Станції (stations)

Основа Інтернету Сінгапура (OIC) складається з n станцій, яким присвоєно **індекси** від 0 до $n-1$. Також є $n-1$ двосторонніх з'єднань, пронумерованих від 0 до $n-2$. Кожне з'єднання сполучає дві різні станції. Дві станції, сполучені одним з'єднанням, називаються сусідами.

Шляхом від станції x до станції y називається послідовність різних станцій $a_0, a_1, \cdots, a_p$, така що $a_0 = x$, $a_p = y$, та кожні дві послідовні станції на шляху є сусідами. Існує **точно один** шлях від довільної станції x до іншої довільної станції y.

Довільна станція x може створити пакет (набір даних) та відправити його на якусь іншу станцію y, яка називаться **ціллю** пакету. Цей пакет потрібно провести вздовж єдиного шляху від x до y наступним чином. Розглянемо станцію z де у поточний момент знаходиться пакет, чиєю ціллю є y ($z \neq y$). У цій ситуації станція z:

1. виконує **процедуру проведення** що визначає сусіда z який знаходиться на єдиному шляху від z до y, та
2. відсилає пакут цьому сусіду.

Однак, станції мають обмежену пам'ять та не зберіають повний список з'єднань у OIC, щоб їх використовувати у процедурі проведення.

Вашою задачею є реалізувати схему проведення для OIC, яка складається з двох процедур.

- Першій процедурі на вході задається n, список з'єднань у OIC та ціле число $k \geq n - 1$. Вона назначає кожній станції **унікальну** цілочисельну **мітку** між 0 та k, включно.

- Друга процедура є процедура проведення, яка завантажується на всі станції після призначення міток. Їй на вхід надходить **тільки** наступне:

    - s, **мітка** станції де зараз знаходиться пакет,
    - t, **мітка** цілі цього пакету ($t \neq s$),
    - c, список **міток** усіх сусідів s.

    Вона має повертати **мітку** сусіда s, куди треба відправити пакет.

У одній з підзадач, кількість балів, що отримає ваш розв'язок, залежить від максимального значення мітки, що призначено станціям (взагалі, чим менше, тим краще).

Деталі реалізації

Ви маєте реалізувати наступні процедури:

```
int[] label(int n, int k, int[] u, int[] v)
```

- n: кількість станцій у OIC.
- k: максимальна мітка, яку можна використовувати.
- u та v: масиви розміру $n-1$, що описують з'єднання. Для кожного i ($0 \leq i \leq n-2$), з'єднання i сполучає станції з індексами $u[i]$ та $v[i]$.
- Ця процедура має повертати один масив L розміру n. Для кожного i ($0 \leq i \leq n-1$) $L[i]$ є міткою, призначеною станції з індексом i. Усі елементи масиву L мають бути різними з діапазону від 0 до k, включно.

```
int find_next_station(int s, int t, int[] c)
```

- s: мітка станції, де знаходиться пакет.
- t: мітка цільової станції пакету.
- c: масив, що задає список міток усіх сусідів s. Масив c відсортовано у зростаючому порядку.
- Ця процедура має повертати мітку сусіда s куди треба відправити пакет.

Кожен набір тестових даних включає один або більше незалежних сценаріїв (тобто, різних описів OIC). Для набору тестових даних що включає r сценаріїв, **програма**, що викликає описані вище процедури, виконується рівно два рази наступним чином.

Протягом першого запуску програми:

- процедура `label` викликається r разів,
- повернуті мітки зберігаються системою перевірки та
- `find_next_station` не викликається.

Протягом другого запуску програми:

- `find_next_station` може викликатись багато разів. Для кожного виклику обирається **випадковий** сценарій та мітки, що повернула процедура `label` для цього сценарію, використовуються як вхідні дані у `find_next_station`.
- `label` не викликається.

Зокрема, вся інформація, збережена у статичних або глобальних змінних під час першого виконання програми, не буде доступною у процедурі `find_next_station`.

Приклад

Розглянемо наступний виклик:

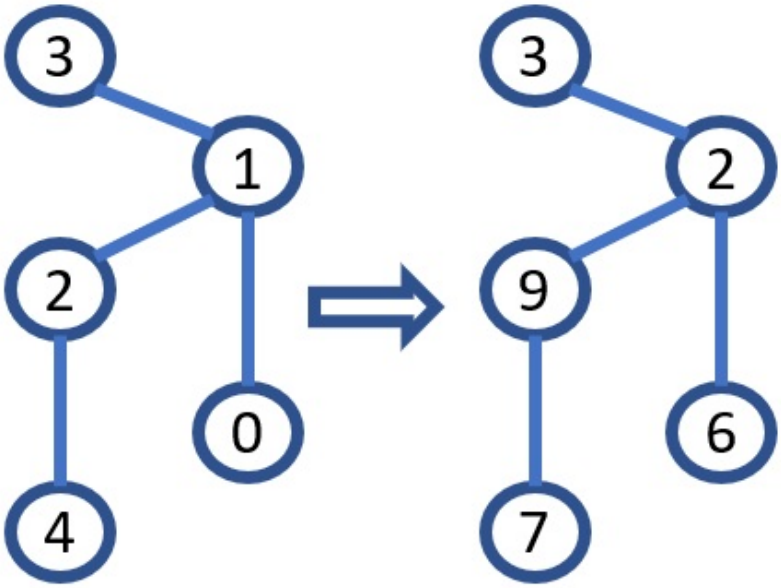
```
label(5, 10, [0, 1, 1, 2], [1, 2, 3, 4])
```

