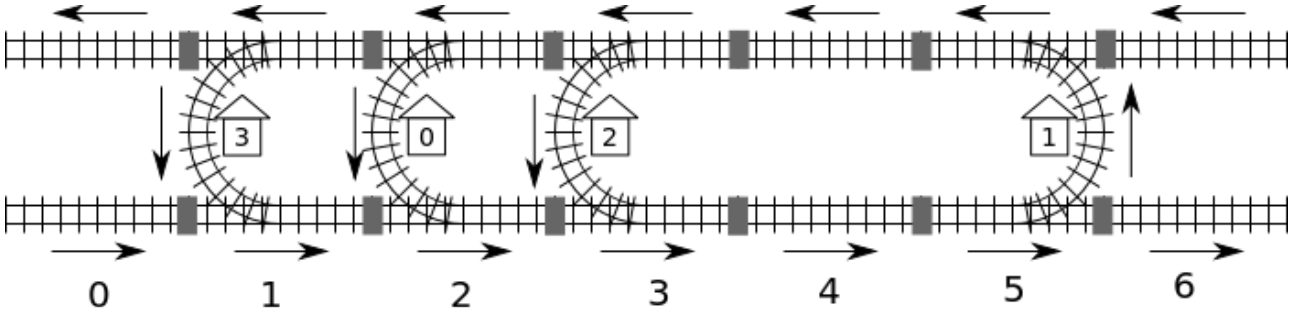
Vasút

Taiwan vasútvonala összeköti a nyugati és a keleti partot, m blokkból áll, az egymást követő blokkokat a $0, \ldots, m-1$ számokkal azonosítjuk, az első a legnyugatibb. Minden blokk északi része kelet-nyugati irányban egyirányú, a déli része pedig nyugat-keleti irányban, valamint lehet közöttük egyirányú összekötő szakasz, amin van állomás.

Háromféle blokk van. Közülük kettő összekötő szakaszt tartalmaz. A *C* típusú szakaszon csak északról délre lehet haladni az állomáson keresztül, a *D* típusú szakaszon pedig csak délről északra. A harmadik, *üres* blokkban nincs összekötő szakasz és állomás. Az ábrán például a 0. blokk üres, az 1. blokk *C*, az 5. blokk pedig *D* típusú. Az egymás melletti blokkokat szürke téglalapok kapcsolják össze.



A vonalon n állomás van, 0-tól $n-1$-ig sorszámozva. Feltehető, hogy *bármely állomásról bármely másik állomásra el lehet jutni* a vonalon. Például a 0. állomásról a 2. állomásra a 2. blokkból kell indulni, a déli oldalon át kell haladni a 3. és a 4. blokkon, utána az 1. állomáson áthaladva északi oldalra jutunk, ahol átmegyünk a 4. blokkon, és végül elérjük a 2. állomást a 3. blokkban.

Két állomás távolsága a közöttük levő útvonalon az összekötő szürke téglalapok minimális száma. Például a legrövidebb út a 0. és a 2. állomás között a 2-3-4-5-4-3 blokkok sorozata, amely 5 összekötőn halad át, azaz a távolság 5.

Csak azt tudjuk, hogy a 0. állomás melyik blokkban van és az *C* típusú. Megkérdezhetjük bármely két állomás távolságát.

Feladat

A `findLocation` függvényt kell megírnod, amely megadja minden állomás blokk sorszámát és típusát!

- `findLocation(n, first, location, stype)`
  - `n`: az állomások száma.
  - `first`: a 0. állomás blokk sorszáma
  - `location`: n elemű tömb; eredményül a `location[i]`-be kell tenned az i. állomás

blokk sorszámát!

- `stype`: n elemű tömb; eredményül az `stype[i]`-be kell tenned az i. állomás blokk típusát: 1-et *C* típusúnál és 2-t *D* típusnál!

Két állomás távolságának lekérdezésére a `getDistance` függvényt használhatod.

- `getDistance(i, j)` megadja az `i`. és a `j`. állomás távolságát, `getDistance(i, i)` értéke 0 lesz ($0 \leq i, j \leq n-1$). `getDistance(i, j)` -1 lesz, ha `i` vagy `j` tartományon kívüli érték.

Részfeladatok

A tesztekben a blokkok m száma nem nagyobb 1,000,000-nál. Néhány részfeladatban a `getDistance` hívások száma korlátozott. A programodra 'wrong answer' hibaüzenetet kapsz, ha ezt a korlátot túlléped.

részfeladat	pont	n	getDistance hívások	megjegyzés
1	8	$1 \leq n \leq 100$	korlátlan	A 0.-at kivéve minden állomás blokkja *D* típusú.
2	22	$1 \leq n \leq 100$	korlátlan	A 0. állomástól jobbra levő állomások *D*, a balra levők *C* típusú blokkban vannak.
3	26	$1 \leq n \leq 5,000$	$n(n-1)/2$	nincs más korlát
4	44	$1 \leq n \leq 5,000$	$3(n-1)$	nincs más korlát

Megvalósítás

A `rail.c`, `rail.cpp` vagy `rail.pas` fájlt kell beküldened! Ebben kell megvalósítanod a `findLocation` függvényt! C/C++ esetén include-olnod kell a `rail.h`-t!

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);
```

A `getDistance` deklarációja:

C/C++ program

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

Pascal program

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

Minta értékelő

A minta értékelő a bemenetet a következő formában olvassa:

- 1. sor: a részfeladat sorszáma
- 2. sor: `n`
- $3+i$. sor ($0 \leq i \leq n-1$): `stype[i]` (1 *C* típusú blokknál és 2 *D* típusú blokknál), `location[i]`.

A mintaértékelő a `Correct` választ írja ki, ha `location[0]` ... `location[n-1]` és `stype[0]` ... `stype[n-1]` megfelel a bemenetnek, egyébként az `Incorrect` választ.