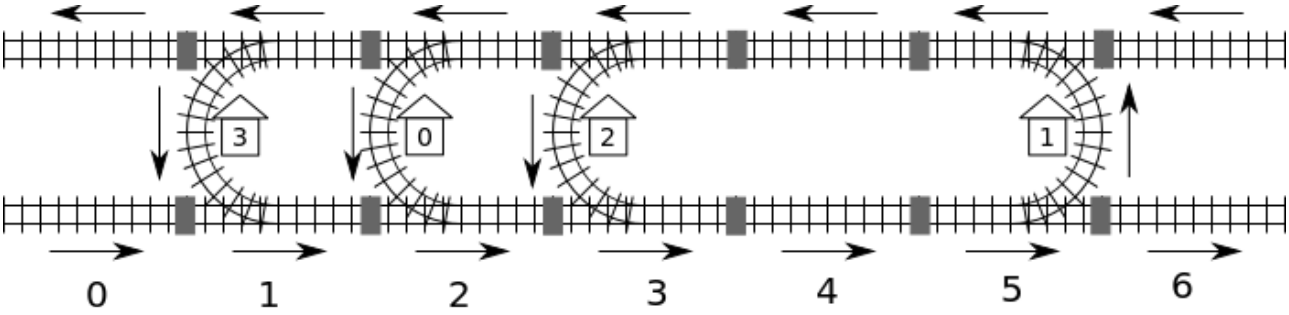
Željeznica

Zapadnu i istočnu obalu Tajvana povezuje duga željeznička pruga koja se sastoji od m blokova. Blokovi su označeni rednim brojevima $0, 1, \ldots, m-1$ od zapada prema istoku. Svaki blok sastoji se od dvije jednosmjerne trake. Sjeverna traka vodi prema zapadu, a južna prema istoku. U sredini bloka, između traka, može se nalaziti i željeznička postaja.

Razlikujemo tri vrste blokova s obzirom na položaj postaje: *C* blokove, *D* blokove i *prazne* blokove. U *C* bloku nalazi se željeznička postaja u koju se ulazi sa sjeverne strane, a izlazi se na južnu stranu. *D* blok je obrnut, dakle, u njemu se nalazi postaja u koju se ulazi sa južne strane, a izlazi se na sjevernu stranu. U praznim blokovima nema postaje. Na donjoj slici blok 0 je prazan, blok 1 je *C* blok, a blok 5 je *D* blok. Susjedni blokovi su povezani *spojnicama* koje su na donjoj slici prikazane osjenčanim pravokutnicima.



Postoji n stanica označenih rednim brojevima $0, 1, \ldots, n-1$. Možete pretpostaviti da se *od svake postaje vlakom može doći do svake druge postaje*. Na primjer, od postaje 0 do postaje 2 može se doći tako da krenemo iz bloka 2, prođemo kroz blok 3 i 4 po južnoj traci pruge, zatim kroz stanicu 1 dođemo na sjevernu stranu pruge pa po toj strani prođemo kroz blok 4 i konačno dođemo u blok 3 u postaju broj 2.

Udaljenost od postaje A do postaje B definira se kao *najmanji mogući* prijeđeni broj spojnica na nekoj ruti od A do B. Na primjer, najkraći put od postaje 0 do postaje 2 je kroz blokove 2-3-4-5-4-3 i prolazi kroz 5 spojnica pa je udaljenost 5.

Računalni sustav nadzire željeznicu. Nažalost, zbog nestanka struje i gubitka informacija, računalo više ne zna gdje su postaje niti u kojim vrstama blokova se nalaze. Jedino što je ostalo zapamćeno je informacija u kojem bloku se nalazi postaja 0 i informacija da je to *C* blok. Na svu sreću, računalo može upitati za udaljenost od neke postaje do neke druge postaje. Na primjer, može upitati "Koja je udaljenost od postaje 0 do postaje 2?" i dobit će odgovor 5.

Zadatak

Implementirajte funkciju `findLocation` koja za svaku postaju određuje redni broj i vrstu bloka u kojem se nalazi.

- `findLocation(n, first, location, stype)`
  - `n`: broj postaja
  - `first`: redni broj bloka stanice 0
  - `location`: niz veličine n; trebate zapisati redni broj bloka stanice i u `location[i]`.
  - `stype`: niz veličine n; trebate zapisati vrstu bloka stanice i u `stype[i]`: 1 za *C* blok, a 2 za *D* blok.

Možete pozivati funkciju `getDistance` kako bi odredili redne brojeve i vrste blokova.

- `getDistance(i, j)` vraća udaljenost od stanice `i` do stanice `j`. `getDistance(i, i)` vratit će 0. `getDistance(i, j)` vratit će -1 ako ne vrijedi $0 \leq i, j \leq n-1$.

Podzadaci

U svim podzadacima, broj blokova m neće biti veći od $1,000,000$. U nekim podzadacima broj poziva funkcije `getDistance` je ograničen. Ograničenje za broj poziva ovisi o podzadatku. Vaš program dobit će *'wrong answer'* ako prekorači to ograničenje.

podzadatak	broj bodova	n	ograničenje za getDistance	napomena
1	8	$1 \leq n \leq 100$	neograničeno	Sve stanice osim stanice 0 su u *D* blokovima.
2	22	$1 \leq n \leq 100$	neograničeno	Sve stanice istočno od stanice 0 su u *D* blokovima, a sve stanice zapadno od stanice 0 su u *C* blokovima.
3	26	$1 \leq n \leq 5,000$	$n(n-1)/2$	nema dodatnih ograničenja
4	44	$1 \leq n \leq 5,000$	$3(n-1)$	nema dodatnih ograničenja

Implementacijski detalji

Morate *submitati* točno jednu datoteku, `rail.c`, `rail.cpp` ili `rail.pas`. U toj datoteci moraju biti implementirane gore navedene funkcije.

C/C++ program

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

Paskal program

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

Prototipi funkcije `getDistance` su sljedeći:

C/C++ program

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

Paskal program

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

Sample grader

Sample grader prima ulaz sljedećeg oblika:

- 1. linija: redni broj podzadatka
- 2. linija: `n`
- linije $3+i, 4+i, \ldots, 2+n$: `stype[i]` (1 za *C* blok, a 2 za *D* blok), `location[i]`.

Sample grader ispisat će `Correct` ako `location[0]`, ..., `location[n-1]` i `stype[0]`, ..., `stype[n-1]` koje vraća vaša `findLocation` funkcija odgovaraju ulaznim podacima, a `Incorrect` ako ne odgovaraju.