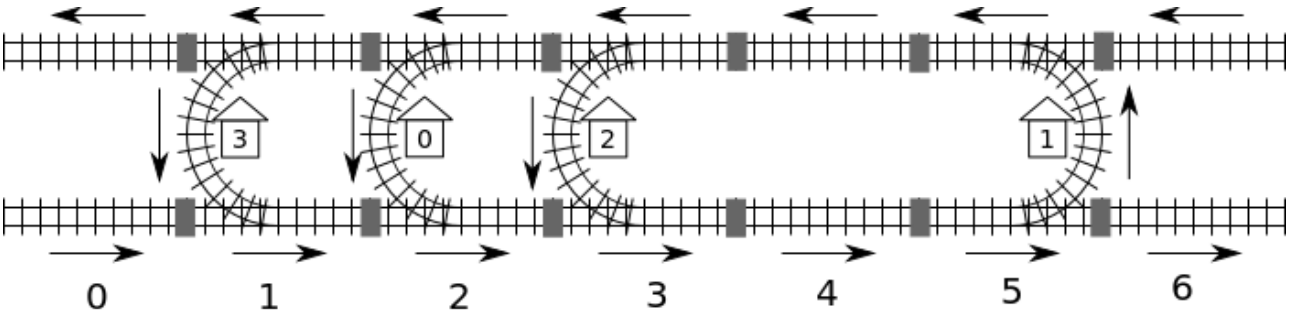
Rail

Taiwán tiene un sistema de trenes que conecta la costa oeste con la costa este de la isla. El sistema está dividido en m bloques numerados $0, .., m-1$ comenzando desde el lado oeste. Cada bloque tiene una linea en dirección oeste y una linea en dirección este. La linea en dirección oeste está ubicada al norte del bloque (arriba), y la linea en dirección este está ubicada al sur del bloque (abajo). Cada bloque puede o no tener una estación de trenes ubicada entre las lineas norte y sur (que se muestran como casas en el dibujo).

Hay tres tipos de bloques dependiendo de si tiene o no una estación y la dirección en la que se accede a esa estación. Un bloque de tipo *C* tiene una estación a la que se debe entrar desde el norte y salir hacia el sur. Un bloque de tipo *D* tiene una estación a la que se debe entrar desde el sur y salir hacia el norte. Finalmente un bloque *vacío* es uno que no tiene estación. Por ejemplo, en la siguiente figura los bloques 0, 4 y 6 están *vacíos*, los bloques 1, 2 y 3 son de tipo *C*, y el bloque 5 es de tipo *D*. Para moverse horizontalmente entre bloques, se debe pasar por un *conector* (que une pistas de bloques adyacentes), mostrados como rectángulos sombreados en la figura.



Hay n estaciones en el sistema de trenes numeradas desde 0 hasta $n-1$. Asumimos que *siempre es posible ir desde cada estación a cualquier otra estación*. Por ejemplo, en la figura, podemos ir desde la estación 0 hasta la estación 2 partiendo en el bloque 2, pasando por los bloques 3 y 4 usando la linea sur, luego cambiándose de la linea sur a la norte en el bloque 5 (pasando por la estación 1), y finalmente pasando a través del bloque 4 usando la linea norte, para llegar al bloque 3 donde se encuentra ubicada la estación 2.

Dado que hay muchas rutas posibles, la distancia entre estaciones está definida como el menor número de conectores por los que hay que pasar. Por ejemplo, la ruta más corta desde la estación 0 a la estación 2 pasa por los bloques 2-3-4-5-4-3 a través de 5 conectores, por lo que la distancia es 5.

El sistema computacional que maneja los trenes ha fallado y ya no sabe en qué bloque se encuentra cada estación ni de qué tipo es este bloque. La única información con que cuenta es el número del bloque donde se encuentra la estación 0, y que este bloque es siempre de tipo *C*. Afortunadamente el sistema computacional puede preguntar por la distancia necesaria para ir de cada estación a cualquier otra. Por ejemplo, puede preguntar "¿cuál es la distancia para ir de la estación 0 a la estación 2?" y recibirá como respuesta '5'.

Tarea

Debes implementar la función `findLocation` que determina para cada estación el número y tipo del bloque en el que se encuentra.

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

Debes llamar a la función `getDistance` para determinar la posición y tipos de las estaciones.

- `getDistance(i, j)` retorna la distancia para ir del bloque `i` al bloque `j`. `getDistance(i, i)` retorna siempre 0. `getDistance(i,j)` retorna -1 si `i` o `j` están fuera del rango $0 \leq i, j \leq n-1$.

Subtareas

En cada subtarea la cantidad m de bloques será menor o igual a 1,000,000. En algunas subtareas la cantidad de llamadas a `getDistance` está limitada. El límite varía según subtareas y tu programa recibirá 'wrong answer' si excede ese límite.

Subtarea	puntos	n	llamadas a getDistance	Nota
1	8	$1 \leq n \leq 100$	sin límite	Todas las estaciones excepto la estación 0 están en un bloque de tipo *D*.
2	22	$1 \leq n \leq 100$	sin límite	Todas las estaciones al este de la estación 0 están en bloques de tipo *D* y todas las estaciones al oeste están en bloques de tipo *C*.
3	26	$1 \leq n \leq 5,000$	$n(n-1)/2$	sin restricciones adicionales
4	44	$1 \leq n \leq 5,000$	$3(n-1)$	sin restricciones adicionales

Detalles de implementación

Debes enviar exactamente un archivo llamado `rail.c`, `rail.cpp` o `rail.pas`. Este archivo implementa `findLocation` como se describió arriba, y usando el siguiente prototipo. Además debes incluir `rail.h` para C/C++.

Programa C/C++

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

Programa Pascal

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

El prototipo de `getDistance` es el siguiente.

Programa C/C++

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

Programa Pascal

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

Calificador ejemplo

El *calificador* de ejemplo lee el input en el siguiente formato:

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

El *calificador* imprimirá `Correct` si `location[0]`,...,`location[n-1]` y `stype[0]`,..., `stype[n-1]` entregados por tu programa coincide con el input cuando `findLocation` retorna, o `Incorrect` en caso contrario.