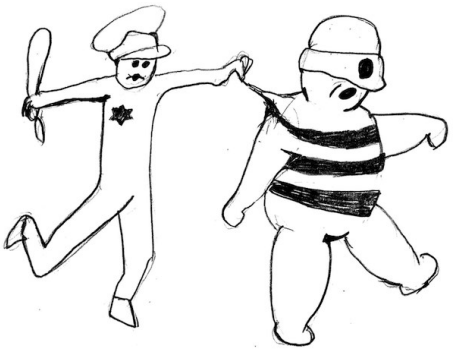
Räuber und Gendarm



In Bytemore City sind die Räuber auf dem Vormarsch! Und nach jedem Raub ist es die Aufgabe eines einsamen Gendarmen, den Räuber durch die engen Straßen der Stadt zu verfolgen, die die Kreuzungen miteinander verbinden. Dummerweise entkommen die Räuber häufig, da sie die Stadt viel besser kennen als die Gendarmen.

Das Bytemore City Police Department (BCPD) will nun Computer einsetzen, um das Verbrechen besser zu bekämpfen. Das BCPD hat dazu bereits einen präzisen Stadtplan von Bytemore City angefertigt. Nun wird noch Software benötigt, die Strategien zur Verfolgung von Räubern ermitteln soll.

In der Software wird die Verfolgung eines Räubers durch einen Gendarmen so modelliert:

1. Der Gendarm wählt eine Kreuzung, von der aus er beginnt.
2. Der Räuber kennt nun die Position des Gendarmen und wählt eine Kreuzung für den Raub. Von da an kennen Räuber und Gendarm gegenseitig ihre Position.
3. Nun ist der Gendarm am Zug. Er kann sich über eine Straße zu einer benachbarten Kreuzung bewegen oder warten (und sich nicht bewegen).
4. Nun ist der Räuber am Zug. Er bewegt sich in jedem Zug über eine Straße zu einer benachbarten Kreuzung. Beachte: Räuber können einfach nicht warten, das Weglaufen gehört zum Berufsethos.
5. Nun ziehen Gendarm und Räuber abwechselnd (der Gendarm beginnt), bis eines der folgenden Dinge passiert:
   (a) Ein Zustand (gegeben durch die Positionen von Räuber und Gendarm und die Angabe, wer als nächster am Zug ist) wiederholt sich. Der Räuber kann also dem Gendarmen beliebig lange aus dem Weg gehen und entkommt so.
   (b) Nach einem Zug befinden sich Räuber und Gendarm auf derselben Kreuzung. In diesem Fall ist der Räuber gefangen.

Aufgabe

Schreibe ein Programm, das bei gegebenem Stadtplan bestimmt, ob der Räuber gefangen werden kann, und, falls ja, die dazu nötigen Züge des Gendarmen angibt.

Dein Programm soll annehmen, dass der Räuber optimale Züge macht.

Implementierung

Du musst zwei Funktionen implementieren:

- `start(N, A)` mit den folgenden Parametern:
  - N – die Anzahl der Kreuzungen (nummeriert von 0 bis $N-1$);
  - A – ein zweidimensionales Array, das die Straßen beschreibt: für $0 \leq i, j \leq N-1$,

$$A[i,j] \text{ ist } \begin{cases} \texttt{false} & \text{falls } i \text{ und } j \text{ nicht durch eine Straße verbunden sind.} \\ \texttt{true} & \text{falls } i \text{ und } j \text{ durch eine Straße verbunden sind.} \end{cases}$$

    Alle Straßen sind in beiden Richtungen begehbar (also $A[i,j] = A[j,i]$ für alle i, j), und es gibt keine "Schleifen" (also: $A[i,i]$ ist `false` für alle i). Außerdem gilt: Für jede Kreuzung gibt es einen Weg zu jeder anderen Kruezung über die Straßen.

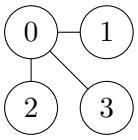
  Falls der Räuber in dem durch den Stadtplan gegebenen Straßennetz gefangen werden kann, soll die Funktion `start` die Nummer der Kreuzung zurückgeben, die der Gendarm als seinen Startpunkt wählen sollte. Sonst soll sie -1 zurückgeben.

