Theseus

> *Dacă toate părțile navei lui Theseus sunt schimbate una câte una de-a lungul timpului, în ce moment — dacă va fi vreunul — nava nu va mai fi aceeași?*

Pe lângă obișnuita lui sesiune de cugetare la abstract, în timpul lui liber Theseus doboară minotauri. Însă de data asta, el trebuie să treacă printr-un labirint întunecat și întortocheat. Deoarece asta nu este deloc o treabă ușoară, el îi cere ajutorul Ariadnei în a îl ghida. Labirintul poate fi reprezentat ca un graf neorientat conex cu n noduri (numerotate de la 1 la n) și m muchii, cu un nod special t, unde se află Minotaurul.

Theseus nu poate vedea deloc graful, dar Ariadne îl poate vedea. Ea și Theseus își vor forma o strategie ca să poată ajunge în siguranță la nodul unde se află Minotaurul: ea va plasa o etichetă de 0 sau 1 pe fiecare din cele m muchii. După acestea, Theseus va intra în labirint printr-un nod s pe care Ariadne nu îl cunoaște dinainte.

Din moment ce labirintul este foarte întunecat, în orice moment de timp el poate vedea doar indicele nodului în care se află, indicii nodurilor vecine și etichetele muchiilor adiacente. În plus, din cauza naturii distorsionate a labirintului, el **niciodată nu își va aminti** nicio informație despre nodurile precedente pe care le-a vizitat.

Pentru a ajunge la Minotaur în siguranță, Theseus se va muta de cel mult $min + C$ ori, unde min este numărul minim de muchii pe drumul de la s la t, și C este o constantă.

Detalii de Implementare

Va trebui să implementați două funcții:

```
std::vector<int> label(int n, std::vector<std::pair<int,int>> edges, int t);
```

- n: numărul de noduri
- $edges$: o listă de lungime m ce descrie muchiile grafului
- t: nodul destinație
- Această funcție trebuie să returneze o listă de etichete de lungime m unde al i-lea element poate fi 0 sau 1, ce reprezintă eticheta celei de-a i-a muchii pentru toate $0 \leq i < m$.
- Fiecare muchie trebuie etichetată ori cu 0 ori cu 1. Etichetarea ei cu o etichetă diferită va duce la **undefined behaviour**.
- Această funcție va fi apelată **exact o dată** pentru fiecare test.

```
int travel(int n, int u, std::vector<std::pair<int,int>> neighbours);
```

- n: numărul de noduri din graf
- u: nodul curent
- $neighbours$: o listă de perechi (v, e) reprezentând o muchie între u și v etichetată cu e
- Această funcție trebuie să returneze un nod vecin în care se va muta Theseus. Dacă nodul vecin este t, programul se va termina de la sine.
- Se garantează că pentru orice apel la această funcție, u nu va fi egal cu nodul special t.
- O apelare a acestei funcții reprezintă o mutare în labirint. Așadar, pentru fiecare test, această funcție poate fi apelată **ori de oricâte ori e necesar** pentru fiecare test.

Atenție! Programul nu are voie să folosească nicio variabilă globală/statică pentru comunicarea între diferite instanțe ale `label` sau `travel`. Orice încercare de a nu lua în seamă acest fapt va duce la **undefined behaviour**.

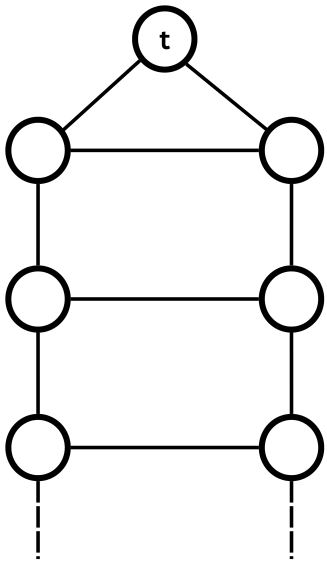
Restricții

- $1 \leq n \leq 10000$
- $1 \leq m \leq 50000$
- $C = 14$
- Nodul de start s este fixat pentru fiecare test înainte de a apela funcția `label`.

