Zadanie: SON
Sonda [A]

POTYCZKI ALGORYTMICZNE

Potyczki Algorytmiczne 2019, runda piąta. Limity: 256 MB, 2 s. *13.12.2019*

Bajtocka Agencja Kosmiczna kilka lat temu odkryła planetoidę BA-1T. Bajtazar nadzoruje misję bezzałogowej sondy badawczej, która ma za zadanie sprowadzić na Ziemię próbki znalezionych tam minerałów. Niestety, niekompetentni programiści wprowadzili do systemu sondy wiele błędów, przez co komunikacja z nią stała się bardzo utrudniona. Pomimo tego, Bajtazar musi wykonać misję.

Bajtazar wyznaczył na planetoidzie n istotnych punktów, z których należy pobrać próbki minerałów. Punkty ponumerowane są liczbami naturalnymi od 1 do n, przy czym sonda znajduje się początkowo w punkcie numer 1. Istnieje pewna liczba dwukierunkowych połączeń pomiędzy tymi punktami. Jeśli sonda znajduje się na jednym końcu połączenia, to może wykonać ruch na jego drugi koniec. Niestety, informacje o połączeniach zostały zakodowane w sondzie i Bajtazar ich nie zna. Na szczęście pamięta on, że sonda może dostać się z dowolnego punktu do dowolnego innego punktu, być może przy wykorzystaniu innych pośrednich punktów.

W teorii sprawa jest prosta. Bajtazar może wydawać sondzie polecenia ruchu do dowolnego innego punktu, a sonda powinna poinformować Bajtazara, czy ruch się udał.

W praktyce jednak jest to trochę bardziej skomplikowane przez błędy w oprogramowaniu. Po pierwsze, jeśli Bajtazar zleci ruch do punktu, w którym sonda obecnie się znajduje, to urządzenie wybuchnie i przeprowadzenie badań będzie niemożliwe. Jeśli bezpośrednie połączenie z docelowym punktem nie istnieje, to sonda nie zmieni swojego położenia i Bajtazar otrzyma zwrotną odpowiedź negatywną. Jeśli natomiast połączenie istnieje, to sonda się tam przemieści. Jednak ze względu na kolejny błąd w oprogramowaniu, Bajtazar otrzyma pozytywną odpowiedź tylko wtedy, gdy sonda wykona parzysty udany ruch. W przeciwnym razie otrzyma on odpowiedź negatywną – dokładnie taką samą jak przy braku połączenia.

Sonda może również zbadać punkt, w którym obecnie się znajduje (czyli zebrać próbki materiałów). Aby jednak zachować porządek w badaniach, każdy punkt musi zostać zbadany dokładnie raz.

Jako że sonda znajduje się daleko od Ziemi, to wydawanie poleceń jest bardzo czasochłonne, więc Bajtazar może wydać polecenie ruchu co najwyżej milion razy.

Pomóż Bajtazarowi przeprowadzić badania i przywrócić Agencji wiarę w ich misję!

Komunikacja

To zadanie jest interaktywne. Należy napisać program, który będzie komunikował się z sondą, używając do tego dostarczonej biblioteki. Należy w tym celu dołączyć nagłówek `sonlib.h` za pomocą dyrektywy

```
#include "sonlib.h"
```

Biblioteka udostępnia następujące funkcje:

- `int GetN()` – Zwraca parametr n ($3 \leq n \leq 400$), czyli liczbę istotnych punktów na planetoidzie.
- `int GetSubtask()` – Zwraca parametr s ($0 \leq s \leq 3$) opisany poniżej w sekcji „Podzadania”.
- `int MoveProbe(int v)` – Wydaje sondzie polecenie przesunięcia się do punktu v. Następnie:
  - Jeśli nie jest spełniony warunek $1 \leq v \leq n$, sonda znajduje się obecnie w punkcie v lub jeśli program wydał wcześniej 1 000 000 poleceń `MoveProbe`, to działanie programu zostaje przerwane, a wynikiem testu jest błędna odpowiedź.
  - Jeśli nie ma połączenia z obecnego punktu do v, to sonda nie wykonuje ruchu, a funkcja zwraca 0.
  - Jeśli połączenie z obecnego punktu do v istnieje, sonda przenosi się do punktu v. Jeśli jest to parzysty *udany ruch* sondy (na przykład drugi lub dziesiąty), funkcja zwraca 1. W przeciwnym razie funkcja zwraca 0.
- `void Examine()` – Wydaje sondzie polecenie zbadania punktu, w którym obecnie się znajduje. Następnie:
  - Jeśli sonda spróbuje zbadać punkt, który już uprzednio został zbadany, to działanie programu zostaje przerwane, a wynikiem testu jest błędna odpowiedź.
  - Jeśli sonda zbadała każdy punkt dokładnie raz, biblioteka kończy działanie programu, akceptując rozwiązanie.

