# 기지국 (stations)

싱가포르 인터넷 백본(SIB)은 $n$개의 기지국으로 이루어져 있는데, 각 기지국은 0부터 $n-1$까지 **번호**가 매겨져 있다. 기지국들은 0부터 $n-2$까지 번호가 매겨진 양방향 링크로 연결되어 있다. 각 링크는 서로 다른 두 기지국을 연결한다. 같은 링크로 연결된 두 기지국은 이웃이라고 한다.

기지국 $x$에서 기지국 $y$까지 경로는 서로 다른 기지국 $a_0, a_1, \cdots, a_p$의 서열인데, $a_0 = x$이고 $a_p = y$이다. 또한 서열에서 연속한 두 기지국은 이웃이다. 어떤 두 기지국 $x$와 $y$를 고르더라도, 이 둘을 잇는 경로는 **정확하게 하나**이다.

기지국 $x$는 패킷(데이터 덩어리)을 만들고 이를 다른 기지국 $y$로 보낼 수 있는데, 이 기지국을 패킷의 **목표**라고 한다. 이 패킷은 $x$에서 $y$를 이으면서 다음 조건을 만족하는 유일한 경로를 따라 전송되어야 한다. 현재 패킷이 기지국 $z$에 있고, 이 패킷의 목표가 기지국 $y (z \neq y)$라고 하자. 이 상황에서 기지국 $z$는 다음 일을 한다.

1. **라우팅 절차**를 수행해서 $z$에서 $y$로 가는 경로에 있는 $z$의 이웃을 찾는다.
2. 패킷을 이 이웃에게 전송한다.

그러나, 기지국의 메모리가 제한되어서 싱가포르 인터넷 백본의 모든 링크를 저장하여 라우팅 절차를 수행할 수 없다.

여러분이 할 일은 싱가포르 인터넷 백본에서 사용할 라우팅 방법을 구현하는 것이다. 이는 다음 두 단계로 이루어진다.

- 첫번째 단계에서는 싱가포르 인터넷 백본의 링크의 수 $n$과 정수 $k \geq n-1$를 입력으로 받는다. 이 단계에서는 각 기지국에 0 이상 $k$ 이하인 **서로 다른** 정수 **레이블**을 할당한다.
- 두번째 단계가 라우팅 절차이고, 레이블이 할당된 다음에 모든 기지국에서 적용된다. 이 단계에서는 **오직** 다음 입력만 주어진다.
    - $s$, 현재 패킷이 있는 기지국의 **레이블**
    - $t$, 패킷의 목표 기지국의 **레이블** $(t \neq s)$,
    - $c$, $s$의 모든 이웃들의 **레이블들**의 리스트

  패킷이 전송될 $s$의 이웃의 **레이블**을 리턴해야 한다.

한 서브태스크에서는, 여러분의 점수는 기지국에 할당된 레이블 값의 최대값과 관련이 있다. (일반적으로는, 이 값이 작을 수록 좋다.)

## Implementation details

다음 함수들을 구현해야 한다.

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

- $n$: 싱가포르 인터넷 백본의 기지국 수.
- $k$: 사용할 수 있는 레이블의 최대값
- $u$ and $v$: 각각 길이 $n-1$인 배열로 링크를 기술한다. 각각 $i$에 대해서 ($0 \le i \le n-2$), 링크 $i$는 $u[i]$와 $v[i]$를 연결한다.
- 이 함수는 길이 $n$인 배열 $L$을 리턴한다. 각각 $i$ ($0 \le i \le n-1$)에 대해서 $L[i]$는 기지국 $i$에 할당된 레이블이다. $L$의 모든 원소는 서로 달라야 하며 $0$ 이상 $k$ 이하의 값이다.

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

- $s$: 패킷이 있는 기지국의 레이블
- $t$: 이 패킷의 목표 기지국의 레이블
- $c$: $s$의 모든 이웃의 레이블을 저장하는 배열. 배열 $c$는 오름차순으로 정렬되어 있다.
- 이 함수는 패킷이 전송될 $s$의 이웃의 레이블을 리턴해야 한다.

