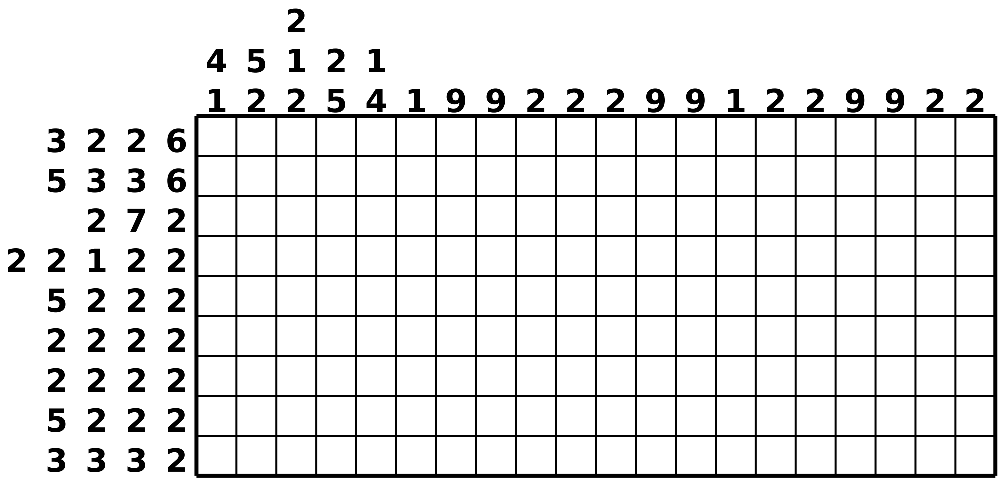
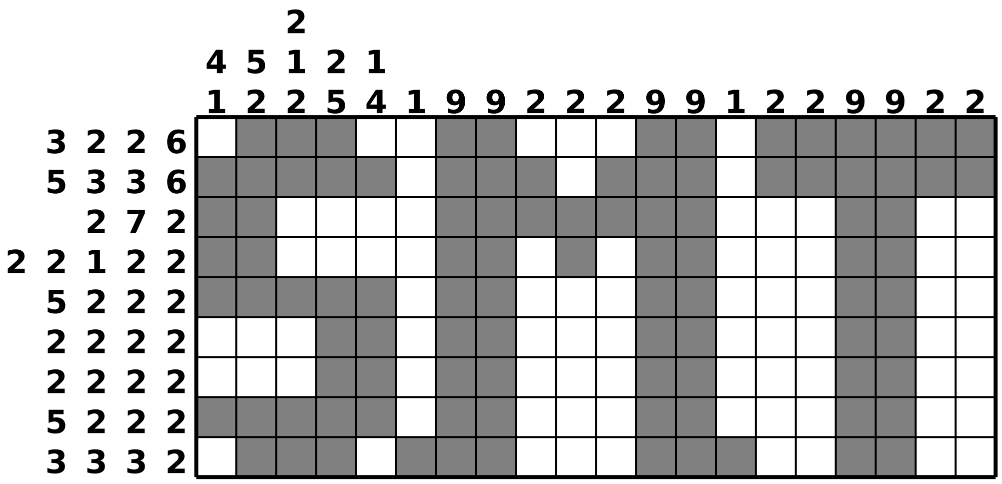
Task: 2
Nonograms

deadline: May 27, 2018, 23:55

A nonogram is a popular puzzle where a hidden picture made of white and black squares is described by numbers. To solve the puzzle, you have to analyze the clues to reveal the hidden picture. An example puzzle is shown below:



Each row and each column is described by a sequence of numbers. Numbers correspond to the lengths of the connected segments of black squares in the given row or column, from top/left to bottom/right. More precisely, $a_1\ a_2\ a_3\ \ldots\ a_k$ means that the content of the given row or column belongs to the regular language $\square^*\blacksquare^{a_1}\square^+\blacksquare^{a_2}\square^+\ldots\square^+\blacksquare^{a_k}\square^*$. The solution to the puzzle above is shown below.



Your task is to solve a nonogram puzzle with the help of an SMT solver. Write a program that reads the description of a puzzle from standard input (in the format described below), translates the problem above to an instance of SMT, runs Z3 over it, and writes to standard output either `unsolvable` (no solution exists), `many solutions` (multiple solutions exist), or the unique solution (in the format described below).

Input

The first line of the input contains two numbers: X (the width of the picture) and Y (the height of the picture).

Each of the following Y lines of input contains a description of a row. A description of a row contains $k+1$ numbers $a_1, \ldots, a_k, 0$ separated by single spaces, representing the numbers describing the given row, and the terminating 0. Each of the following *X* lines contains a description of a column, in the same format. Rows and columns are described from top to bottom and from left to right.

Output

If no solution has been found, the first and only line of the output should contain `unsolvable`. If multiple solutions have been found, the first and only line of the output should contain `many solutions`. If the solution is unique, the output should contain *Y* lines, each of them containing *X* characters `.` (white square) or `#` (black square).

Example

For the input data:

```
20 9
3 2 2 6 0
5 3 3 6 0
2 7 2 0
2 2 1 2 2 0
5 2 2 2 0
2 2 2 2 0
2 2 2 2 0
5 2 2 2 0
3 3 3 2 0
4 1 0
5 2 0
2 1 2 0
2 5 0
1 4 0
1 0
9 0
9 0
2 0
2 0
2 0
9 0
9 0
1 0
2 0
2 0
9 0
9 0
2 0
2 0
```

the correct result is:

```
.###..##...##.######
#####.###.###.######
##....#######...##..
##....##.#.##...##..
#####.##...##...##..
...##.##...##...##..
...##.##...##...##..
#####.##...##...##..
.###.###...###..##..
```

Allowed languages

You are allowed to choose any programming language accepted by the graders. The following languages are accepted: C, C++ (gcc), Java, Python, OCaml, Haskell. Other languages need approval of the grader (Eryk Kopczyński, erykk@mimuw.edu.pl). Please provide all the relevant information needed to compile/run your solution (e.g., for Python, start with a comment stating Python v2 or Python v3; for gcc, include a Makefile so that we know the compiler options used). You can also choose to either use Z3 directly via the API, or to communicate with Z3 via the DIMACS or SMTLIB format.

Test

Moodle includes a script to test whether your solution reads the input and prints the output in the expected format. Make sure that the testing script returns `OK` for your program (named **nonogram-solver**). A solution not conforming to the input/output specification (e.g., by reading from a file) may lose 10% points.

You can include your own inputs with your solution, so that the graders can have fun guessing the hidden picture.