# 3-3. 로카히아 유적

고대 로카히아 문명은 인류 역사상 유일하게 마법을 사용했던 문명이다. 지금도 아르키미아 평야에는 로카히아 문명의 흔적을 찾아볼 수 있다. 아르키미아 평야에는 $N$개의 로카히아 유적이 남아있으며 두 개의 유적을 연결하는 $N-1$개의 통로가 있다. 유적의 번호는 각각 $0, 1, \cdots, N-1$이며 통로의 번호는 각각 $0, 1, \cdots, N-2$이다. $i$번 $(0 \le i \le N-2)$ 통로는 $S[i]$번 유적과 $E[i]$번 유적을 양방향으로 연결하며 $N$개의 유적들은 통로를 통해 모두 연결되어 있다.

탐험을 좋아하는 모험가 제운이는 로카히아 문명을 탐험하다가 들판에서 제단을 발견하였다. 성취감을 느낀 제운이가 제단에 발을 올리는 순간 제운이는 제단의 저주에 걸리게 되었다. 저주에 의해 제운이는 유적에서 오직 통로를 통해서만 이동할 수 있다. 제운이는 저주를 풀기 위하여 유물을 모으려고 한다.

각 유적에는 정확히 하나의 유물이 있다. $i$번 $(0 \le i \le N-2)$ 유적에 있는 유물의 번호는 $i$번이다. 제운이는 유적에 있는 유물을 자유롭게 가져갈 수 있다. 유적이 총 $N$개 있으므로 제운이는 최대 $N$개의 유물을 가져갈 수 있다. 제운이가 $r$개의 유물을 가져갔다면 제운이의 마력은 $r$가 된다. $i$번 $(0 \le i \le N-2)$ 통로에는 미지의 안개가 끼어있어서 제운이의 마력이 $M[i]$ 이상일 때만 지나갈 수 있다.

제운이는 유적 하나를 자유롭게 선택하여 해당 유적에서 탐험을 시작할 수 있다. 제운이는 이번 탐험을 통해 $\lfloor N/2 \rfloor$개보다 많은 유물을 모으려고 하기에, 어느 유적에서 탐험을 시작하는지는 상당히 중요하다. 제운이는 사전 답사를 하여 $R1$번 유물과 $R2$번 유물을 동시에 가져갈 수 있는지 체크한 다음에 본격적으로 유물을 모으러 갈 것이다. 제운이를 도와 사전 답사를 어떻게 해야 하는지 구하는 프로그램을 작성하여라.

## 요구 사항

다음 함수를 구현해야 한다. 각 테스트 케이스에 대해 그레이더는 이 함수를 한 번 호출한다.

```
int FindBase(int N)
```

- $N$: 유적의 수.
- 이 함수에서는 제운이가 $\lfloor N/2 \rfloor$개보다 많은 유물을 모으기 위해 탐험을 시작해야 할 유적의 번호를 반환한다. 만약 답이 여러 개이면 그 중에 아무거나 반환한다. 만약 유물을 많이 모을 수 없다면 -1을 반환한다.
- 이 함수는 한 번 호출된다.

FindBase 함수 안에서 아래 함수를 호출할 수 있다.

```
int CollectRelics(int R1, int R2)
```

- $R1$: 첫 번째 유물의 번호 ($0 \leq R1 \leq N-1$).
- $R2$: 두 번째 유물의 번호 ($0 \leq R2 \leq N-1$, $R1 \neq R2$).
- 이 함수는 $R1$번 유물과 $R2$번 유물을 모두 모으기 위해 탐험을 시작해야 할 유적의 번호를 반환한다. 해당 유적이 여러 개이면 그 중에 최소 번호를 반환한다. 만약 두 유물을 모두 모을 수 없다면 -1을 반환한다.
- 이 함수는 최대 600번 호출할 수 있다.

위 조건을 만족하지 않거나 FindBase의 반환값이 올바르지 않으면, 당신의 프로그램은 Wrong Answer으로 채점된다. 나머지 경우에는 당신의 프로그램은 Accepted으로 채점된다.

## 제한

- $1 \leq N \leq 200$
- 모든 $0 \leq i \leq N-2$에 대해,
  - $0 \leq S[i] \leq N-1$
  - $0 \leq E[i] \leq N-1$
  - $S[i] \neq E[i]$
  - $1 \leq M[i] \leq N$
- 유적들은 통로를 통해 모두 연결되어 있다.

이 문제에서 그레이더는 적응적이지 않다 (NOT adaptive). 이것은 $S$, $E$, $M$이 그레이더의 수행 초기에 고정되어서 당신이 요청하는 쿼리에 따라 바뀌지 않는다는 것을 의미한다.

## 부분문제

1. (100점) 추가 제약 조건은 없다.

당신의 프로그램이 Accepted로 채점되었다면, 당신의 점수는 CollectRelics를 호출한 횟수 $C$에 의해 계산한다. 계산 방법은 아래와 같다.

- $C \leq 300$ 이면, $P = 100$.
- $301 \leq C \leq 400$ 이면, $P = 77$.
- $401 \leq C \leq 500$ 이면, $P = 51$.
- $501 \leq C \leq 600$ 이면, $P = 39$.
- $C > 600$ 이면, $P = 0$.

당신의 점수는 가장 낮은 점수를 받은 테스트 케이스의 점수이다 .

## 예제

유적의 구조가 다음과 같다고 해보자.

```
0-1: 2
1-3: 1
1-2: 4
2-4: 1
```

다음 호출이 발생한다.

```
FindBase(5)
```

`FindBase` 함수 안에서 다음과 같은 함수 호출을 할 수 있다.

```
CollectRelics(0, 2)
CollectRelics(1, 3)
CollectRelics(2, 4)
```

함수 호출의 반환값은 각각 -1, 1, 2이다.

`FindBase` 함수는 1 또는 3을 반환해야 한다.

## 샘플 인터페이스

문제 페이지에서 샘플 코드를 다운로드받을 수 있다. 만약 Visual Studio나 Eclipse, Code::Blocks와 같은 IDE 툴을 사용한다면 `lokahia.cpp`, `lokahia.h`, `grader.cpp`를 한 프로젝트에 넣어서 컴파일하면 된다. 터미널에서 코드를 컴파일한다면 대회 페이지에 있는 컴파일 명령어를 이용하면 된다.

답안을 제출할 때에는 `lokahia.cpp`를 제출하면 된다.

### Input format

- line 1: $N$
- line $2 + i$ ($0 \le i \le M - 2$): $S[i]$ $E[i]$ $M[i]$

### Output format

당신의 프로그램이 `Accepted`으로 채점되었다면, 샘플 그레이더는 첫 번째 줄에 `Correct`를 출력하고 두 번째 줄에 `CollectRelics` 함수의 호출 횟수를 출력한다.

당신의 프로그램이 `Wrong Answer`으로 채점되었다면, 샘플 그레이더는 첫 번째 줄에 에러 메시지를 출력한다.