Social Engineering

Aufgabenname	Social Engineering
Eingabedatei	Interaktive Aufgabe
Ausgabedatei	Interaktive Aufgabe
Zeitlimit	5 Sekunden
Speicherlimit	256 Megabytes

Ein soziales Netzwerk besteht aus einem ungerichteten, zusammenhängenden Graphen mit n Knoten und m Kanten, wobei jeder Knoten eine Person ist und zwei Personen befreundet sind, wenn sie durch eine Kante verbunden sind. Maria ist Mitglied in diesem sozialen Netzwerk. Sie fordert ihre Freunde gerne zu verschiedenen Dingen heraus. Das bedeutet, dass sie zuerst eine einfache Aufgabe durchführt und dann einen ihrer Freunde auffordert, dasselbe zu tun. Dieser Freund wird dann einen seiner Freunde herausfordern, der wiederum einen seiner Freunde herausfordert, und so weiter. Es kann vorkommen, dass die gleiche Person mehr als einmal herausgefordert wird. Jedes Freundespaar kann jedoch höchstens einmal an der Herausforderung teilnehmen (sobald eine Person A eine Person B herausfordert, können weder Person A noch Person B die jeweils andere Person erneut herausfordern). Mit anderen Worten, die Herausforderungen bilden einen Weg im Graphen, wobei eine Kante nie mehr als einmal benutzt werden darf.
Eine Person verliert die Herausforderung, wenn sie an der Reihe ist und keinen ihrer Freunde mehr herausfordern kann. Die Herausforderungen gehen immer von Maria aus und bisher verliert sie selten. Jetzt haben die übrigen n - 1 Personen beschlossen zusammenzuarbeiten, um Maria dazu zu bringen, die nächste Herausforderung zu verlieren. Es ist deine Aufgabe, diese Bemühungen zu koordinieren.

Implementierung

Du musst folgende Funktion implementieren:

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

Diese Funktion spielt das Spiel auf einem Graphen mit n Knoten und m Kanten. Diese Funktion wird einmal durch den Grader aufgerufen. Die Liste `edges` enthält genau m Paare von Ganzzahlen (u, v), was bedeutet, dass eine Kante die Knoten u und v verbindet. Die Knoten sind von 1 bis n

nummeriert. Maria ist immer Knoten 1. Deine Funktion kann folgende Methoden aufrufen:

```
int GetMove();
```

Diese Methode sollte immer dann aufgerufen werden, wenn Maria am Zug ist, z.B. bei Beginn des Spiels. Wenn du diese Methode aufrufst, wenn Maria nicht an der Reihe ist, erhälst du das Feedback `Wrong Answer`. Die Methode kann einen der folgenden Werte zurückgeben:

- eine Ganzzahl v, wobei $2 \leq v \leq n$. Das bedeutet, dass Maria die Person mit dem Index v herausfordert. Dies ist immer ein legaler Zug.
- Die Zahl 0, wenn Maria das Spiel aufgibt. Maria wird immer aufgeben, wenn es für sie keine legalen Züge mehr gibt. Wenn dies geschieht, sollte die Funktion `SocialEngineering` returnen, und du erhältst das Feedback `Accepted`.

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

Diese Methode sollte immer dann aufgerufen werden, wenn Maria nicht an der Reihe ist. Das bedeutet, dass die Person, die am Zug ist, die Person v herausfordert. Wenn dies kein legaler Zug ist oder Maria zum Zeitpunkt des Aufrufs am Zug ist, dann erhältst du das Feedback `Wrong Answer`.

Wenn Maria zu Beginn des Spiels eine Gewinnstrategie hat, sollte `SocialEngineering` returnen, *bevor* `GetMove()` zum ersten Mal aufgerufen wird. Du wirst dann das Ergebnis `Accepted` erhalten.

Beschränkungen

- $2 \leq n \leq 2 \cdot 10^5$.
- $1 \leq m \leq 4 \cdot 10^5$.
- Der Graph ist zusammenhängend. Jedes ungeordnete Paar von Knoten erscheint höchstens einmal als Kante, und jede Kante verläuft zwischen zwei verschiedenen Knoten.

Teilaufgaben

Maria wird immer optimal spielen, d. h. sie wird immer dann einen optimalen Zug machen, wenn sie eine Gewinnstrategie hat. Wenn sie keine Gewinnstrategie hat, wird sie versuchen, dein Programm auf verschiedene Weisen zu einem Fehler zu bewegen. Sie wird nur dann aufgeben, wenn sie keine legale Züge mehr ausführen kann, ausser in Teilaufgabe 3.

1. (15 Punkte): $n, m \leq 10$.
2. (15 Punkte): Jede Person ausser Maria hat maximal 2 Freunde.

3. (20 Punkte): Maria wird sofort aufgeben, es sei denn, sie hat eine Gewinnstrategie.

4. (25 Punkte): $n, m \leq 100$.

5. (25 Punkte): Keine weiteren Einschränkungen.

Beispielinteraktion

Deine Aktion	Aktion des Graders	Erklärung
-	`SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering` wird mit einem Graph mit 5 Knoten and 6 Kanten aufgerufen.
`GetMove()`	`Returns 4`	Maria fordert Person 4 heraus.
`MakeMove(2)`	-	Person 4 fordert Person 2 heraus.
`MakeMove(5)`	-	Person 2 fordert Person 5 heraus.
`MakeMove(1)`	-	Person 5 fordert Maria heraus.
`GetMove()`	`Returns 0`	Maria kann keinen legalen Zug mehr ausführen, also gibt sie auf.
`Returns`	-	Du hast das Spiel gewonnen und solltest `SocialEngineering` returnen lassen.

Deine Aktion	Aktion des Graders	Erklärung
-	`SocialEngineering(2, 1, {{1,2}})`	`SocialEngineering` wird mit einem Graphen mit 2 Knoten und 1 Kante aufgerufen.
`Returns`	-	Maria hat bei diesem Graphen eine Gewinnstrategie, deshalb solltest du ohne einen Aufruf der Funktion `GetMove()` aufgeben.

Beispielgrader

Der mitgelieferte Beispielgrader, `grader.cpp`, im Aufgabenanhang `SocialEngineering.zip`, liest die Standardeingabe im folgenden Format:

- Die erste Zeile enthält die Anzahl der Knoten n und die Anzahl der Kanten m im Graphen.

- Die folgenden *m* Zeilen enthalten zwei Ganzzahlen *u* und *v*, die angeben, dass es eine Kante zwischen *u* und *v* gibt.

Der Beispielgrader liest die Eingabe und ruft die Funktion `SocialEngineering` in der Lösung der Teilnehmerin auf. Beachte, dass der Beispielgrader nicht Marias Gewinnstrategie implementiert und nur als Beispiel für eine Interaktion zur Verfügung steht.
Um den Beispielgrader mit deiner Lösung zu kompilieren, kannst du den folgenden Befehl an der Terminal-Eingabeaufforderung verwenden:

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

Dabei ist `solution.cpp` deine Lösungsdatei, die an das CMS übermittelt werden soll. Um das Programm mit der Beispieleingabe im Anhang auszuführen, gib den folgenden Befehl an der Terminal-Eingabeaufforderung ein:

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