Всього є 5 станцій та 4 з'єднання що сполучають пари станцій з індексами $(0, 1)$, $(1, 2)$, $(1, 3)$ та $(2, 4)$. Кожена мітка може бути цілим числом від 0 до $k = 10$.

Щоб повідомити наступні мітки :

Індекс	Мітка
0	6
1	2
2	9
3	3
4	7

процедура `label` має повернути [6, 2, 9, 3, 7]. Чиста на наступному малюнку показують індекси (ліва сторона) та призначені мітки (права сторона).



Припустимо, що мітки призначено як описано вище та розглянемо наступний виклик:

```
find_next_station(9, 6, [2, 7])
```

Це означає, що пакет знаходиться на станції з міткою 9 та цільова станція має мітку 6. Мітки станцій на шляху до цільової станції є $[9, 2, 6]$. Отже, цей виклик має повертати 2, що є міткою станції куди треба відправити пакет (вона має індекс 1).

Розглянемо інший можливий виклик:

```
find_next_station(2, 3, [3, 6, 9])
```

Процедура має повернути 3, оскільки цільова станція з міткою 3 є сусідом станції з міткою 2 і, отже, має отримати пакет напряму.

Обмеження

- $1 \leq r \leq 10$

Для кожного виклику `label`:

- $2 \leq n \leq 1000$
- $k \geq n - 1$
- $0 \leq u[i], v[i] \leq n - 1$ (для всіх $0 \leq i \leq n - 2$)

Для кожного виклику `find_next_station`, вхідні дані стосуються випадкового обраного попереднього виклику `label`. Нехай мітки згенеровано. Тоді:

- s та t є мітками двох різних станцій.
- c є послідовністтю усіх міток сусідів станції з міткою s, у порядку зростання.

Для кожного випадку тестових даних загальна довжина усіх масивів c що передаються до процедури `find_next_station` не перевищує $100\ 000$ для всіх сценаріїв разом.

Підзадачі

1. (5 балів) $k = 1000$, всі станції мають не більше двох сусідів.
2. (8 балів) $k = 1000$, з'єднання i сполучає станції $i + 1$ та $\left\lfloor \frac{i}{2} \right\rfloor$.
3. (16 балів) $k = 1\ 000\ 000$, не більше однієї станції має більше двох сусідів.
4. (10 балів) $n \leq 8$, $k = 10^9$
5. (61 бал) $k = 10^9$

У підзадачі 5 ви можете отримати часткові бали. Нехай m буде максимумом міток що повернула `label` для всіх сценаріїв. Ваші бали для цієї підзадачі будуть обчислюватись відповідно до наступної таблиці:

Максимальна мітка	Бали
$m \geq 10^9$	0
$2000 \leq m < 10^9$	$50 \cdot \log_{5 \cdot 10^5}(\frac{10^9}{m})$
$1000 < m < 2000$	50
$m \leq 1000$	61

Приклад модуля перевірки

Приклад модуля перевірки читає вхідні дані у наступному форматі:

- рядок 1: r

Далі йдуть r блоків, кожен з яких описує окремий сценарій. Формат кожного блоку є таким:

- рядок 1: n k
- рядок $2 + i$ ($0 \leq i \leq n - 2$): $u[i]$ $v[i]$
- рядок $1 + n$: q: кількість викликів `find_next_station`.
- рядок $2 + n + j$ ($0 \leq j \leq q - 1$): $z[j]$ $y[j]$ $w[j]$: **індекси** станцій, що використовуються у j-му виклику `find_next_station`. Пакет знаходиться на станції $z[j]$, станція $y[j]$ є цільовою, та станція $w[j]$ є станцією куди треба відправити пакет.

Приклад модуля перевірки друкує результат у наступному форматі:

- рядок 1: m

Далі йдуть r блоків, що відповідають послідовним сценаріям на вході. Формат кожного блоку є таким:

- рядок $1 + j$ ($0 \leq j \leq q - 1$): **індекс** станції, чию **мітку** повернув j-й виклик `find_next_station` у цьому сценарії.

Зауважте, що кожен запуск прикладу модуля перевірки викликає як `label`, так і `find_next_station`.