Tezeusz

> *Jeśli będziemy wymieniali po kolei wszystkie części statku Tezeusza, jedna po drugiej, to kiedy (jeśli w ogóle) przestanie to być ten sam statek?*

W czasie wolnym od prowadzenia rozważań filozoficznych, Tezeusz zwykł zabijać potwory. Tym razem jednak, musi on przejść przez ciemny i zawiły labirynt. Zdaje on sobie sprawę z tego, że nie jest to błahostka. Poprosił więc Ariadnę o pomoc w przeprowadzeniu go przez korytarze. Labirynt to po prostu spójny graf z n wierzchołkami (ponumerowanymi od 1 do n) i m krawędziami, wraz z wyróżnionym specjalnym wierzchołkiem t zamieszkałym przez Minotaura.

Tezeusz nie widzi grafu, ale za to Ariadna tak. Stworzą oni strategię, by pomóc mu bezpiecznie dostać się do wierzchołka z Minotaurem. Ariadna poetykietyuje wszystkie m krawędzi za pomocą liczb 0 i 1. Następnie, Tezeusz wejdzie do labiryntu pewnym wierzchołkiem s, nieznanym Ariadnie.

Jako że jest ciemno, w dowolnym momencie widzi on tylko indeks wierzchołka, w którym się znajduje oraz indeksy jego sąsiadów wraz z etykietami prowadzących do nich krawędzi. Ponadto, z powodu zawiłości labiryntu, **nie jest on w stanie odtworzyć** żadnej informacji o poprzednio odwiedzonych wierzchołkach.

Żeby bezpiecznie dotrzeć do Minotaura, Tezeusz może wykonać co najwyżej $min + C$ przejść krawędziami, gdzie min jest minimalną liczbą krawędzi na ścieżce między s i t a C jest stałą.

Szczegóły implementacyjne

Musisz zaimplementować następujące dwie funkcje:

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

- n: liczba wierzchołków
- $edges$: lista m krawędzi grafu
- t: docelowy wierzchołek
- Procedura ta powinna zwrócić listę m etykiet równych 0 lub 1 i odpowiadających liczbom do wpisania w kolejne krawędzie
- Zwrócenie którejkolwiek etykiety różnej od 0 i 1 prowadzi do **undefined behaviour**.
- Funkcja ta jest wywoływana **dokładnie raz** w każdym teście

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

- n: liczba wierzchołków
- u: aktualny wierzchołek
- $neighbours$: lista par (v, e) oznaczających krawędź pomiędzy u i v z etykietą e
- Funkcja ta powinna zwracać numer wierzchołka, do którego należy przejść. Jeśli tym wierzchołkiem jest t, program automatycznie kończy działanie.
- Zagrawantowane jest, że w dowolnym wywołaniu tej funkcji u będzie różne od t.
- Jako że funkcja ta reprezentuje ruch w grafie, w każdym teście może ona być wywołana **tyle razy ile potrzeba**, by dotrzeć do wierzchołka docelowego.

Uwaga! Program nie powinien używać żadnych globalnych ani statycznych zmiennych do komunikacji pomiędzy różnymi wywołaniami funkcji `label` i `travel`. Dowolna taka próba doprowadzi do **undefined behaviour**.

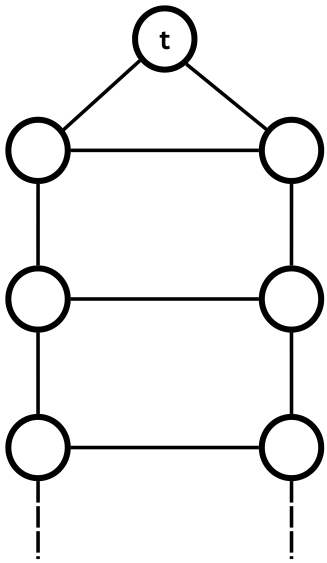
Ograniczenia

- $1 \leq n \leq 10000$
- $1 \leq m \leq 50000$
- $C = 14$
- Wierzchołek startowy s jest ustalony przed wywołaniem funkcji `label`.

