<!--

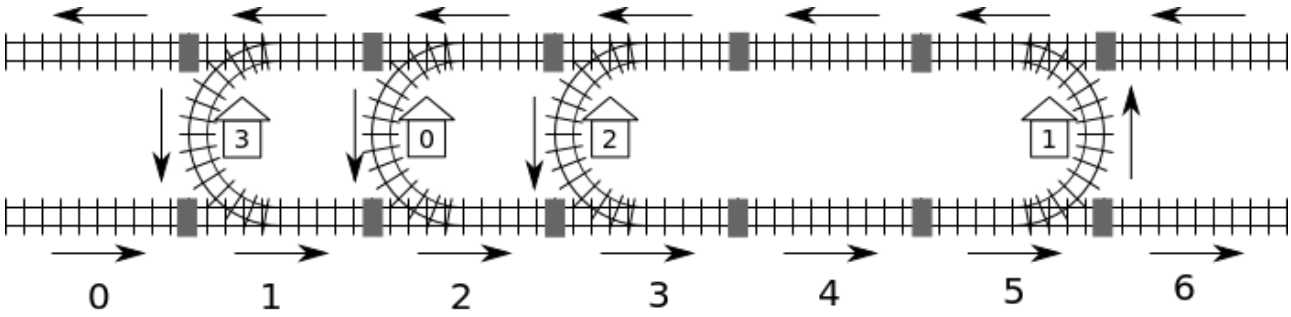
Rail

-->

鉄道（Rail）

台湾には，西海岸と東海岸を結ぶ大きな鉄道路線がある．
路線は m ブロックからなり，ブロックは西から順番に 0, ..., $m-1$ の番号がついている．
各ブロックには，北側を西方向に進める線路と，南側を東方向に進める線路がある．
さらにいくつかのブロックには，北側の線路と南側の線路の間に駅がある．

ブロックには，次の 3 つのタイプがある．
タイプ *C* のブロックには駅があり，駅には北側の線路から入って南側の線路へ出なければならない．
タイプ *D* のブロックには駅があり，駅には南側の線路から入って北側の線路へ出なければならない．
タイプ *E* のブロックには駅がない．
例えば下図では，
ブロック 0, 4, 6 はタイプ E,
ブロック 1, 2, 3 はタイプ C,
ブロック 5 はタイプ D である．
隣接するブロックの線路は *コネクター* で繋がっており，下図では影付きの長方形で表されている．



この鉄道には 0 から $n - 1$ の番号がついた n 個の駅がある．
ここで，*どの駅からどの駅へも線路を辿って移動できるものとする*．
例えば，駅 0 から駅 2 へは，
ブロック 2 から始めて，ブロック 3 とブロック 4 の南側の線路を通り，ブロック 5 で駅 1 を通り，ブロック 4 の北側の線路を通り，ブロック 3 にある駅 2 に到着す

る，
というように移動できる．

ある駅から他のある駅への距離は，
移動ルートがいくつか考えられるうちでの，通過するコネクターの個数の *最小値* と定義する．
例えば，駅 0 から駅 2 への最短ルートは，2-3-4-5-4-3 とブロックを通るものであり，5 個のコネクターを通過するので，距離は 5 である．

鉄道はコンピューターシステムに管理されている．
不幸にも停電が発生してしまい，コンピューターは駅がどこにあったか，またブロックがどのタイプであったかがわからなくなってしまった．
今コンピューターがわかっている手がかりは，駅 0 があるブロックの番号と，そのブロックはタイプ C であるという事実だけである．
幸い，コンピューターは「ある駅から他のある駅への距離はいくつか？」というクエリを出すことができる．
例えば，コンピューターが「駅 0 から駅 2 への距離はいくつか？」というクエリを出すと，5 という答えを受け取ることになる．

<!--

Task

-->

課題（Task）

あなたは，各駅があるブロックの番号とそのブロックのタイプを決定する関数 `findLocation` を実装しなければならない．

<!--

- `findLocation(n, first, location, stype)`
  - `n`: the number of stations.
  - `first`: the block number of station 0.
  - `location`: array of size n; you should place the block number of station i into `location[i]`.
  - `stype`: array of size n; you should place the block type of station i into `stype[i]`: 1 for type C and 2 for type D.
    -->
- `findLocation(n, first, location, stype)`
  - `n`: 駅の個数.
  - `first`: 駅 0 があるブロックの番号.

