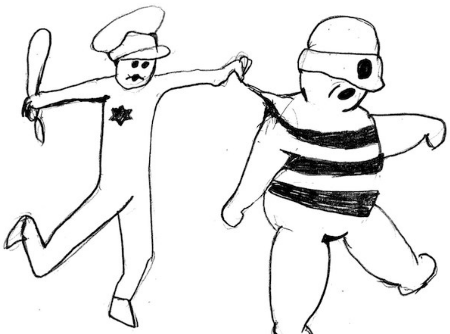
The Cop and the Robber



Bytemoren kaupungissa rikosten määrä on saavuttamassa kaikkien aikojen huipun. Muun rötöstelyn lisäksi ryöstöjä tapahtuu joka päivä. Joka kerta kun ryöstö tapahtuu, on yksittäisen partioivan poliisin tehtävä jahdata ryöstäjä kiinni kadunkulmia yhdistäviä kapeita kujia pitkin. Ikävä kyllä, useimmiten rosvot pääsevät pakoon takaa-ajajiltaan, koska he tuntevat kaupungin paljon poliisia paremmin.

Bytemoren kaupungin poliisilaitos järjestää kokouksen jonka tehtävänä on vähentää rikollisuutta. Yksi aloitteista on käyttää tietokonetta apna rosvojen jahtaamisessa. Tätä varten poliisilaitos on tehnyt tarkan kartan kaupungista. Nyt he tarvitsevat tietokoeohjelmaa jahtausstrategioiden päättämiseen.

Rosvojahti jossa yksi poliisi jahtaa yhtä rosvoa on seuraavanlainen.

1. Poliisi valitsee kadunkulman jossa partioi.
2. Rosvo valitsee kadunkulman jossa ryöstö tehdään (hän tietää aina missä poliisi on). Tästä lähtien oletetaan aina että sekä poliisi että rosvo tietävät missä kumpikin ovat.
3. Poliisi siirtyy siirrollaan joko viereiseen kadunkulmaan (sellaiseen johon kulkee nykyisestä kuja) tai odottaa (ei siirry).
4. Rosvo siirtyy aina vuorollaan viereiseen kadunkulmaan. Huomaa, että toisin kuin poliisit, rosvot eivät voi odottaa paikallaan. Heidän vaistonsa saa heidät jatkamaan juoksua.
5. Poliisi ja rosvo tekevät siirtoja vuorotellen (aloittaen poliisista) kunnes yksi seuraavista tapahtuu:
   (a) tilanne toistuu (tilanne määritellään rosvon ja poliisin sijainteina ja sillä, kenen vuoro on seuraavaksi). Tämä vastaa sitä, että rosvo voi vältellä poliisia loputtomasti, joten rosvo pääsee pakoon;
   (b) poliisi ja rosvo ovat samassa kadunkulmassa kumman tahansa vuoron jälkeen. Tässä tapauksessa poliisi saa rosvon kiinni.

Task

You have to write a program which, given the map of the city, would determine whether catching the robber is possible, and if it is, would catch him by making moves on behalf of the police officer.

Your program must assume that the robber moves optimally.

Implementation

You need to implement two functions:

- `start(N, A)` which takes the following parameters:

- ◦ N — the number of corners (corners are labelled from 0 to $N-1$);
- ◦ A — a two–dimensional array that describes the alleys: for $0 \le i, j \le N-1$,

$$A[i,j] \text{ is } \begin{cases} \texttt{false} & \text{if } i \text{ and } j \text{ are not joined by any alley} \\ \texttt{true} & \text{if } i \text{ and } j \text{ are joined by an alley} \end{cases}$$

  All alleys will be bidirectional (i.e. $A[i,j] = A[j,i]$ for all values of i and j) and there will be no alleys connecting a corner to itself (i.e. $A[i,i]$ will be `false` for all values of i). Also, you may assume that it will always be possible to reach any corner from any other corner by moving along the alleys.

  If it is possible to catch the robber on the map described by the parameters, function `start` should return the label of the corner on which the police officer chooses to patrol. Otherwise, it should return -1.

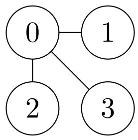
- `nextMove(R)` which takes as a parameter the label R of the current corner of the robber and must return the label of the corner where the officer will be after his move.

Function `start` will be called exactly once before any calls to `nextMove` are made. If `start` returns -1, then `nextMove` will not be called. Otherwise, `nextMove` will be called repeatedly until the pursuit ends. More precisely, the program will terminate as soon as one of the following happens:

- `nextMove` returns an invalid move;
- the situation repeats itself;
- the robber is caught.

Example

Let's take a look at the example illustrated on the right. In this case any corner is a good starting position for the police officer. If he starts in the corner 0, he can wait in his first move and the robber will run into him. On the other hand, if he starts on any other corner, he can wait until the robber moves to corner 0, and then move there.



Here's how a sample session could look like:

Function call	Returns
`start(4, [[0, 1, 1, 1], [1, 0, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0]])`	3
`nextMove(1)`	3
`nextMove(0)`	0

Note: in the call to `start` above `0` zero denotes `false` and `1` denotes `true` for brevity.

Scoring

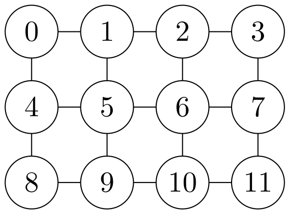
In order to score full points, your solution must:

1. correctly determine whether the police officer can catch the robber;
2. successfully catch the robber by making moves on behalf of the police officer.

However, in subtasks 3 and 4, solutions that only implement the first requirement will score 30% subtask points.

Subtask 1 (15 points): $2 \le N \le 500$. There will be exactly one path between every pair of corners.

Subtask 2 (15 points): $2 \le N \le 500$. The network of corners and alleys will form a grid-shaped structure. The grid will have at least two rows and columns and the street corner labelling will follow the pattern illustrated below.



Subtask 3 (30 points): $2 \le N \le 100$.

Subtask 4 (40 points): $2 \le N \le 500$.

Constraints

Time limit: 1 s.

Memory limit: 256 MB.

Experimentation

The sample grader on your computer will read data from the standard input. The first line of the input should contain integer N — the number of corners. The following N lines should contain the adjacency matrix A. Each of these lines should contain N numbers, where each one is 0 or 1. The matrix must be symmetric and the main diagonal values must all be zeroes.

The next line should contain number 1, if police can catch the robber, and 0 otherwise.

Finally, if police officer can catch the robber, N lines should follow, describing the strategy of the robber. Each of these lines should contain $N+1$ integers between 0 and $N-1$. The value at row r and column c, where $c < N$, corresponds to a situation where it's robber's turn, the police officer is at corner r and the robber is at corner c, and represents the corner, which the robber has to move to. The main diagonal values will be ignored, as they correspond to situations where the robber and the police officer are at the same corner. The right–most column defines robber's starting corners for each possible police officer's starting corner.

Here is an example input to the sample grader which represents three corners that are connected together:

```
3
0 1 1
1 0 1
1 1 0
1
0 2 1 2
2 0 0 2
1 0 0 1
```

And here is the input which matches the example given in the task statement above:

```
4
0 1 1 1
1 0 0 0
1 0 0 0
1 0 0 0
1
0 0 0 0 1
2 0 0 0 2
3 0 0 0 3
1 0 0 0 1
```