# Théseus

> *Pokud by se po několika staletích údržby jednotlivé části Théseovy lodi postupně vyměnily, byla by to stále tatáž loď?*

Když Théseus hluboce neuvažuje nad výše uvedenou otázkou, tak ve volném čase buď vozí sůl nahoru a dolů nebo zabíjí Mínótaury. V takovém případně ale musí nejprve projít temným a klikatým labyrintem. Protože toto zrovna není jeho silná stránka a bojí se, že se zdrží jako rumunský vlak, požádal Izabelu o vedení skrze labyrint. Labyrint si můžeme představit jako souvislý neorientovaný graf s $n$ vrcholy (očíslovanými od $1$ do $n$) a $m$ hranami, se speciálním vrcholem $t$, ve kterém se nachází Mínótauros.

Théseus nevidí celý graf, ale Izabela ano. Ona a Théseus se domluví na strategii, jak ho navézt do vrcholu, kde je Mínótauros. Izabela umístí cedulku s číslem $0$ nebo $1$ na každou z hran. Poté Théseus vstoupí do labyrintu skrze vrchol $s$, který Izabela předem nezná.

Protože je velká tma, Théseus vždy vidí pouze číslo vrcholu, ve kterém je, čísla sousedních vrcholů a cedulky na přilehlých hranách. Navíc díky tomu, že se při průchodu hranou vždy Théseus praští do hlavy, tak si **nemůže zapamatovat** žádnou informaci o předchozích vrcholech, které navštívil.

Aby Théseus Mínótaura bezpečně dosáhl, tak může udělat nejvýše $min + C$ kroků, kde $min$ je minimální počet hran na cestě z $s$ do $t$ a $C$ je níže specifikovaná konstanta.

## Implementační podrobnosti

Máte implementovat následující funkce:

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

- $n$ je počet vrcholů.
- $edges$ je vektor délky $m$ popisující hrany grafu.
- $t$ je cílový vrchol (kde je Mínótauros).
- Funkce by měla vrátit vektor délky $m$, kde $i$-tý prvek může být buď $0$ nebo $1$, reprezentující cedulku na $i$-té hraně pro každé $0 \leq i < m$.
- Každá hrana musí být ocedulkována buď pomocí $0$ nebo $1$. Umístění jiné cedulky vede na **nedefinované chování**.
- Tato funkce je zavolaná **právě jednou** za každé spuštění programu.

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

- $n$ je počet vrcholů.
- $u$: aktuální vrchol.
- $neighbours$ je vektor dvojic $(v, e)$ značící, že existuje hrana mezi $u$ a $v$ s cedulkou $e$.
- Funkce by měla vrátit sousedící vrchol, do kterého se má Théseus přesunout. Pokud je tento vrchol $t$, tak program automaticky skončí.
- Je garantováno, že v žádném volání této funkce nebude vrchol $u$ totožný se speciálním vrcholem $t$.
- Volání této funkce reprezentuje pohyb labyrintem. Proto v každém spuštění programu tato funkce může být volána **tolikrát, kolikrát bude potřeba** k dosažení cíle.

**Varování!** Program by neměl používat žádné globální/statické proměnné a používat je pro komunikaci mezi různými voláními `label` a `travel`. Jakýkoliv pokus o porušení tohoto pravidla vede k **nedefinovanému chování**.

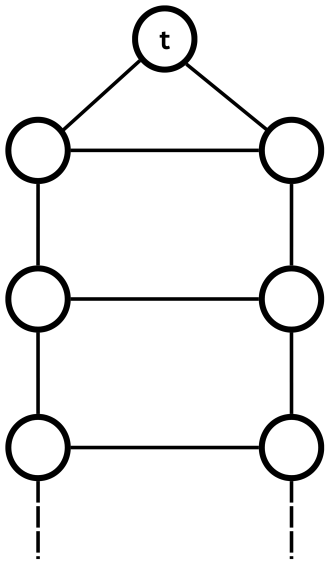
## Omezení

- $1 \leq n \leq 10\,000$
- $1 \leq m \leq 50\,000$
- $C = 14$
- Startovní vrchol $s$ je v každém testu zafixován před zavoláním `label`.

## Podúlohy

1. (4 body) Graf je klika (tj. existuje hrana mezi každými dvěma vrcholy $1 \leq u < v \leq n$).
2. (10 bodů) Vzdálenost mezi cílem a jakýmkoliv vrcholem v grafu je nejvýše $2$ hrany.
3. (11 bodů) Graf je strom.
4. (13 bodů) Graf je bipartitní (tj. existuje způsob, jak rozdělit vrcholy grafu na dvě podmnožiny takové, že neexistuje hrana mezi žádnými dvěma vrcholy ze stejné množiny).
5. (12 bodů) Graf je žebřík (viz definice níže).
6. (50 bodů) *Bez dalších omezení.*

**Poznámka:** *Žebřík* je graf, co se skládá s dvou rovnoběžných cest stejné délky, kde je každá dvojice odpovídajících vrcholů spojená hranou, příčkou žebříku. Dále je na jednom z konců žebříku speciální vrchol – cíl $t$, který je spojený s oběma konci žebříku, čímž se chová jako jejich společný rodič. V takovémto grafu bude $n$ vždy liché. Viz obrázek níže.

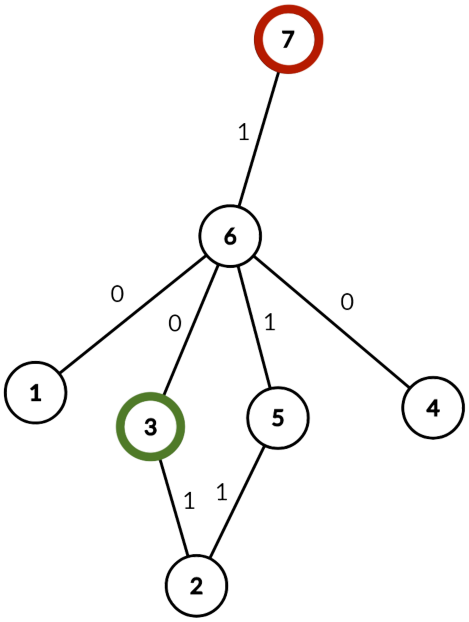



## Ukázka

Uvažujme graf se $7$ vrcholy a $7$ hranami uvedenými níže. Počáteční vrchol bude mít číslo $3$ (označený **zeleně**) a vrchol s Mínótaurem bude mít číslo $7$ (označený **červeně**). Grader nejprve zavolá:

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

Předpokládejme, že zavolání `label` vrátí `{0, 1, 1, 1, 0, 1, 0}`. Pak bude výsledný graf vypadat takto:




Následuje posloupnost volání `travel`, která povede ke správnému řešení:

| Volání | Návratová hodnota |
|---|---|
| `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 |

Poté, co je vrácena hodnota $7$ (tj. destinace), program skončí.

## Ukázkový grader

Ukázkový grader čte vstup v následujícím formátu:

- řádek $1$: $n$ $m$
- řádek $2$: $s$ $t$
- řádek $3 + i$: $a$ $b$ udává, že existuje hrana mezi $a$ a $b$.

Ukázkový grader nejprve zavolá `label` s odpovídajícími parametry a označí hrany podle vráceného vektoru.

Poté zavolá `travel` s parametry $n$, $s$ a sousedy $s$. Po prvním volání bude dále volat `travel`, kde aktuální vrchol bude ten vrácený předchozím voláním, dokud nedojde do destinace $t$.

Nakonec vypíše počet kroků, které vaše řešení udělalo, a vrcholy, které po řadě dostalo.