# Tezej

> *Če skozi čas Tezejevi ladji zamenjamo vse dele, kdaj — če sploh — to potem ni več Tezejeva ladja?*

V prostem času, kadar ne modruje in tuhta o globinah abstraktnega, Tezej premaguje minotavre. Tokrat je pri tem postavljen pred novo zagonetko: najti mora pot skozi temân, zvit in pretkan labirint. Ker to ni enostaven podvig, se je nameril poprositi lepo Ariadno za pomoč.

Labirint si lahko predstavljamo kot neusmerjen graf z $n$ vozlišči, označenimi od $1$ do $n$, in $m$ povezavami. Imamo tudi posebno vozlišče $t$, kjer ždi Minotaver.

Za razliko od Tezeja, ki grafa sploh ne vidi, ima Ariadna popoln pregled. Skupaj se dogovorita za strategijo, s pomočjo katere bo lahko Tezej varno prišel do vozlišča z Minotavrom: Ariadna bo vsako izmed $m$ povezav označila z eno od nalepk $0$ ali $1$. Tezej bo potem vstopil v labirint na neko vozlišče $s$, ki ga Ariadna prej ne pozna.

Ker je v labirintu zelo temno, Tezej v kateremkoli trenutku vidi samo oznako vozlišča, na katerem je, oznake sosednjih vozlišč in pa Ariadnine oznake na povezavah do teh sosednjih vozlišč. Ker je labirint tako zamotan in zapleten, velja tudi, da se Tezej **nikoli ne more spomniti** nobenih informacij o prejšnjih vozliščih, ki jih je že obiskal.

Da bi varno našel Minotavra, se mora Tezej premakniti največ $min + C$-krat, kjer je $min$ najmanjše število povezav na poti od vozlišča $s$ do vozlišča $t$. Število $C$ je konstanta.

## Podrobnosti implementacije

Napisati moraš dve funkciji:

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

- `n`: število vozlišč
- `edges`: seznam dolžine `m`, ki opisuje vse povezave v grafu
- `t`: ciljno vozlišče, z Minotavrom
- Funkcija naj vrne seznam Ariadninih nalepk, torej števil $0$ ali $1$, pri čemer ima $i$-ti element seznama nalepko za povezavo $i$, za vse $0 \leq i < m$.
- Vsaka povezava mora biti označena bodisi z $0$ bodisi z $1$. Če za kako povezavo vrneš kako drugo število, bo to vodilo v **nedefinirano obnašanje** programa.
- Funkcija se kliče **natanko enkrat** za vsak testni primer.

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

- `n`: število vozlišč v grafu
- `u`: trenutno vozlišče, kjer se nahaja Tezej
- `neighbours`: seznam parov $(v, e)$, ki pomenijo, da je med vozliščema $u$ in $v$ povezava z Ariadnino oznako $e$.
- Funkcija naj vrne sosednje vozlišče, na katerega naj se premakne Tezej. Če je to $t$, se bo program samodejno zaključil.
- Zagotovljeno je, da bo za vsak klic te funkcije $u$ različen od $t$.
- Klic te funkcije predstavlja premik skozi labirint, torej bo za vsak testni primer klicana **kolikor je potrebno velikrat**, da Tezej doseže ciljno vozlišče.

**Pozor!** Program ne sme uporabljati nobenih globalnih ali statičnih spremenljivk in jih uporabljati za komuniciranje med različnimi klici `label` ali `travel`. Če bi poskušal to zaobiti, bo to povzročilo **nedefinirano obnašanje** programa.

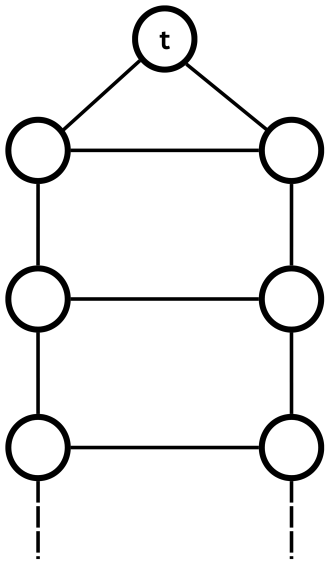
## Omejitve

- $1 \leq n \leq 10000$
- $1 \leq m \leq 50000$
- $C = 14$
- Začetno vozlišče $s$ je fiksno za vsak testni primer, še preden se pokliče funkcija `label`.

