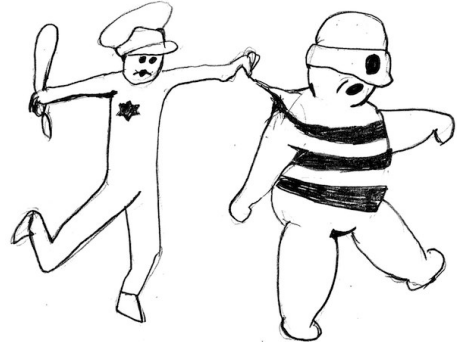


## Cop and Robber

In the city of Bytemore crime level is hitting an all-time high. Among other misdemeanours, robberies are happening every day. And when the crime is committed, it is always up to a lone patrolling police officer to chase down the robber through the narrow alleys that connect street corners (commonly referred to simply as *corners*). Unfortunately, more often than not, robbers escape their pursuers, because they know the city much better than the police.



The Bytemore City Police Department (BCPD) is organising a summit to reduce crime. One of the initiatives is to use computer aid when pursuing the robbers. For this purpose, the BCPD has made a precise map of the city. Now they need computer software to find chasing strategies.

The pursuit problem of one officer chasing one robber is modelled as follows:

1. The police officer chooses a corner on which to patrol.
2. The robber then chooses a corner for the robbery (he knows where the officer is). From this moment on it is always assumed that both the officer and the robber know where each other is.
3. The police officer's move consists of him moving to a neighbouring corner (i.e. one that is connected to the current one by an alley) or waiting (i.e. not moving).
4. The robber's move consists of him moving to a neighbouring corner. Note that, unlike the police, robbers cannot wait. It is in their instinct to keep running.
5. The police officer and the robber keep making moves one after another (starting with the officer) until one of the following happens:
  - (a) situation repeats itself (situation is defined by the positions of both agents and the side whose turn it is to move next). This corresponds to the robber being able to avoid the police officer indefinitely, so the robber escapes;
  - (b) the police officer and the robber meet on the same corner after a move of either of them. In this case the police officer catches the robber.

## 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  $0 \leq i, j \leq N - 1$ ,

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

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`.

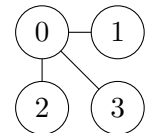
- `nextMove(R)` which takes as a parameter the label  $R$  of the current corner of the robber and must return the label of the corner where the officer will be after his move.

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:

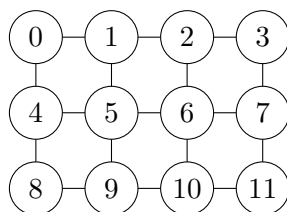
Function call	Returns
<code>start(4, [[0, 1, 1, 1], [1, 0, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0]])</code>	3
<code>nextMove(1)</code>	3
<code>nextMove(0)</code>	0

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

## Scoring

**Subtask 1 (16 points):**  $2 \leq N \leq 500$ . Each pair of corners will be connected by exactly one path of alleys.

**Subtask 2 (14 points):**  $2 \leq N \leq 500$ . 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):**  $2 \leq N \leq 100$ .

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

Your solution should fulfill two requirements:

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, if that is possible.

In subtasks 1 and 2, your solution must implement both requirements to score any points. In subtasks 3 and 4, solutions that only implement the first requirement will score 30% of subtask's points. If your solution only aims for partial score, you can terminate the program by outputting any invalid move (e.g. return  $-1$  in `nextMove`).

Note that the standard requirements (time and memory limits, no runtime errors) must be satisfied to score any points.

## Constraints

**Time limit:** 1.5 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 last value at row  $r$  defines robber's starting corner if the police officer's starting corner is  $r$ .

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
```

## Three Friends

Three friends like to play the following game. The first friend chooses a string  $S$ . Then the second friend constructs a new string  $T$  that consists of two copies of the string  $S$ . Finally, the third friend inserts one letter at the beginning, the end or somewhere inside the string  $T$ , thereby creating a string  $U$ .



### Task

You are given the string  $U$  and your task is to reconstruct the original string  $S$ .

### Input

The first line of the input contains  $N$ , the length of the final string  $U$ . The string  $U$  itself is given on the second line. It consists of  $N$  uppercase English letters (A, B, C, ..., Z).

### Output

Your program should print the original string  $S$ . However, there are two exceptions:

1. If the final string  $U$  could not have been created using the above procedure, you should print NOT POSSIBLE.
2. If the original string  $S$  is not unique, you should print NOT UNIQUE.

### Examples

Input	Output
7 ABXCABC	ABC

Input	Output
6 ABCDEF	NOT POSSIBLE

Input	Output
9 ABABABABA	NOT UNIQUE

### Scoring

**Subtask 1 (35 points):**  $2 \leq N \leq 2001$ .

**Subtask 2 (65 points):**  $2 \leq N \leq 2000001$ .



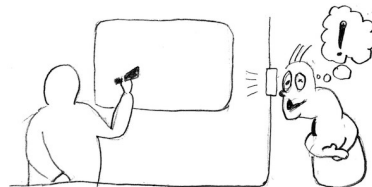
## Constraints

**Time limit:** 0.5 s.

**Memory limit:** 256 MB.

## Sequence

Adam wrote down a sequence of  $K$  consecutive positive integers starting with  $N$  on a blackboard. When he left, Billy came in and erased all but one digit from each number, thus creating a sequence of  $K$  digits.



## Task

Given the final sequence left on the blackboard, find the smallest value of  $N$  with which the initial sequence might have started.

## Input

The first line of the input contains a single integer  $K$  — the length of the sequence. The second line contains  $K$  integers  $B_1, B_2, \dots, B_K$  — Billy's sequence ( $0 \leq B_i \leq 9$ ), in the order in which it is written on the blackboard.

## Output

The output should consist of a single line with the smallest value of  $N$  with which the initial sequence might have started.

## Example

Input	Output	Comments
6 7 8 9 5 1 2	47	$N = 47$ would correspond to Adam's sequence being 47 48 49 50 51 52 from which Billy's sequence can indeed be obtained. As no smaller value of $N$ would work, the answer is 47.

## Scoring

**Subtask 1 (9 points).**  $1 \leq K \leq 1000$ , correct answer does not exceed 1000.

**Subtask 2 (33 points).**  $1 \leq K \leq 1000$ .

**Subtask 3 (25 points).**  $1 \leq K \leq 100\,000$ , all elements of the given sequence are equal.

**Subtask 4 (33 points).**  $1 \leq K \leq 100\,000$ .

## Constraints

Time limit: 1 s.

Memory limit: 256 MB.