თეზევსი

> *თუ დროის განმავლობაში თეზევსის გემის ყველა ნაწილი რიგრიგობით შეიცვლება, რა მომენტიდან — თუ საერთოდ არსებობს — აღარ იქნება ის იგივე გემი?*

თეზევსმა უნდა გამოაღწიოს ლაბირინთიდან. ამაში მას არიადნე დაეხმარება. ლაბირინთი შეგვიძლია წარმოვიდგინოთ, როგორც არამიმართული გრაფი n წვერო (1-დან n-მდე გადანომრილი) და m წიბოთი, რომლის t წვეროშიც ზის მინოტავრი.

თეზევსს არ შეუძლია გრაფის დანახვა, მაგრამ არიადნეს შეუძლია. არიადნე დაეხმარება თეზევსს მივიდეს მინოტავრამდე: ის თითოეულ წიბოს დააწერს 0-ს ან 1-ს. ამის შემდეგ კი თეზევსი დადგება s წვეროში, რომლის ნომერიც არიადნემ წინასწარ არ იცის.

იმის გამო რომ ლაბირინთში ბნელა, თეზევსს შეუძლია დაინახოს იმ წვეროს ნომერი, რომელშიც ახლა იმყოფება, ამ წვეროს მეზობელი წვეროების ნომრები და რიცხვები, რომლებიც ამ წვეროდან გამომავალ წიბოებს აწერია. ასევე, ლაბირინთის ზებუნებრივობის გამო, **თეზევსს ავიწყდება ყველა ინფორმაცია, რომელიც აქამდე ნამყოფ წვეროებზე იცოდა**.

იმისათვის რომ თეზევსმა მიაღწიოს მინოტავრს, ის უნდა გადაადგილდეს მაქსიმუმ $min + C$-ჯერ, სადაც min არის მინიმალური წიბოების რაოდენობა s-დან t-მდე, ხოლო C არის მუდმივა.

იმპლემენტაციის დეტალები

თქვენ უნდა დაწეროთ ორი ფუნქცია:

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

- n: წვეროების რაოდენობა
- $edges$: m ზომის წიბოების მასივი
- t: დანიშნულების ადგილი (წვერო)
- ფუნქციამ უნდა დააბრუნოს m ზომის მასივი, რომელშიც წერია თითოეულ წიბოზე დაწერილი რიცხვი - 0 ან 1.
- მასივში 0 და 1-ისაგან განსხვავებული რიცხვის ჩაწერის შემთხვევაში პროგრამა **გაურკვეველად** მოიქცევა.
- თითოეული ტესტისთვის ეს ფუნქცია გამოიძახება **ზუსტად ერთხელ**.

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

- n: წვეროების რაოდენობა
- u: წვერო, რომელშიც თეზევსი ახლა იმყოფება
- $neighbours$: წყვილების (v, e) სია, რაც ნიშნავს, რომ u-სა და v-ს შორის არის წიბო, რომელზეც აწერია e
- ამ ფუნქციამ უნდა დააბრუნოს მეზობელი წვერო, რომელშიც თეზევსი გადავა. თუ ეს წვერო აღმოჩნდა t, პროგრამა ავტომატურად მორჩება.
- გარანტირებულია, რომ u არ იქნება t-ს ტოლი.
- თითოეული ტესტისთვის ეს ფუნქცია გამოიძახება **რამდენჯერაც საჭიროა**, სანამ თეზევსი საბოლოო წვეროს არ მიაღწევს.

ყურადღება! თქვენმა პროგრამამ არ უნდა გამოიყენოს გლობალური/სტატიკური ცვლადები, იმისათვის რომ ინფორმაცია გაცვალოს `label` და `travel` ფუნქციებს შორის. ნებისმიერ ასეთ შემთხვევაში პროგრამა მოიქცევა **გაურკვევლად**.

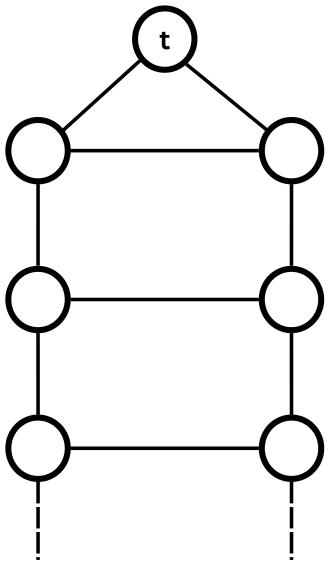
შეზღუდვები

- $1 \leq n \leq 10000$
- $1 \leq m \leq 50000$
- $C = 14$
- ყოველი ტესტისთვის საწყისი წვერო s ფიქსირებულია `label` ფუნქციის გამოძახებამდე.