## Podnaloge

1. (4 točke) Graf je klika (tj. med vsakima dvema vozliščema $1 \leq u < v \leq n$ obstaja povezava).
2. (10 točk) Razdalja med ciljnim vozliščem in katerimkoli drugim vozliščem grafa je kvečjemu 2 povezavi.
3. (11 točk) Graf je drevo.
4. (13 točk) Graf je dvodelen (tj. obstaja način, da razdelimo vozlišča v dve podmnožici tako, da so vse povezave le med množicama, nikoli pa znotraj ene).
5. (12 točk) Graf je lestev (glej definicijo spodaj).
6. (50 točk) Brez dodatnih omejitev.

**Opomba:** Lestev je graf, ki je sestavljen iz dveh poti ali verig enakih dolžin, kjer je vsak par istoležnih vozlišč povezan, kot prečke lestve. Nadalje, na enem koncu lestve je posebno (ciljno) vozlišče $t$, ki je povezano z obema koncema lestve, in s tem deluje kot skupen prednik. Zagotovljeno je, da bo $n$ lih za vsak tak graf. Oglej si sliko spodaj:

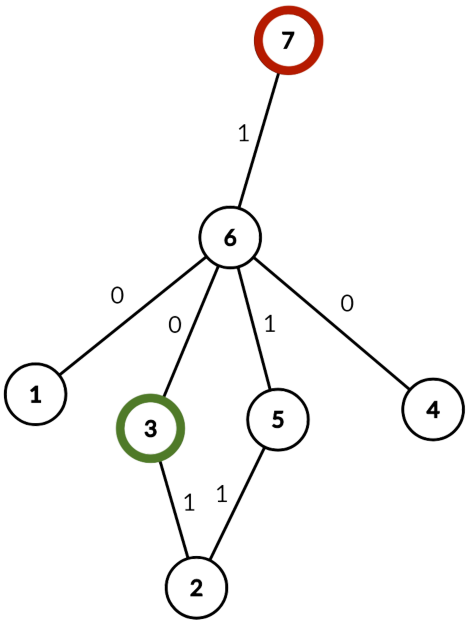



## Primer

Poglejmo graf s $7$ vozlišči in $7$ povezavami (naštetimi spodaj). Začetno vozlišče je $3$ (označeno z **zeleno**), ciljno pa $7$ (označeno z **rdečo**). Ocenjevalec bo najprej poklical:

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

Predpostavimo, da ta klic `label` vrne `{0, 1, 1, 1, 0, 1, 0}`. Rezultat je takle graf:



V tabeli je eno od možnih zaporedij klicev `travel`, ki vodi do pravilne rešitve:

| Klic | Vrnjena vrednost |
|---|---|
| `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 |

Ko je vrnjena vrednost 7 (tj. ciljno vozlišče), se program ustavi.

## Testni ocenjevalec

Ocenjevalec prebere vhodne podatke v tej obliki:

- Prva vrstica: števili $n$ in $m$.
- Druga vrstica: števili $s$ in $t$.
- Vrstice $3 + i$: števili $a$ in $b$, ki pomenita, da sta vozlišči $a$ in $b$ povezani.

Ocenjevalec bo najprej klical `label` z ustreznimi parametri in si shranil oznake povezav glede na vrnjen seznam.

Potem bo klical `travel` s parametri $n$, $s$ in sosedi $s$. Po prvem klicu bo zaporedoma klical `travel` tako, da bo trenutno vozlišče tisto, ki ga je vrnil prejšnji klic, dokler funkcija enkrat ne vrne vozlišča $t$.

Na koncu bo izpisal število potez, ki jih je tvoja rešitev naredila, in po vrsti vozlišča, ki jih je vračala.