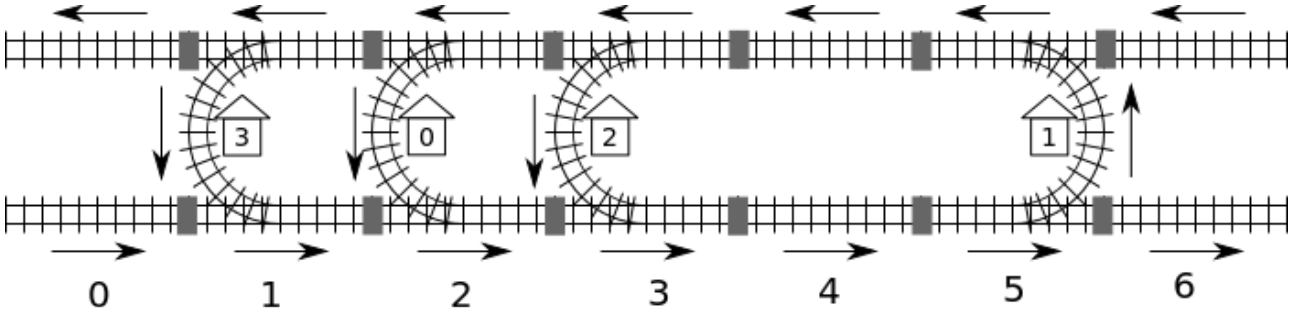
Rail

Taiwan tem uma grande linha de caminho de ferro que liga a costa oeste à costa este da ilha. A linha consiste em m blocos. Os blocos consecutivos estão numerados como $0, \ldots, m-1$, a começar na costa oeste. Cada bloco tem um trilho de este para oeste ao norte, um trilho de este para oeste ao sul e opcionalmente uma estação de comboios (trens) entre eles.

Existem três tipos de blocos. Um bloco do tipo *C* tem uma estação de comboio onde se entra pelo trilho ao norte e se sai pelo trilho ao sul; um bloco do tipo *D* tem uma estação de comboio onde se entra pelo trilho ao sul e se sai pelo trilho ao norte; e um bloco do tipo *vazio* que não tem nenhuma estação de comboio. Por exemplo, na figura seguinte os blocos 0, 4 e 6 são do tipo vazio, os bloco 1, 2 e 3 são do tipo C e o bloco 5 é do tipo D. Trilhos de blocos adjacentes estão ligados por *conectores* que estão mostrados como retângulos sombreados na figura seguinte.



O sistema de caminhos de ferro tem n estações numeradas de 0 a $n-1$. Assume-se que *é possível ir de qualquer estação para qualquer outra estação* ao seguir os trilhos. Por exemplo, é possível ir da estação 0 para a estação 2 ao começar no bloco 2, passando nos blocos 3 e 4 pelo trilho ao sul depois passando no bloco 5 pela estação 1, posteriormente passando no bloco 4 pelo trilho ao norte e finalmente chegando à estação 2 no bloco 3.

Visto que há vários caminhos possíveis, a distância de uma estação para outra é definida como o número *mínimo* de conectores que o caminho atravessa. Por exemplo, o caminho mais curto da estação 0 para a 2 é pelos blocos 2-3-4-5-4-3 e passa por 5 conectores, por isso a distância é 5.

Um sistema informático gere o sistema de caminhos de ferro. Infelizmente, devido a uma falha de energia, o sistema informático perdeu a informação sobre onde está cada estação e em que tipo de bloco está. A única informação que reteve é o bloco da estação número 0 e que esse é sempre um bloco do tipo C. Felizmente, o computador consegue perguntar qual é a distância de qualquer estação até qualquer outra estação. Por exemplo, o computador pode perguntar 'qual é a distância da estação 0 até a estação 2?' e receberá a resposta 5.

Tarefa

Você deve implementar uma função `findLocation` que determina para cada estação o número do bloco e o tipo do bloco.

- `findLocation(n, first, location, stype)`
  - `n`: o número de estações.
  - `first`: o número do bloco da estação 0.
  - `location`: um vetor de tamanho n; será necessário colocar o número do bloco da estação i em `location[i]`.
  - `stype`: um vetor de tamanho n; será necessário colocar o tipo do bloco da estação i em `stype[i]`: 1 para o tipo C e 2 para o tipo D.

É possível chamar a função `getDistance` para ajudar a encontrar as localizações e os tipos das estações.

- `getDistance(i, j)` devolve a distância da estação `i` para a estação `j`. `getDistance(i, i)` devolve 0. `getDistance(i, j)` devolve -1 se `i` ou `j` está fora do intervalo $0 \le i, j \le n-1$.

Subtarefas

Em todas as subtarefas o número de blocos m não é maior do que 1,000,000. Nalgumas das subtarefas o número de chamadas a `getDistance` é limitado. O limite varia de subtarefa para subtarefa. O programa receberá 'wrong answer' ('resposta errada') se exceder este limite.

subtarefa	pontos	n	chamadas a getDistance	nota
1	8	$1 \le n \le 100$	ilimitado	todas as estações exceto a 0 estão em blocos do tipo D.
2	22	$1 \le n \le 100$	ilimitado	todas as estações a este da estação 0 estão em blocos do tipo D e todas as estações a oeste da estação 0 estão em blocos do tipo C.
3	26	$1 \le n \le 5,000$	$n(n-1)/2$	não há limites adicionais
4	44	$1 \le n \le 5,000$	$3(n-1)$	não há limites adicionais

Detalhes de Implementação

É preciso submeter exatamente um ficheiro, chamado `rail.c`, `rail.cpp` ou `rail.pas`. Este ficheiro implementa a função `findLocation` como descrito acima usando as assinaturas seguintes. É preciso também incluir um ficheiro de cabeçalho `rail.h` para as implementações em C/C++.

Programa em C/C++

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

Programa em Pascal

```
procedure findLocation(n, first : longint; var location,
```

```
  stype : array of longint);
```

As assinaturas do `getDistance` são as seguintes.

Programa em C/C++

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

Programa em Pascal

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

Avaliador exemplo

O avaliador exemplo lê a entrada no seguinte formato:

- linha 1: o número da subtarefa
- linha 2: `n`
- linhas $3 + i$, ($0 \leq i \leq n - 1$): `stype[i]` (1 para o tipo C e 2 para o D), `location[i]`.

O avaliador exemplo vai imprimir `Correct` se `location[0]` ... `location[n-1]` e `stype[0]` ... `stype[n-1]` calculados pelo programa submetido forem iguais à entrada quando `findLocation` retorna ou `Incorrect` caso contrário.