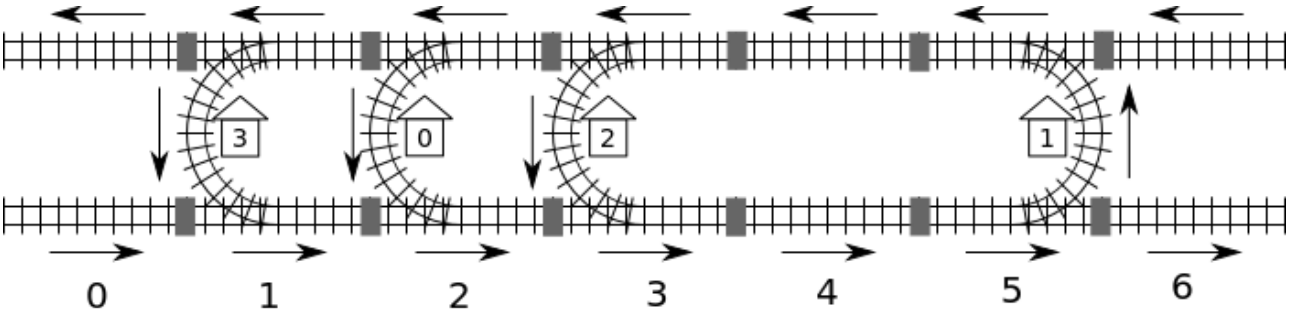
Rail (Vías)

Taiwan tiene una vía de tren que conecta la costa oeste y este de la isla. La vía está formada por m bloques. Los bloques están numerados de forma consecutiva $0, \ldots, m-1$, iniciando del oeste hacia el este. Cada bloque tiene una vía en su lado norte que va en dirección oeste y una en su lado sur que va en dirección este. Opcionalmente un bloque puede tener una estación de tren entre la vía del norte y la del sur.

Existen tres tipos de bloques. Los bloques tipo *C* tienen una estación de tren en la que el tren entra por la vía del norte y sale por la vía del sur. Los bloques tipo *D* tienen una estación en la que el tren entra por la vía del sur y sale por la vía del norte. Por último, los bloques de tipo *vacío* no tienen estación de tren. Por ejemplo, en la siguiente imagen, los bloques 0, 4 y 6 son de tipo *vacío*, los bloques 1, 2 y 3 son de tipo *C* y el bloque 5 es de tipo *D*. Los bloques se conectan entre ellos horizontalmente. Las vías de dos bloques contiguos están unidas por *conectores*, representados por rectángulos grises en la siguiente imagen:



El sistema de vías tiene n estaciones numeradas de 0 a $n-1$. Asumimos que *es posible viajar de cualquier estación a cualquier otra estación* a través de las vías de tren. Por ejemplo, se puede viajar de la estación 0 a la estación 2. Iniciando del bloque 2, se pasa a través de los bloques 3 y 4 por la vía del sur, se cruza a la vía del norte por medio de la estación 1 en el bloque 5, luego se pasa al bloque 4 por la vía del norte y finalmente se llega a la estación 2 desde el bloque 3.

Dado que pueden existir múltiples rutas, la distancia de una estación a otra es definida como la *mínima* cantidad de *conectores* por los que debe pasar el tren. Por ejemplo. la ruta más corta de la estación 0 a la 2 es a través de los bloques 2-3-4-5-4-3 y pasa en total por 5 *conectores*, por lo tanto la distancia es 5.

Una computadora maneja todo el sistema de vías. Desafortunadamente, después de un problema eléctrico la computadora ya no conoce la ubicación de las estaciones ni en qué tipo de bloques se encuentran las estaciones. La única pista que la computadora sabe es el número del bloque donde se encuentra la estación 0, que siempre es un bloque de tipo *C*. Afortunadamente, la computadora puede responder la distancia de cualquier estación a cualquier otra estación. Por ejemplo, la computadora puede responder la pregunta: '¿cuál es la distancia de la estación 0 a la estación 2?' y la computadora responderá 5.

Problema

Debes implementar la función `findLocation` que encuentre para cada estación su número de bloque y qué tipo de bloque es.

- `findLocation(n, first, location, stype)`
  - `n`: el número de estaciones.
  - `first`: el número de bloque donde se encuentra la estación 0.
  - `location`: arreglo de tamaño n; debes poner el número de la estación i en `location[i]`.
  - `stype`: arreglo de tamaño n; debes poner el tipo de bloque de la estación i en `stype[i]`: 1 para bloques tipo *C* y 2 para bloques de tipo *D*.

Puedes llamar la función `getDistance` que te ayudará a encontrar el tipo y número de bloque de cada estación.

- `getDistance(i, j)` regresa la distancia de la estación `i` a la estación `j`. `getDistance(i, i)` regresará 0. `getDistance(i, j)` regresará -1 si `i` o `j` están fuera del rango $0 \leq i, j \leq n-1$.

Subproblemas

En todos los subproblemas, el número de bloques m nunca será más de 1,000,000. En algunos subproblemas está limitada la cantidad de llamadas que se pueden hacer a `getDistance`. El limite varía por subproblema. Tu programa obtendrá 'wrong answer' si el límite es excedido.

subproblema	puntos	n	llamadas a getDistance	notas
1	8	$1 \leq n \leq 100$	ilimitado	Todas las estaciones, excepto la 0, son de tipo *D*.
2	22	$1 \leq n \leq 100$	ilimitado	Todas las estaciones al este de la estación 0 son de tipo *D* y todas las estaciones al oeste de la estación 0 son de tipo *C*.
3	26	$1 \leq n \leq 5,000$	$n(n-1)/2$	No hay límites adicionales.
4	44	$1 \leq n \leq 5,000$	$3(n-1)$	No hay límites adicionales.

Detalles de implementación

Debes subir exctamente un archivo, llamado `rail.c`, `rail.cpp` o `rail.pas`. Este archivo debe tener implementado `findLocation` como fue descrito anteriormente. También necesitas incluir el archivo `rail.h` para implementaciones en C/C++.

Programa en C/C++

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

Programa en Pascal

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

El prototipo `getDistance` es de la siguiente manera.

Programa en C/C++

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

Programa en Pascal

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

Evaluador de ejemplo

El evaluador de ejemplo lee la entrada de la siguiente manera:

- linea 1: el número de subproblema
- linea 2: `n`
- linea $3 + i$, ($0 \leq i \leq n - 1$): `stype[i]` (1 para tipo *C* y 2 para tipo *D*), `location[i]`.

El evaluador de ejemplo imprimirá `Correct` si los `location[0]` ... `location[n-1]` y `stype[0]` ... `stype[n-1]` que calculaste son los correctos cuando `findLocation` termina de ejecutarse, o `Incorrect` si no coinciden.