Thészeusz (Theseus)

> *Ha a Thészeusz hajójának minden alkatrészét egyenként kicserélik az idő múlásával, melyik ponton — ha van ilyen — szűnik meg ugyanaz a hajó lenni?*

Amikor éppen nem az absztrakt dolgokon töpreng, Thészeusz szabadidejében minótauroszokat öl. Ezúttal azonban előbb egy sötét és kanyargós labirintuson kell áthaladnia. Mivel ez nem könnyű feladat, Ariadné segítségét kéri, hogy vezesse őt. A labirintus egy összefüggő irányítatlan gráfnak tekinthető n csúccsal (1-től n-ig címkézve) és m éllel, egy t kitüntetett csúccsal, ahol a Minótaurosz ül.

Thészeusz egyáltalán nem látja a gráfot, Ariadné viszont igen. Thészeusszal közösen kialakítanak egy stratégiát, hogy a férfi biztonságosan eljusson a Minótauroszhoz: a nő minden m élen elhelyez egy 0 vagy 1 feliratú címkét. Ezek után Thészeusz egy olyan s csúcson keresztül fog belépni a labirintusba, amelyet Ariadné előre nem ismer.

Mivel nagyon sötét van, minden pillanatban csak annak a csúcsnak az indexét látja, amelyben van, illetve a szomszédos csúcsok indexeit és a szomszédos élek címkéit. Továbbá, a labirintus csavaros jellege miatt **soha nem tud felidézni** semmilyen információt a korábban meglátogatott csúcsokról.

Ahhoz, hogy biztonságosan elérje a Minótauroszt, Thészeusznak legfeljebb $min + C$ alkalommal kell mozognia, ahol min az s-től t-ig tartó útvonal éleinek minimális száma, C pedig egy konstans.

Implementációs részletek

A következő két függvényt kell implementálnod:

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

- n: a csúcsok száma
- $edges$: egy m hosszú lista, amely a gráf éleit írja le
- t: a célcsúcs
- Ez a függvény a címkék egy m hosszú listáját kell visszaadja, ahol az i-edik elem 0 vagy 1 lehet és az i-edik él címkéjét reprezentálja minden $0 \leq i < m$ esetén.
- Minden él címkéje csak 0 vagy 1 lehet. Ha bármi mással címkézzük fel, akkor az **nem definiált viselkedéshez** vezet (**undefined behavior**)
- Minden tesztesetben **pontosan egyszer** hívjuk meg ezt a függvényt

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

- n: a gráf csúcsainak a száma
- u: a jelenlegi csúcs
- $neighbours$: egy (v, e) párokból álló lista, amely azt jelzi, hogy az u és v csúcsok között vezet él, melynek címkéje e
- Ez a függvény egy szomszédos csúcsot kell visszadjon, ahova szeretnénk továbbmenni. Ha ez a csúcs a t csúcs, akkor a program futása automatikusan véget ér.
- Garantált, hogy a függvény bármilyen meghívásakkor az u csúcs különbözik a kitüntetett t csúcstól
- Ennek a függvénynek a meghívása egy lépést reprezentál a labirintusban. Ezért minden tesztesetben ezt a függvényt **annyiszor hívhatják meg, ahányszor ez szükséges.**

Figyelem! A program ne használjon globális/statikus változókat a `label` vagy `travel` különböző példányai közötti kommunikációra. Bármilyen kísérlet ennek megkerülésére **nem definiált viselkedéshez** vezet.

