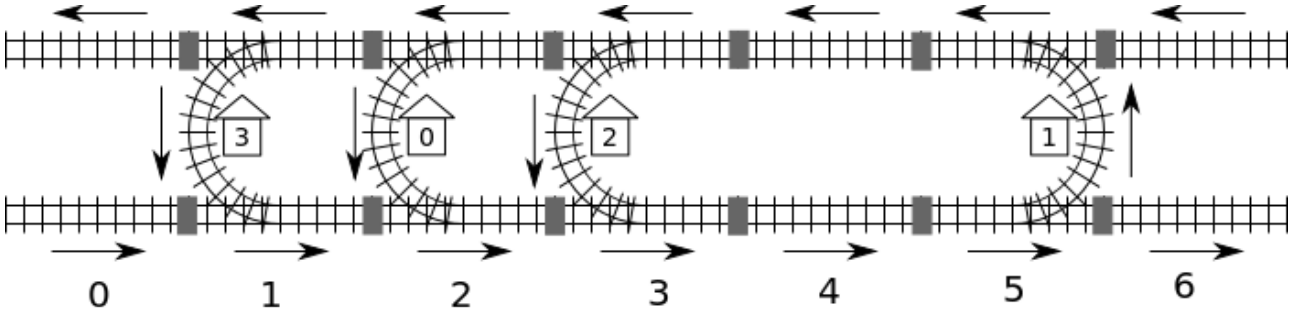
Željeznice

Taiwan posjeduje razvijenu mrežu pruga koja povezuje zapadne i istočne obale ostrva. Linija se sastoji od m blokova. Uzastopni blokovi su numerisani sa $0,\ldots,m-1$, počevši od zapadnog kraja. Svaki blok se sastoji od dva jednosmjerna kolosijeka. Sjeverni kolosijek vodi prema zapadu, a južna prema istoku. U sredini bloka, između traka može se nalaziti željeznička stanica.

Postoje ukupno tri tipa blokova. Blok tipa *C* ima željezničku stanicu u koju morate ući sa sjeverne strane i izaći na južnu stranu, blok tipa *D* ima željezničku stanicu u koju morate ući sa južne strane i izaći na sjevernu stranu, dok blok tipa *prazan* nema nikakvih stanica. Na primjer, na sljedećoj slici blok 0 je tipa *prazan*, blok 1 je tipa *C*, a blok 5 je tipa *D*. Blokovi su horizontalno povezani medju sobom. Kolosijeci susjednih blokova su povezani *spojnicama* koji su prikazani kao osjenčeni pravougaonici na istoj slici.



Cijeli željeznički sistem ima ukupno n stanica numerisanih od 0 do $n-1$. Možemo pretpostaviti da *je moguće ići od bilo koje stanice do bilo koje stanice* datim kolosijecima. Na primjer, možemo ići od stanice 0 do stanice 2 počevši sa blokom 2, pa prolazeći blokove 3 i 4 južnim kolosijekom, prolazeći zatim stanicu 1, pa onda prolazeći blok 4 sjevernim kolosijekom, i konačno doći u stanicu 2 u bloku 3.

Kako ima više mogućih puteve, udaljenost od jedne do druge stanice je definisana kao *minimalan* broj konektora kroz koje neki put prolazi. Na primjer, najkraći put od stanice 0 do stanice 2 prolazi blokovima 2-3-4-5-4-3, drugim rječima prolazi kroz 5 konektora, pa je udaljenost jednaka 5.

Kompjuterski sistem ima zadatak da upravlja ovim željezničkim sistemom. Na žalost nakon jednog prekida struje kompjuter ne poznaje više gdje se nalaze stanice i kakvi tipovi blokova se nalaze na njima. Jedinu pomoć koju kompjuter ima je broj bloka na kojem je stanica 0, i taj blok je uvijek tipa *C*. Srećom, kompjuter može ispitivati udaljenosti od bilo koje stanice do bilo koje stanice. Na primjer, kompjuter može pitati 'koja je udaljenost od stanice 0 do stanice 2?' i dobiće kao odgovor 5.

Zadatak

Vaš zadatak je da implementirate funkciju `findLocation` koja za svaku stanicu odredjuje broj i tip bloka.

- `findLocation(n, first, location, stype)`

- `n`: broj stanica.
- `first`: broj bloka stanice 0.
- `location`: niz dužine n; vi treba da upišete broj bloka stanice i u `location[i]`.
- `stype`: niz dužine n; vi treba da upišete tip bloka stanice i u `stype[i]`: 1 za tip C i 2 za tip D.

Možete pozvati funkciju `getDistance` koja vam može pomoći da nadjete lokacije i tipove stanica.

- `getDistance(i, j)` vraća udaljenost od stanice `i` do stanice `j`. `getDistance(i, i)` vraća 0. `getDistance(i, j)` vraća -1 ako su `i` ili `j` van područja $0 \leq i, j \leq n-1$.

Podzadaci

U svim podzadacima broj blokova m nije veći od 1,000,000. U nekim podzadacima ukupan broj poziva funkcije `getDistance` je ograničen. Ovo ograničenje se mijenja u zavisnosti od podzadatka. Vaš program će biti evaluiran kao 'wrong answer' ako predje ovo ograničenje.

podzadatak	poeni	n	getDistance pozivi	komentar
1	8	$1 \leq n \leq 100$	neograničeno	Sve stanice osim stanice 0 su u blokovima tipa D.
2	22	$1 \leq n \leq 100$	neograničeno	Sve stanice sa desne strane stanice 0 su u blokovima tipa D, a sve stanice sa lijeve strane stanice 0 su u blokovima tipa C.
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

Detalji o implementaciji

Treba da submitujete tačno jedan file, pod nazivom `rail.c`, `rail.cpp` ili `rail.pas`. Ovaj file implementira funkciju `findLocation` na gore opisani način. Potrebno je takodje da dodate header file `rail.h` za C/C++ implementaciju.

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

Grader čita ulazne podatke date u sljedećem formatu:

- line 1: the subtask number
- line 2: `n`
- line $3 + i$, ($0 \leq i \leq n - 1$): `stype[i]` (1 za tip C i 2 za tip D), `location[i]`.

Grader će odštampati `Correct` ako su vrijednosti `location[0]` ... `location[n-1]` i `stype[0]` ... `stype[n-1]` koje je izračunala vaša procedura `findLocation` iste kao i ulazni podaci, ili `Incorrect` ukoliko ima nekih razlika.