- location: サイズ n の配列であり，あなたは，駅 i があるブロックの番号を location[i] に入れなければならない．
- stype: サイズ n の配列であり，あなたは，駅 i があるブロックのタイプを stype[i] に入れなければならない．ただし，タイプ C なら 1，タイプ D なら 2 を入れよ．

あなたは，駅の位置やブロックのタイプを考えるために，関数 getDistance を呼ぶことができる．

<!--

- getDistance(i, j) returns the distance from station i to station j. getDistance(i, i) will return 0. getDistance(i, j) will return -1 if i or j is outside the range $0 ¥leq i, j ¥leq n - 1$.
  -->
- getDistance(i, j) は駅 i から駅 j への距離を返す．getDistance(i, i) は 0 を返す．i または j が $0 ¥leq i, j ¥leq n - 1$ の範囲外にあるとき，getDistance(i, j) は -1 を返す．

<!--

Subtasks

-->

小課題（Subtasks）

すべての小課題で，ブロックの個数 m は 1,000,000 以下である．
いくつかの小課題では，getDistance の呼び出し回数が制限される．
その回数は小課題によって異なる．
回数制限を超えた場合は，'wrong answer' と判定される．

小課題	得点	n	getDistance の呼び出し回数	追加の制約
1	8	$1 ¥leq n ¥leq 100$	無制限	駅 0 以外のすべての駅はタイプ D のブロックにある．
2	22	$1 ¥leq n ¥leq 100$	無制限	駅 0 より東にあるすべての駅はタイプ D のブロックにあり，駅 0 より西にあるすべての駅はタイプ C のブロックにある．
3	26	$1 ¥leq n ¥leq 5,000$	$n (n-1) / 2$	追加の制約なし
4	44	$1 ¥leq n ¥leq 5,000$	$3 (n - 1)$	追加の制約なし

<!--

Implementation details

-->

実装の詳細（Implementation details）

あなたは，rail.c，rail.cpp または rail.pas という名前のファイルをちょうど 1 つ提出しなければならない．
そのファイルは，上記のような findLocation を以下のシグネチャに従って実装する．
さらに，C/C++ の場合は，rail.h を include する必要がある．

<!--

C/C++ program

-->

C/C++ プログラム

<!--

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

-->

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

<!--

Pascal program

-->

Pascal プログラム

<!--

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

-->

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

getDistance のシグネチャは以下の通りである．

<!--

C/C++ program

-->

C/C++ プログラム

<!--

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

-->

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

<!--

Pascal program

-->

Pascal プログラム

<!--

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

-->

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

<!--

Sample grader

-->

採点プログラムのサンプル (Sample grader)

採点プログラムのサンプルは，以下の形式で入力を読む：

<!--

- line 1: the subtask number
- line 2: n
- line $3 + i$, ($0 ¥leq i ¥leq n - 1$): stype[i] (1 for type C and 2 for type D), location[i].
  -->
- 1 行目： 小課題の番号
- 2 行目： n
- $3 + i$ 行目 ($0 ¥leq i ¥leq n - 1$)： stype[i]（タイプ C なら 1，タイプ D なら 2）と location[i].

採点プログラムのサンプルは，
findLocation が終了したときに location[0] ... location[n-1] と stype[0] ... stype[n-1] が入力と一致していれば Correct と出力し，
そうでなければ Incorrect と出力する．