Тезей

> *Ако всички части от кораба на Тезей се подменят една по една, в кой момент — ако някога — това спира да бъде същия кораб?*

Когато не е дълбоко в абстрактни размишления, Тезей прекарва свободното си време убивайки минотаври. Този път обаче той първо трябва да премине през тъмен и усукан лабиринт. Тъй като това не е лесна задача, той моли Ариадна да го насочва. Лабиринтът може да бъде разгледан като свързан неориентиран граф с n върха (номерирани с числата от 1 до n), m ребра и специален връх t, където се намира Минотавъра.

Тезей не вижда нищо от графа, а Ариадна вижда целия граф. Тя и Тезей ще измислят стратегия, така че той да може да достигне до върха с Минотавъра: Ариадна ще маркира всяко от m-те ребра или с 0, или с 1. След това Тезей ще влезе в лабиринта през връх s, който не е известен на Ариадна.

Понеже в лабиринта е много тъмно, във всеки един момент Тезей вижда само индекса на върха, в който е, индексите на съседните върхове, както и маркерите върху ребрата към съседните върхове. Заради сложната форма на лабиринта той **не може да помни** никаква информация свързана с предишни върхове, в които е бил.

За да достигне Минотавъра безопасно, Тезей трябва да се придвижи най-много $min + C$ пъти, като min е най-малкият брой ребра на път между s и t, а C е константа.

Детайли по имплементацията

Трябва да имплементирате две функции:

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

- n: броят върхове
- $edges$: списък с дължина m, описващ ребрата в графа
- t: специалният връх, съдържащ Минотавъра
- Тази функция трябва да върне списък с дължина m, съдържащ маркерите на ребрата. Елементът на позиция i трябва да е 0 или 1 и задава маркера на i-тото ребро за $0 \leq i < m$.
- Всяко ребро трябва да бъде маркирано или с 0, или с 1. Маркирането на ребро с различна стойност ще доведе до **недефинирано поведение (undefined behavior)**.
- Тази функция ще бъде извикана **точно веднъж** за всеки тест.

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

- n: броят върхове в графа
- u: текущият връх, в който е Тезей
- $neighbours$: списък от двойки (v, e), обозначаващи, че съществува ребро между u и v с маркер e
- Тази функция трябва да върне съседния връх, към който да се придвижва Тезей. Ако съседният връх е t, програмата ще приключи веднага.
- Гарантирано е, че за всяко извикване на функцията, u няма да бъде равно на специалния връх t.
- Едно извикване на тази функция отговаря на едно предвижване през лабиринта. Съответно за всеки тест функцията може да бъде извикана **колкото на брой пъти се налага** за да се достигне специалния връх.

Внимание! Не е позволено програмата Ви да използва глобални или статични променливи за комуникация между различни извиквания на `label` или `travel`. Всякакви опити това да бъде направено ще доведат до **недефинирано поведение (undefined behavior)**.

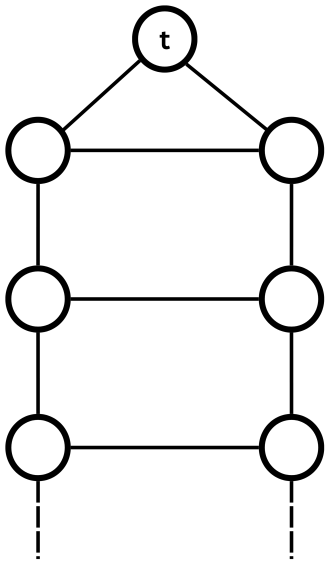
Ограничения

- $1 \leq n \leq 10000$
- $1 \leq m \leq 50000$
- $C = 14$
- Началният връх s е фиксиран за всеки тест преди извикване на функцията `label`.

Подзадачи

1. (4 точки) Графът е пълен (т.е. има ребро между всеки два върха $1 \leq u < v \leq n$).
2. (10 точки) Разстоянието между върха t и всеки друг връх в графа е най-много 2 ребра.
3. (11 точки) Графът е дърво.
4. (13 точки) Графът е двуделен (т.е. има начин върховете да се разделят на две групи, така че да няма ребро между два върха от една група).
5. (12 точки) Графът е стълба (вижте определението по-долу)
6. (50 точки) Няма допълнителни ограничения.

Забележка: Граф стълба е граф, състоящ се от два успоредни пътя (или вериги) с еднаква дължина, като между всяка двойка съответстващи върхове има ребро, образуващо стъпало от стълбата. Допълнително, в единия край на стълбата е специалният връх t, който е свързан и с двата края на стълбата, действащ като техен общ родител. Гарантирано е, че n ще бъде нечетно за този тип графи. Вижте картинката по-долу за пример.

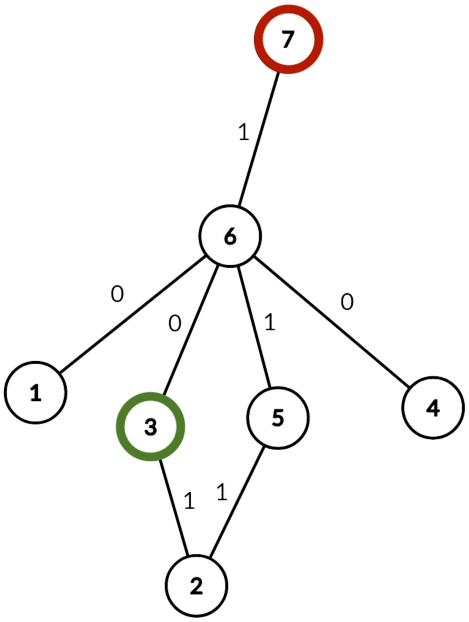


Пример

Да разгледаме граф със 7 върха и 7 ребра (изброени по-долу). Нека началният връх е 3 (обозначен със **зелено**) и крайният е 7 (обозначен с **червено**). Първоначално грейдърът ще извика:

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

Да приемем, че извикването към `label` връща `{0, 1, 1, 1, 0, 1, 0}`. Тогава графът би изглеждал като:



Следва поредица от примерни извиквания към функцията `travel`, които водят до вярно решение:

Извикване	Върната стойност
`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

При върната стойност 7 (т.е. крайния връх), програмата спира.

Локален грейдър

Локалният грейдър чете входа в следния формат:

- ред 1: n m
- ред 2: s t
- ред $3 + i$: a b обозначаващи, че има ребро свързващо a и b.

Грейдърът първо ще извика `label` със съответните параметри и ще получи маркерите за ребрата на графа.

След това грейдърът ще извика `travel` с параметри n, s и съседите на s. След първото извикване той ще продължава да прави извиквания към `travel`, използвайки за текущ връх този, който предното извикване е върнало, докато не се достигне крайния връх t.

В края грейдърът ще изведе броя придвижвания, които Вашето решение е направило, както и върховете, които са били посетени от него по ред.