Subtaskuri

1. (4 puncte) Graful este o clică (i.e. există o muchie între oricare două noduri $1 \leq u < v \leq n$).
2. (10 puncte) Distanța dintre destinație și orice nod din graf este de cel mult 2 muchii.
3. (11 puncte) Graful este un arbore.
4. (13 puncte) Graful este bipartit (altfel spus, există o împărțire a nodurilor din graf în două submulțimi astfel încât nu există nicio muchie între două noduri din aceeași submulțime).
5. (12 puncte) Graful va fi o scară (vedeți definiția de mai jos).
6. (50 puncte) Fără restricții adiționale.

Notă: Un graf scară este un graf format din două drumuri paralele (sau lanțuri) de aceeași lungime, cu fiecare pereche de noduri corespondente conectată printr-o muchie, formând treptele scării. În plus, la unul dintre capetele scării, există un nod special — nodul destinație t — care este conectat la ambele drumuri ale scării, acționând ca un părinte comun al lor. Se garantează că n va fi impar pentru orice astfel de graf. A se observa imaginea de mai jos.



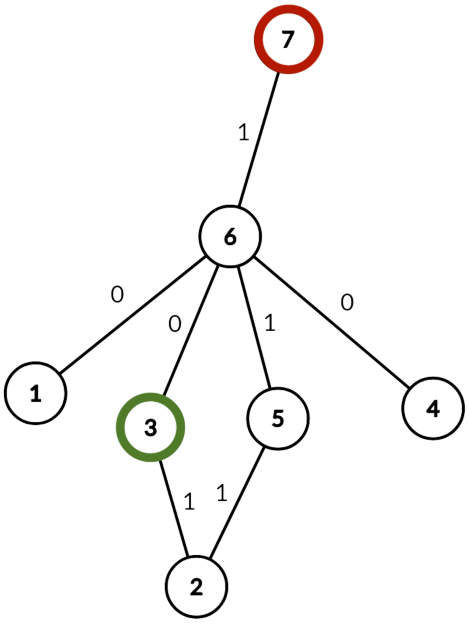
Exemplu

Exemplul 1

Să presupunem că graful nostru are 7 noduri și 7 muchii (listate mai jos). Nodul de start va fi 3 (marcat cu **verde**) și nodul destinație 7(marcat cu **roșu**). Graderul va apela mai întâi:

```
label(7, {{1, 6}, {7, 6}, {2, 5}, {3, 2}, {3, 6}, {6, 5}, {6, 4}}, 7)
```

Să presupunem că apelul la `label` returnează `{0, 1, 1, 1, 0, 1, 0}`. Atunci graful rezultat va arăta astfel:



Ce urmează este o posibilă secvență de apeluri la `travel` ce vor duce la o soluție corectă:

Apel	Valoare returnată
`travel(7, 3, {{2, 1}, {6, 0}})`	2
`travel(7, 2, {{5, 1}, {3, 1}})`	5
`travel(7, 5, {{6, 1}, {2, 1}})`	6
`travel(7, 6, {{3, 0}, {5, 1}, {1, 0}, {4, 0}, {7, 1}})`	4
`travel(7, 4, {{6, 0}})`	6
`travel(7, 6, {{3, 0}, {5, 1}, {1, 0}, {4, 0}, {7, 1}})`	7

Când valoarea returnată va fi 7 (adică destinația), programul se va opri.

Grader local

Graderul local citește inputul în formatul următor:

- linia 1: n m
- linia 2: s t
- linia $3 + i$: a b reprezentând că există o muchie ce conectează nodurile a și b.

Mai întâi graderul va apela `label` cu parametrii corespunzători și va eticheta muchiile grafului conform vectorului returnat.

Apoi va apela `travel` cu parametrii n, s și vecinii lui s. După prima apelare va continua să facă apelări succesive la `travel`, cu nodul curent fiind cel returnat de către apelarea precedentă, până când se va ajunge la nodul destinație t.

La final va afișa numărul de mutări efectuate de soluția ta și nodurile parcurse în ordinea aferentă.