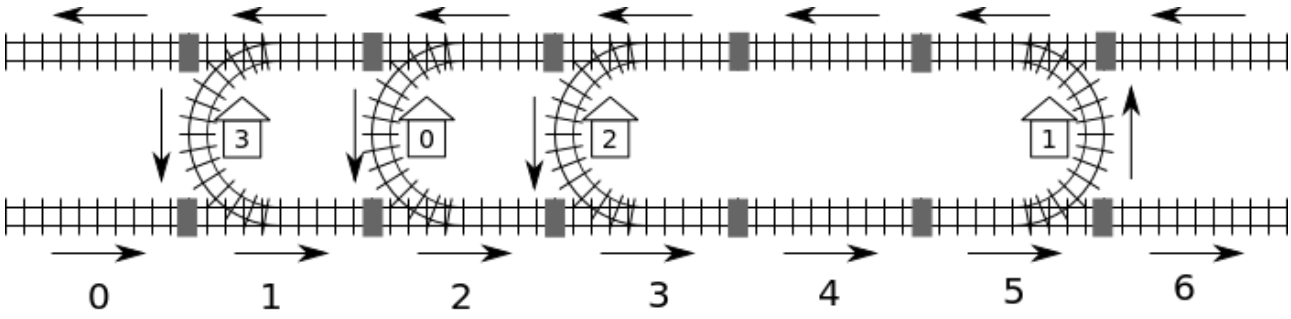
Dəmiryolu (Rail)

Tayvanda adanın qərb və şərq sahilini birləşdirən böyük dəmiryol xətti vardır. Xətt m blokdan ibarətdir. Bloklar ardıcıllığı qərb sahildən başlayaraq $0, \ldots, m-1$ kimi nömrələnib. Hər blokun şimalda bir qərb-istiqamətli xətti, cənunda şərq-istiqamətli xətti var və onların arasında stansiya ola da bilər, olmaya da.

Üç növ blok var. *C* növ blokda qatar stansiyası var və siz ora şimaldakı xətdən daxil olmalı, cənubdakı xətdən isə çıxmalısınız. *D* növ blokda qatar stansiyası var və siz ora cənubdakı xətdən daxil olmalı, şimaldakı xətdən isə çıxmalısınız. *Boş* növ blokda isə stansiya yoxdur. Məsələn, aşağıdakı şəkildə 0 bloku boşdur, 1 bloku C növdür, 5 bloku isə D növdür. Bloklar bir-birilə üfüqi birləşib. Qonşu blokların xətləri bir-birinə *bağlayıcılarla* bağlanıblar (şəkildə boz düzbucaqlılarla göstərilib).



Dəmiryolu sistemi n stansiyadan ibarətdir və onlar 0-dan to $n-1$-dək nömrələnib. Hesab edirik ki, hər hansı yolla *istənilən stansiyadan istənilən başqa stansiyaya* gedə bilərik. Məsələn, 0 stansiyasından 2 stansiyasına getmək üçün 2 blokundan başlamaq, cənub xətti ilə 3 və 4 bloklarından keçmək, sonra 1 stansiyasından keçmək, sonra şimal xətti ilə 4 blokundan keçmək və nəhayət, 3 blokundakı 2 stansiyasına çatmaq olar.

Mümkün marşrutların sayı çox olduğundan bir stansiyadan o birinədək məsafə onlar arasındakı bağlayıcıların *minimal* sayı ilə müəyyən olunur. Məsələn, 0 və 2 stansiyaları arasında ən qısa marşrut 2-3-4-5-4-3 bloklarından (5 bağlayıcıdan) keçir, deməli, məsafə 5-ə bərabərdir.

Kompüter sistemi dəmiryol şəbəkəsini idarə edir. Təəssüf ki, cərəyan kəsildikdə kompüter stansiyaların harada olduğunu və hansı növ bloklarda olduğunu unudur. Kompüter üçün yeganə ipucu 0 stansiyasının blokunun nömrəsidir və o, həmişə C növ blokda olur. Xoşbəxtlikdən, kompüter hər hansı stansiyadan istənilən başqa stansiyadək məsafəni soruşa bilər. Məsələn, kompüter '0 və 2 stansiyaları arasında məsafə nəyə bərabərdir?' sualını verə və 5 cavabını ala bilər.

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 right of station 0 are in type D blocks, and all stations to the left 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.