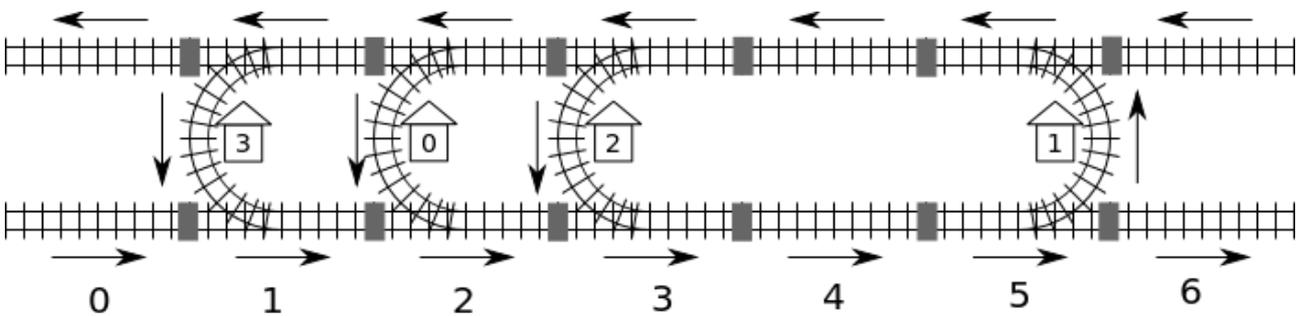




철로

타이완에는 섬의 서쪽과 동쪽 해안을 연결하는 철도선이 존재한다. 이것은 m 개의 블록들로 이루어져 있다. 연속된 블록들은 서쪽(그림에서 왼쪽) 끝에서부터 $0, \dots, m - 1$ 과 같이 번호 붙여진다. 각 블록은 북쪽에 서쪽방향으로만 움직일수 있는 트랙과 남쪽에 동쪽방향으로만 움직일수 있는 트랙이 있고, 일부 블록에는 북쪽과 남쪽 트랙 사이에 역이 존재한다.

블록들은 세가지 타입으로 구분된다. 타입 C 블록에는 북쪽 트랙으로부터 들어와서 남쪽 트랙으로 나가는 역이 존재한다. 타입 D 블록에는 남쪽 트랙으로부터 들어와서 북쪽 트랙으로 나가는 역이 존재한다. 타입 empty 블록에는 역이 존재하지 않는다. 예를 들어, 아래 그림에서 블록 0, 4, 6은 타입 empty, 블록 1, 2, 3은 타입 C, 그리고 블록 5는 타입 D이다. 인접한 블록의 트랙들은 아래 그림의 칠해진 사각형으로 표시된 커넥터로 연결된다.



철로 시스템에는 0부터 $n - 1$ 까지 숫자로 표현하는 n 개의 역들이 존재한다. 트랙을 따라서 임의의 역에서 임의의 다른 역으로 갈수 있다고 가정한다. 예를 들어, 다음과 같이 역 0에서 역 2로 움직일 수 있다: 블록 2에서 시작해서 남쪽 트랙을 따라서 블록 3과 4를 지나고 역 1을 통과해서 블록 5를 지나고 북쪽 트랙으로 블록 4를 지나서 블록 3에서 역 2에 도착한다.

한 역에서 다른 역으로 가는 여러 경로들이 존재하기 때문에 두 역 간의 거리는 경로가 지나는 커넥터들의 최소 수로 정의한다. 예를 들어, 역 0에서 역 2로의 최소 길이 경로는 블록 2-3-4-5-4-3이고 5개의 커넥터들을 통과함으로 거리는 5이다.

철로 시스템은 컴퓨터로 관리된다. 불행히도 정전후에 컴퓨터는 역이 어디에 있는지 어떤 타입의 블록 안에 있는지 더이상 알 수 없게 되었다. 단지 주어진 힌트는 역 0은 항상 타입 C 블록 안에 있다는 것과 역 0가 속한 블록의 번호를 알수 있다. 다행히 컴퓨터는 임의의 역에서 임의의 다른 역까지의 거리를 질의할 수 있다. 예를 들어, 컴퓨터는 '역 0에서 역 2까지의 거리가 무엇인가?'와 같은 질의를 할 수 있고 답 5를 받을 수 있다.

문제

각 역에 대해서 이 역이 속한 블록의 번호와 타입을 결정하는 함수 findLocation을 구현하시오.

■ findLocation(n , first, location, stype)

■ n : 역들의 수.

- first: 역 0이 속한 블록의 번호.
- location: 크기 n 의 배열; 역 i 가 속한 블록의 번호를 $location[i]$ 에 저장한다.
- stype: 크기 n 의 배열; 역 i 가 속한 블록의 타입을 $stype[i]$ 에 저장한다: 타입 C에 대해 1, 타입 D에 대해 2.

역들의 위치와 타입을 구하기 위해서 함수 `getDistance`를 호출할 수 있다.

- `getDistance(i, j)` 역 i 에서 역 j 까지의 거리를 리턴한다. `getDistance(i, i)`는 0을 리턴한다. i 또는 j 가 범위 $0 \leq i, j \leq n - 1$ 을 벗어나면 -1을 리턴한다.

부분 문제

모든 부분 문제에서 블록들의 수 m 은 1,000,000을 넘을 수 없다. 어떤 부분 문제에서는 `getDistance`의 호출 수는 제한된다. 제한은 부분 문제마다 다를 수 있다. 여러분의 프로그램이 이 제한을 넘기면 'wrong answer'를 받게될 것이다.

부분 문제	점수	n	<code>getDistance</code> 호출횟수	주의
1	8	$1 \leq n \leq 100$	무제한	0을 제외한 모든 역들은 타입 D 블록안에 있다.
2	22	$1 \leq n \leq 100$	무제한	역 0의 동쪽의 모든 역들은 타입 D 블록 안에 있고 역 0의 서쪽의 모든 역들은 타입 C 블록 안에 있다.
3	26	$1 \leq n \leq 5,000$	$n(n - 1)/2$	추가적 제한 없음
4	44	$1 \leq n \leq 5,000$	$3(n - 1)$	추가적 제한 없음

구현 내용

여러분은 `rail.c`, `rail.cpp` 또는 `rail.pas`라는 정확히 하나의 파일을 제출해야 한다. 이 파일은 다음 선언들을 사용해서 위에서 설명한 바와 같은 함수 `findLocation`를 구현한다. 또한 여러분은 C/C++ 구현을 위해 헤더 파일 `rail.h`를 `#include`해야 한다.

C/C++ program

```
void findLocation(int n, int first, int location[], int stype[]);
```

Pascal program

```
procedure findLocation(n, first : longint; var location,
  stype : array of longint);
```

`getDistance`의 선언들은 다음과 같다.

C/C++ program

```
int getDistance(int i, int j);
```

Pascal program

```
function getDistance(i, j: longint): longint;
```

Sample grader

Sample grader는 다음과 같은 형식으로 입력을 읽는다:

- line 1: 부분 문제 번호
- line 2: n
- line $3 + i$, ($0 \leq i \leq n - 1$): $stype[i]$ (타입 **C**에 대해 1, 타입 **D**에 대해 2), $location[i]$.

`findLocation`이 리턴할 때 여러분의 프로그램이 계산한 $location[0] \dots location[n-1]$ 과 $stype[0] \dots stype[n-1]$ 이 입력과 일치하면 **sample grader**는 **Correct**를 출력한다. 그렇지 않고 일치하지 않으면 **Incorrect**를 출력한다.