Podzadania

1. (4 punkty) Graf jest kliką (istnieje krawędź pomiędzy dowolnymi dwoma wierzchołkami $1 \leq u < v \leq n$).
2. (10 punktów) Pomiędzy wierzchołkiem docelowym a dowolnym innym wierzchołkiem istnieje ścieżka składająca się z co najwyżej dwóch krawędzi.
3. (11 punktów) Graf jest drzewem.
4. (13 punktów) Graf jest dwudzielny (da się podzielić wierzchołki na dwa podzbiory, takie że dowolna krawędź łączy dwa wierzchołki z różnych podzbiorów).
5. (12 punktów) Graf jest drabiną (definicja poniżej).
6. (50 punktów) Brak dodatkowych ograniczeń.

Uwaga: Drabina składa się z dwóch równoległych ścieżek tej samej długości, w której każda para odpowiadających sobie wierzchołków ma między sobą krawędź (tworzą one szczeble). Dodatkowo, na jednym z końców drabiny jest specjalny wierzchołek (wierzchołek docelowy t) połączony z oboma "górnymi" końcami drabiny. Jest gwarantowane, że n jest nieparzyste w dowolnym takim teście. Zwróć uwagę na poniższy obrazek.



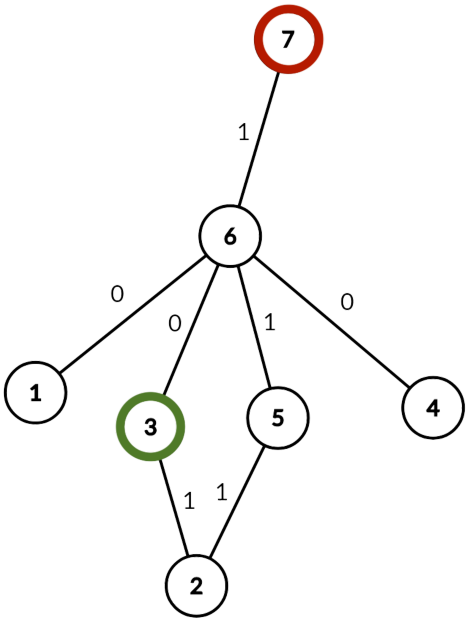
Przykład

Przykład 1

Rozważmy graf z 7 wierzchołkami i 7 krawędziami (wypisanymi poniżej). Wierzchołek startowy to 3 (oznaczony kolorem **zielonym**) a docelowy to 7 (oznaczony kolorem **czerwonym**). Sprawdzaczka wywoła najpierw:

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

Załóżmy, że `label` zwróci `{0, 1, 1, 1, 0, 1, 0}`. Wtedy zwrócony graf będzie wyglądał następująco:



Następuje wtedy ciąg wywołań `travel`. Przykładowy taki ciąg, prowadzący do dobrego wyniku to:

Wywołanie	Zwrócona wartość
`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

Po zwróceniu 7 (wierzchołka docelowego), program się zatrzymuje.

Przykładowa sprawdzaczka

Przykładowa sprawdzaczka czyta dane w następującym formacie:

- linia 1: $n\ m$
- linia 2: $s\ t$
- linia $3 + i$: $a\ b$ oznaczające, że istnieje krawędź łącząca a i b.

Najpierw sprawdzaczka wywoła `label` z odpowiednimi parametrami i przypisze wierzchołkom etykiety zgodnie z wektorem zwróconym przez tę funkcję.

Wtedy wywoła `travel` z parametrami n, s i sąsiadami s. Potem będzie ona wywoływała `travel` dla wierzchołków będących wynikami poprzedniego wywołania tej funkcji, aż osiągnięty zostanie wierzchołek t.

Na końcu sprawdzaczka wypisze ciąg ruchów wykonanych przez Twój program.