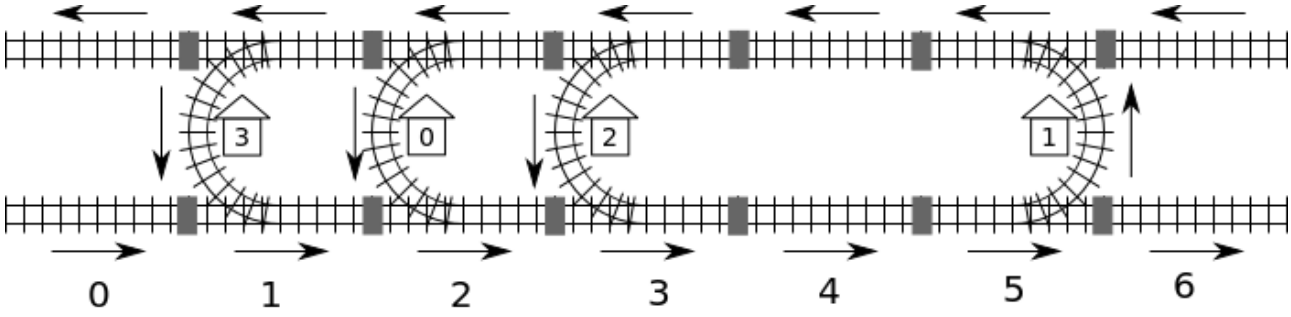
Rail

Taiwan has a big railway line connecting the western and eastern shores of the island. The line consists of m blocks. The consecutive blocks are numbered $0, \ldots, m-1$, starting from the western end. Each block has a one-way west-bound track on the north, a one-way east-bound track on the south, and optionally a train station between them.

There are three types of blocks. A type *C* block has a train station that you must enter from the northern track and exit to the southern track, a type *D* block has a train station that you must enter from the southern track and exit to the northern track, and a type *empty* block has no train station. For example, in the following figure blocks 0, 4 and 6 are type empty, blocks 1, 2 and 3 are type C, and block 5 is type D. Tracks of adjacent blocks are joined by *connectors*, shown as shaded rectangles in the following figure.



The rail system has n stations numbered from 0 to $n-1$. We assume that *we can go from any station to any other station* by following the track. For example we can go from station 0 to station 2 by starting from block 2, then passing through blocks 3 and 4 by the southern track, and then passing block 5 through station 1, then passing through block 4 by the northern track, and finally reaching station 2 at block 3.

Since there are multiple possible routes, the distance from one station to another is defined as the *minimum* number of connectors the route passes through. For example the shortest route from station 0 to 2 is through blocks 2-3-4-5-4-3 and passes through 5 connectors, so the distance is 5.

A computer system manages the rail system. Unfortunately after a power outage the computer no longer knows where the stations are and what types of blocks they are in. The only clue the computer has is the block number of station 0, which is always in a type C block. Fortunately the computer can query the distance from any station to any other station. For example, the computer can query 'what is the distance from station 0 to station 2?' and it will receive 5.

Task

You need to implement a function `findLocation` that determines for each station the block number and block type.

- `findLocation(n, first, location, stype)`
  - `n`: the number of stations.
  - `first`: the block number of station 0.
  - `location`: array of size n; you should place the block number of station i into `location[i]`.
  - `stype`: array of size n; you should place the block type of station i into `stype[i]`: 1 for type C and 2 for type D.

You can call a function `getDistance` to help you find the locations and types of stations.

- `getDistance(i, j)` returns the distance from station `i` to station `j`. `getDistance(i, i)` will return 0. `getDistance(i, j)` will return -1 if `i` or `j` is outside the range $0 \le i, j \le n - 1$.

Subtasks

In all subtasks the number of blocks m is no more than 1,000,000. In some subtasks the number of calls to `getDistance` is limited. The limit varies by subtask. Your program will receive 'wrong answer' if it exceeds this limit.

subtask	points	n	getDistance calls	note
1	8	$1 \le n \le 100$	unlimited	All stations except 0 are in type D blocks.
2	22	$1 \le n \le 100$	unlimited	All stations to the east of station 0 are in type D blocks, and all stations to the west of station 0 are in type C blocks.
3	26	$1 \le n \le 5,000$	$n(n-1)/2$	no additional limits
4	44	$1 \le n \le 5,000$	$3(n-1)$	no additional limits

Implementation details

You have to submit exactly one file, called `rail.c`, `rail.cpp` or `rail.pas`. This file implements `findLocation` as described above using the following signatures. You also need to include a header file `rail.h` for C/C++ implementation.

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);
```

The signatures of `getDistance` are as follows.

C/C++ program

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

Pascal program

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

Sample grader

The sample grader reads the input in the following format:

- line 1: the subtask number
- line 2: `n`
- line $3+i$, ($0 \le i \le n-1$): `stype[i]` (1 for type C and 2 for type D), `location[i]`.

The sample grader will print `Correct` if `location[0]` ... `location[n-1]` and `stype[0]` ... `stype[n-1]` computed by your program match the input when `findLocation` returns, or `Incorrect` if they do not match.