ქვეამოცანები

1. (4 points) გრაფი კლიკია (ანუ ყოველ ორ წვეროს შორის არის წიბო).
2. (10 points) მანძილი ნებისმიერ წვეროსა და საბოლოო წვეროს შორის არის მაქსიმუმ 2 წიბო.
3. (11 points) გრაფი ხეა.
4. (13 points) გრაფი ორწილაა (ანუ გრაფის წვეროები შეგვიძლია გავყოთ ორ სეტად, ისე რომ არცერთი ორი წვერო, რომელიც ერთსა და იმავე სეტშია, ერთმანეთთან დაკავშირებული არ არის).
5. (12 points) გრაფი არის კიბე (განმარტება იხილეთ ქვემოთ).
6. (50 points) დამატებითი შეზღუდვების გარეშე.

Note: კიბე არის გრაფი, რომელშიც არის ორი ტოლი სიგრძის პარალელური გზა (ან ჯაჭვი), რომელთა შესაბამისი წვეროები დაკავშირებული არიან ერთმანეთთან. ასევე, კიბის ერთ ბოლოში მდებარეობს საბოლოო წვერო $-$ t $-$, რომელიც უკავშირდება კიბის ბოლოებს. გარანტირებულია, რომ ასეთ გრაფში n იქნება კენტი. იხილეთ სურათი ქვემოთ.



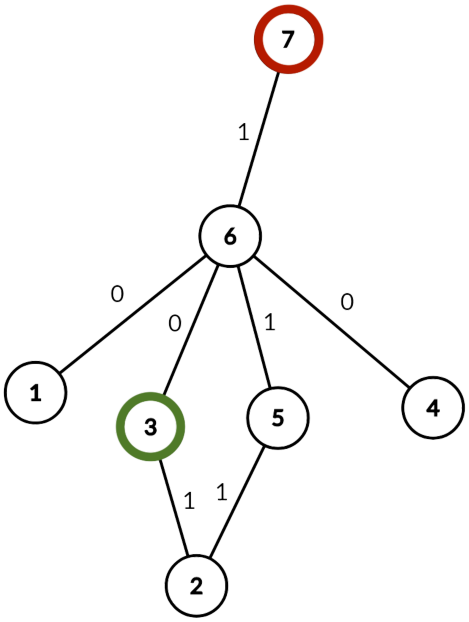
მაგალითები

მაგალითი 1

დავუშვათ გვაქვს გრაფი 7 წვეროთი და 7 წიბოთი. საწყისი წვერო არის 3 (marked with **green**) ხოლო საბოლოო წვერო არის 7 (marked with **red**). grader ჯერ გამოიძახებს:

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

დავუშვათ `label`-მა დააბრუნა `{0, 1, 1, 1, 0, 1, 0}`. შესაბამისად გრაფი გამოიყურება ასე:



`travel`-ის გამოძახების შესაძლო სწორი მიმდევრობა არის შემდეგი:

Call	Returned value
`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

როცა დაბრუნებული მნიშვნელობა არის 7 (საბოლოო წვერო), პროგრამა გაჩერდება.

Sample grader

sample grader კითხულობს შემომავალ მონაცემებს შემდეგ ფორმატში :

- ხაზი 1: n m
- ხაზი 2: s t
- ხაზი $3 + i$: a b აღნიშნავს, რომ a და b წვეროები არიან მეზობლები.

თავიდან grader გამოიძახებს `label`-ს შესაბამისი პარამეტრებით და წიბოებს დააწერს დაბრუნებულ მასივს.

შემდეგ ის გამოიძახებს `travel`-ს პარამეტრებით n, s და s-ის მეზობლები. ამის შემდეგ `travel` გამოიძახება წინა გამოძახებისგან დაბრუნებული მნიშვნელობით სანამ, t-ს არ მიაღწევს.

ბოლოს დაბეჭდავს გადასვლების რაოდენობას და წვეროებს, რომლებიც მოიარა, მოვლის თანმიმდევრობით.