- `nextMove(R)` nimmt als Parameter die Nummer R der Kreuzung, auf der sich der Räuber gerade befindet. Sie liefert die Nummer der Kreuzung, auf der sich der Gendarm nach seinem Zug befindet.

Die Funktion `start` wird genau einmal aufgerufen, vor allen Aufrufen von `nextMove`. Liefert `start` -1, dann wird `nextMove` nicht aufgerufen. Ansonsten wird `nextMove` wiederholt aufgerufen, bis die Verfolgung endet. Genauer gesagt: das Programm terminiert, sobald eines der folgenden Dinge passiert:

- `nextMove` liefert einen ungültigen Zug;
- ein Zustand wiederholt sich;
- der Räuber ist gefangen.

Beispiel



Schauen wir uns das Beispiel rechts an. Hier ist jede Kreuzung eine gute Startposition für den Gendarmen. Startet er in Kreuzung 0, kann er im ersten Zug warten; der Räuber muss sich dann zu ihm bewegen. Startet er in einer anderen Kreuzung, kann er warten, bis der Räuber auf Position 0 ist, und sich dann dorthin bewegen.

In diesem Beispiel könnten die Funktionen also wie folgt aufgerufen werden:

Funktionsaufruf	**Liefert**
`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

Bewertung

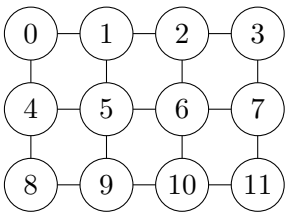
Um volle Punktzahl zu erhalten, muss dein Programm

1. korrekt bestimmen, ob der Gendarm den Räuber fangen kann; und
2. Züge angeben, mit denen der Gendarm den Räuber fangen kann.

In Subtasks 3 und 4 erhalten Lösungen, die nur die erste Anforderung erfüllen, 30% der Punkte.

Teilaufgabe 1 (16 Punkte): $2 \leq N \leq 500$. Zwischen je zwei Kruezungen gibt es genau einen Weg.

Teilaufgabe 2 (14 Punkte): $2 \leq N \leq 500$. Die Kreuzungen und Straßen bilden ein Raster. Das Raster hat mindestens zwei Reihen und Spalten, und die Nummerierung der Kreuzungen folgt dem unten abgebildeten Muster.



Teilaufgabe 3 (30 Punkte): $2 \leq N \leq 100$.

Teilaufgabe 4 (40 Punkte): $2 \leq N \leq 500$.

Constraints

Time Limit: 1,5 s.

Memory Limit: 256 MB.

Experimentieren

Der Beispiel-Grader auf deinem Rechner liest Daten von der Standard-Eingabe. Die erste Zeile der Eingabe sollte aus der Anzahl N der Kreuzungen bestehen. Die folgenden N Zeilen sollten die Adjazenzmatrix A enthalten. Jede dieser Zeilen sollte N Zahlen enthalten, jeweils 0 oder 1. Die Matrix muss symmetrisch sein und die Einträge auf der Hauptdiagonale müssen alle 0 sein.

Die nächste Zeile sollte eine Zahl enthalten: 1, falls die Genarmerie den Räuber fangen kann, andernfalls 0.

Schlussendlich soll im Fall, dass der Gendarm den Räuber fangen kann, sollten N Zeilen folgen, die die Strategie des Räubers beschreiben. Jede Zeile sollte $N + 1$ Zahlen zwischen 0

und $N-1$ einschließlich enthalten. Der Wert in Zeile r und Spalte c wobei $c < N$ beschreibt durch Angabe einer Kreuzungsnummer, wohin der Räuber in der Situation zieht, in der der Räuber am Zug ist, der Gendarm an Kreuzung r und der Räuber an Kreuzung c steht. Die Werte auf der Hauptdiagonale werden ignoriert, da sie Situationen entsprechen, in denen der Räuber bereits verloren hat. Der Eintrag für $c = N$ in jeder Zeile r gibt an, wo der Räuber in Abhängigkeit von der Startposition r des Gendarms startet.

Eine Beispieleingabe für den Beispiel-Grader mit drei Kreuzungen, die jeweils miteinander verbunden sind:

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

Die folgende Eingabe passt zum oben angegebenen Beispiel:

```
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
```