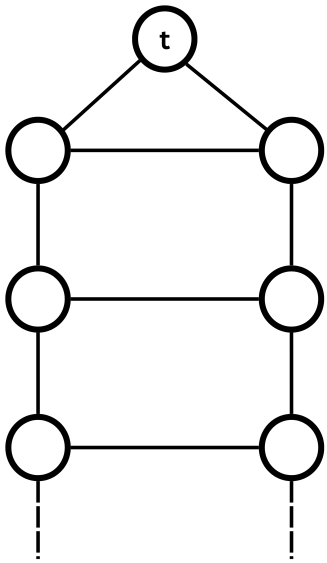
Korlátok

- $1 \leq n \leq 10000$
- $1 \leq m \leq 50000$
- $C = 14$
- Az s célcsúcs minden teszteset esetén a `label` függvény meghívása előtt már rögzített

Részfeladatok

1. (4 pont) A gráf teljes (azaz van él minden $1 \leq u < v \leq n$ csúcsok között).
2. (10 pont) A gráf bármelyik csúcsa és a célcsúcs közötti távolság maximum 2 él.
3. (11 pont) A gráf egy fa.
4. (13 pont) A gráf páros (azaz a gráf csúcsait két részhalmazra lehet osztani úgy, hogy két azonos részhalmazba tartozó csúcs között ne legyen él.).
5. (12 pont) A gráf egy létra (a definíciót lásd lejjebb).
6. (50 pont) Nincsenek további megkötések.

Megjegyzés: A létra egy olyan gráf, amely két azonos hosszúságú párhuzamos útból (vagy láncból) áll, és a láncok megfelelő csúcsait egy él köti össze, amelyek a létra lépcsőfokait alkotják. Ezen kívül a létra egyik végén van egy speciális csúcs $-$ a t célcsúcs $-$, amely a létra mindkét végpontjához kapcsolódik, és gyakorlatilag közös szülőként működik. Garantált, hogy n páratlan lesz bármelyik ilyen gráf esetében. Lásd az alábbi képet.



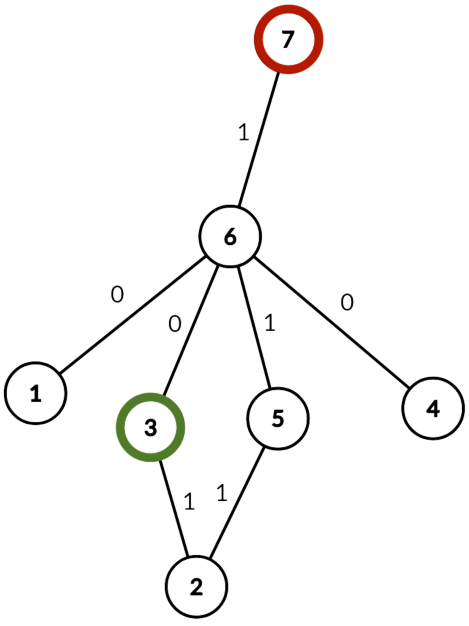
Példák

1. példa

Tegyük fel, hogy van egy 7 csúcsú gráfunk, melyben 7 él van (alább felsorolva). A kezdőcsúcs a 3 (**zölddel** jelölve) és a célcsúcs a 7 (**pirossal** jelölve). Az értékelő először a következőt hívja:

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

Tegyük fel, hogy a `label` meghívása a `{0, 1, 1, 1, 0, 1, 0}` listát adja vissza. Ekkor az eredmény gráf a következőképpen néz ki:



A következőkben a `travel` függvény egy lehetséges hívás-sorozata szerepel, amely egy helyes megoldáshoz vezet:

Hívás	Visszatérési érték
`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

Amikor a visszatérési érték 7 (vagyis a célcsúcs), a program leáll.

Minta értékelő

A minta értékelő az inputot a következő formátumban olvassa:

- 1. sor: n m
- 2. sor: s t
- $3 + i$. sor: a b, jelölve, hogy a és b csúcsok között vezet él

Először az értékelő meghívja a `label` függvényt a megfelelő paraméterekkel, és felcímkézi a gráf éleit a visszaadott vektorral.

Ezután meghívja a `travel` függvényt a következő paraméterekkel: n, s és az s szomszédai. Az első hívás után egymás után többször hívja meg a `travel` függvényt, ahol a jelenlegi csúcs az előző hívás visszatérési értéke, amíg a t célcsúcsot el nem éri.

A végén kiírja a megoldásod lépéseinek számát és a meglátogatott csúcsokat a látogatás sorrendjében.