Zakończenie programu przez bibliotekę zostanie wykonane poprzez wywołanie komendy `exit(0)`.

Sonda znajduje się na początku w punkcie 1. Połączenia pozwalają na odwiedzenie wszystkich punktów (graf jest spójny). Połączenia są ustalone *na początku działania programu*, tj. biblioteka nie zmieni ich na podstawie wcześniejszych zapytań.

Celowe próby wpłynięcia na wewnętrzne działanie biblioteki oceniającej są zakazane.

Podzadania

We wszystkich testach zachodzi $3 \leq n \leq 400$. Twój program może wywołać funkcję `MoveProbe` co najwyżej 1 000 000 razy.

Część punktów za to zadanie można otrzymać za rozwiązanie podzadań, w których graf połączeń między punktami ma szczególną postać. Funkcja `GetSubtask` zwraca parametr s, który pozwala wykryć te podzadania:

- Jeśli $s = 0$, graf nie musi spełniać żadnych dodatkowych założeń.
- Jeśli $s = 1$, graf połączeń jest pełnym grafem dwudzielnym. Zbiór punktów można wtedy podzielić na dwa (nieznane) niepuste podzbiory A, B. Istnieje wtedy $|A| \cdot |B|$ połączeń. Każde z nich łączy punkt ze zbioru A oraz punkt ze zbioru B.
- Jeśli $s = 2$, zagwarantowane jest istnienie trzech konkretnych połączeń: między parą punktów $(1, 2)$, między parą punktów $(2, 3)$ oraz między parą punktów $(3, 1)$.
- Jeśli $s = 3$, każdy punkt jest połączony z dokładnie dwoma innymi punktami.

Możesz założyć, że każdy test wewnątrz pojedynczej grupy testów ma tę samą wartość parametru s.

Przykładowy przebieg programu

W teście przykładowym istnieją połączenia pomiędzy następującymi parami punktów: $(1, 2)$, $(1, 3)$, $(2, 3)$ oraz $(2, 4)$. Zwróć uwagę, że w tym teście wartość parametru s może być równa 0 albo 2.

Wywołana funkcja	**Wynik**	**Nowa pozycja sondy**	**Liczba udanych ruchów**	**Opis**
`GetN()`	4	1	0	Istnieją 4 istotne punkty.
`GetSubtask()`	0	1	0	Parametr s jest równy 0.
`Examine()`	—	1	0	Badamy punkt numer 1.
`MoveProbe(4)`	0	1	0	Brak połączenia punktu 1 z punktem 4.
`MoveProbe(2)`	0	2	1	Sonda wykonuje ruch. Do tej pory sonda wykonała nieparzyście wiele ruchów.
`MoveProbe(3)`	1	3	2	Parzysty ruch sondy.
`Examine()`	—	3	2	Badamy punkt numer 3.
`MoveProbe(2)`	0	2	3	
`Examine()`	—	2	3	Badamy punkt numer 2.
`MoveProbe(4)`	1	4	4	
`Examine()`	—	—	—	Sonda zbadała każdy punkt dokładnie raz. Biblioteka kończy działanie programu, a rozwiązanie przeszło test.

Bajtazar zadał łącznie 5 poleceń typu `MoveProbe`, co mieści się w limicie 1 000 000 poleceń.

Eksperymenty

Przykładowe **błędne** rozwiązanie i przykładowa biblioteka znajdują się w archiwum w dziale *Pliki* na SIO. Biblioteka może różnić się zachowaniem od tej, która zostanie użyta do oceny rozwiązań i nie spełniać założeń zadania. Ma ona jedynie pokazać sposób interakcji z programem.

Rozwiązanie `son.cpp` można skompilować w następujący sposób:

```
g++ -O3 -static -o son son.cpp sonlib.cpp -std=c++17
```

Należy zadbać o to, aby pliki `sonlib.h` oraz `sonlib.cpp` znajdowały się w tym samym folderze co rozwiązanie.

Po skompilowaniu, biblioteka przyjmuje na standardowym wejściu graf. Pierwszy wiersz wejścia powinien zawierać trzy liczby n, m, s – odpowiednio liczbę istotnych punktów, liczbę połączeń między nimi i numer podzadania. Każdy z kolejnych m wierszy powinien zawierać dwa numery punktów, które są końcami danego połączenia. Plik wejściowy odpowiadający testowi przykładowemu znajduje się w archiwum w pliku `son0.in`.