези. Кога ќе се случи ова, вашата програма треба да овозможи функцијата `SocialEngineering` да заврши (return), и ќе добиете резултат `Accepted`.

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

Оваа функција треба да биде повикана секогаш кога не е Атина на ред. Ова означува дека личноста којашто е на ред ја предизвикува личноста v. Ако ова не е легален потег или ако во моментот на повикот на ред е Атина, тогаш ќе добиете резултат `Wrong Answer`.

Ако Атина има победничка стратегија на почетокот на играта, вашата програма треба да овозможи функцијата `SocialEngineering` да заврши (return) *пред* првиот повик до `GetMove()`. Во овој случај ќе добиете резултат `Accepted`.

Ограничувања

- $2 \le n \le 2 \cdot 10^5$.
- $1 \le m \le 4 \cdot 10^5$.
- Графот е сврзан. Секој неподреден пар од темиња ќе се појавува најмногу еднаш како ребро, и секое ребро ќе поврзува две различни темиња.

Подзадачи

Атина секогаш ќе игра перфектно во смисла на тоа дека таа ќе презема победнички потези секогаш кога ќе има победничка стратегија. Ако таа нема победничка стратегија, тогаш таа ќе се обиде да ја намами вашата програма да направи грешка, на различни паметни начини. Таа ќе се откаже само ако нема легални потези, освен во Подзадачата 3.

1. (15 поени) $n, m \le 10$.
2. (15 поени) Сите, освен Атина, имаат најмногу 2 пријатели.

3. (20 поени) Атина ќе се откаже веднаш освен во случај ако има победничка стратегија.

4. (25 поени) $n, m \leq 100$.

5. (25 поени) Нема дополнителни ограничувања.

Пример - Интеракција

Ваша акција	Акција од оценувачот	Објаснување
-	`SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering` се повикува со граф со 5 темиња и 6 ребра.
`GetMove()`	`Враќа 4`	Атина ја предизвикува личноста 4.
`MakeMove(2)`	-	Личноста 4 ја предизвикува личноста 2.
`MakeMove(5)`	-	Личноста 2 ја предизвикува личноста 5.
`MakeMove(1)`	-	Личноста 5 ја предизвикува Атина.
`GetMove()`	`Враќа 0`	Атина нема легални потези, па се откажува.
`Return`	-	Вие сте победник на играта, па треба да овозможите `SocialEngineering` да заврши (return).

Ваша акција	Акција од оценувачот	Објаснување
-	`SocialEngineering(2, 1, {{1,2}})`	`SocialEngineering` се повикува со граф со 2 темиња и 1 ребро.
`Returns`	-	Атина има победничка стратегија на овој граф, па вие треба да направите return без да правите повици до `GetMove()`, за да се откажете.

Пример - Оценувач

Дадениот пример-оценувач, `grader.cpp`, во додатокот на задачата `SocialEngineering.zip`, ги чита влезните податоци од стандардниот влез во следниот формат:

- Првата линија го содржи бројот на темиња, n, и бројот на ребра, m, во графот.

- Следните m линии содржат по два цели броја u и v, кои означуваат дека постои ребро помеѓу темето u и темето v.

Пример-оценувачот ги чита влезните податоци и ја повикува функцијата `SocialEngineering` во решението на корисникот. Да забележиме дека пример-оценувачот не ја имплементира победничката стратегија на Атина, туку ви е даден само за да може да ја испробате интеракцијата.

За да го компајлирате пример-оценувачот со вашето решение, може да ја искористите следната наредба во командната линија (terminal prompt):

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

каде што `solution.cpp` е датотека со вашето решение која ќе ја прикачувате на CMS. За да ја извршите програмата со пример-влезот даден во додатокот, напишете ја следната команда во командната линија (terminal prompt):

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