각각의 테스트 케이스는 하나 이상의 독립적인 시나리오 (즉, 싱가포르 인터넷 백본의 다른 상황)을 포함한다. $r$개의 시나리오를 포함하는 테스트 케이스 하나에 대해서, 위의 함수를 호출하는 **프로그램**은 다음과 같이 정확히 두 번 실행된다.

프로그램이 첫번째 실행될 때는 다음과 같다.

- `label` 함수가 $r$ 번 호출된다.
- 리턴된 레이블들은 채점 시스템에 저장된다.
- `find_next_station`은 호출되지 않는다.

프로그램이 두번째 실행될 때는 다음과 같다.

- `find_next_station` 함수가 여러번 호출될 수 있다. 각각의 호출마다 **임의의** 시나리오가 선택되며, 이 시나리오에서 `label` 함수에 의해 리턴된 레이블이 `find_next_station` 함수의 입력으로 사용된다.
- `label`은 호출되지 않는다.

특히, 프로그램이 첫번째 실행되었을 때 정적 변수나 전역 변수에 저장된 값들은 `find_next_station` 함수에서 사용할 수 없다.

## Example

다음 호출을 생각해보자.

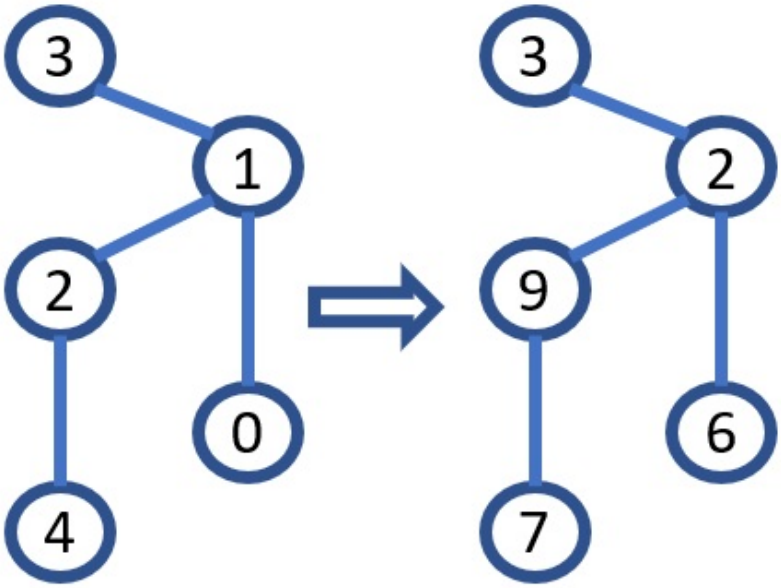
```
label(5, 10, [0, 1, 1, 2], [1, 2, 3, 4])
```

모두 $5$개의 기지국이 있고, $(0, 1)$, $(1, 2)$, $(1, 3)$, $(2, 4)$를 잇는 $4$개의 링크가 있다. 각 레이블은 $0$ 이상 $k = 10$ 이하인 정수이다.

다음과 같이 레이블을 매긴 것을 리턴하기 위해서는

| Index | Label |
|---|---|
| 0 | 6 |
| 1 | 2 |
| 2 | 9 |
| 3 | 3 |
| 4 | 7 |

`label` 함수는 $[6, 2, 9, 3, 7]$을 리턴한다. 다음 그림의 숫자들은, 왼쪽은 인덱스를, 오른쪽은 각 기지국에 할당된 레이블을 의미한다.




위와 같이 레이블이 할당되었을 때 다음 호출을 생각해보자.

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

이는 패킷이 레이블 $9$에 해당하는 기지국에 있고, 목표 기지국의 레이블은 $6$이라는 뜻이다. 목표까지 경로에 있는 기지국들의 레이블은 $[9, 2, 6]$이다. 따라서, 이 함수의 리턴값은 $2$라야 하는데, 패킷이 전송되어야 하는 기지국의 레이블에 해당한다 (이 기지국의 번호는 $1$이다).

또다른 가능한 호출을 생각해보자.

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

이 함수의 리턴값은 $3$이라야 하는데, 레이블이 $3$인 목표 기지국은 레이블이 $2$인 기지국의 이웃이고, 따라서 바로 패킷을 보내줄 수 있기 때문이다.

## Constraints

- $1 \leq r \leq 10$

`label`의 각 호출에 대해서:

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

`find_next_station`의 각 호출에 대해서, 입력은 이전의 `label` 호출 중 임의로 고른 하나에서 선택한다. 이 호출에서 만든 레이블을 생각해보자. 그러면,

- $s$와 $t$는 두 서로 다른 기지국의 레이블이다.
- $c$는 레이블이 $s$인 기지국의 모든 이웃들의 레이블의 서열이며, 오름차순으로 정렬되어 있다.

각각의 테스트 케이스에서, 모든 시나리오를 다 합쳐서 `find_next_station` 함수로 넘겨지는 모든 배열 $c$의 길이의 총합은 $100\ 000$을 넘지 않는다.

## Subtasks

1. (5 points) $k = 1000$, 이웃이 $2$개 보다 많은 기지국은 없다.
2. (8 points) $k = 1000$, 링크 $i$는 기지국 $i + 1$와 기지국 $\left\lfloor \frac{i}{2} \right\rfloor$를 잇는다
3. (16 points) $k = 1\ 000\ 000$, 이웃이 $2$개 보다 많은 기지국은 최대 한 개.
4. (10 points) $n \leq 8$, $k = 10^9$
5. (61 points) $k = 10^9$

서브태스크 5에서 부분 점수를 받을 수 있다. $m$이 모든 시나리오를 통틀어서 `label` 함수가 리턴한 가장 큰 레이블 값이라고 하자. 이 서브태스크에서 여러분의 점수는 다음 표에 의해 계산된다.

| Maximum label | Score |
|---|---|
| $m \geq 10^9$ | $0$ |
| $2000 \leq m < 10^9$ | $50 \cdot \log_{5 \cdot 10^5}(\frac{10^9}{m})$ |
| $1000 < m < 2000$ | $50$ |
| $m \leq 1000$ | $61$ |

## Sample grader

샘플 그레이더는 다음 양식으로 입력을 읽는다.

- line 1: $r$

다음 $r$개의 블럭이 따라오고, 각각의 블럭은 시나리오 하나를 기술한다. 각 블럭의 양식은 다음과 같다.

- line $1$: $n$ $k$
- line $2+i$ ($0 \le i \le n-2$): $u[i]$ $v[i]$
- line $1+n$: $q$: `find_next_station`의 호출 횟수.
- line $2+n+j$ ($0 \le j \le q-1$): $z[j]$ $y[j]$ $w[j]$: $j$ 번째로 `find_next_station` 함수를 호출하였을 때 연관된 기지국들의 **번호**. 기지국 $z[j]$에 현재 패킷이 있고, 기지국 $y[j]$가 패킷의 목표 기지국이며, 기지국 $w[j]$가 패킷이 전송되여야 하는 기지국이다.

샘플 그레이더는 다음 양식으로 출력한다.

- line $1$: $m$

입력된 $r$개의 시나리오의 순서대로 해당하는 $r$개의 블럭을 출력한다. 각 블럭의 양식은 다음과 같다.

- line $1+j$ ($0 \le j \le q-1$): 이 시나리오에서 `find_next_station`의 $j$번째 호출에서 리턴되는 **레이블**의 기지국의 **번호**

샘플 그레이더가 매번 실행될 때마다 `label`과 `find_next_station`을 모두 호출하는데 유의하시오.