Problem 223: A Matter of 0's and 1's

Difficulty: Hard

Author: Richard Green, Whiteley, Hampshire, United Kingdom

Originally Published: Code Quest 2023

Problem Background

You've probably heard of Sudoku puzzles, and may have even written a program to attempt to solve them. A slightly simpler version uses only 0's and 1's, and so is called Binary Sudoku.

As in Sudoku, the puzzle is made up of a grid of numbers in a square pattern. The challenge is to complete the grid in a way that fulfills a simple set of rules:

- Each box must contain a 0 or a 1.
- The same number may not appear more than twice consecutively within any row or column; for example, you may see 010 or 0110, but 01110 is illegal, as three 1's appear together without a 0 to separate them.
- Each row and column should contain an equal number of 0's and 1's.
- Each row is unique, compared to other rows.
- Each column is unique, compared to other columns.

As with Sudoku, Binary Sudoku boils down to a logic problem; using the rules above to make deductions about empty cells. Your task is to write a program that utilizes a series of deductions to determine how complex a particular puzzle is.

Problem Description

There are advanced methods a computer program can use to solve problems, but the aim here is to determine whether the presented problem can be solved by *people*. Your program should use the methods of deduction outlined below to determine if the puzzle can be solved with those methods alone, and if so, if it can be solved entirely with simple logic or if a degree of deduction is required. **Do not use any other methods of deduction when attempting to solve these puzzles.**

Simple Logic

There are three scenarios which, based on the rules outlined above, allow you to make immediate decisions regarding which number to place in a cell.

- Since a number cannot appear three or more times consecutively, any instance of doubled numbers in a row or column must have the opposite number on either side.

	1	1		→	0	1	1	0
0	0			→	0	0	1	

- Likewise, any time there is a single space between two of the same number, that space must be filled by the opposite number:

0		0	→	0	1	0
1		1	→	1	0	1

- Finally, since puzzles are square and each row and column must contain an equal number of 0's and 1's, the number of 1's (and 0's) in each row or column must be equal to half the dimensions of the puzzle. In a puzzle measuring eight cells to a side, a row or column must contain four 0's and four 1's. A row or column that already contains the required amount of one number must fill its remaining cells with the opposite number.

	1		0	1	1	→	0	1	0	0	1	1

Complex Logic

If the three methods above are insufficient to solve a problem, some more advanced methods may be required.

- Each row and column must be unique. If a row (or column) with two incomplete cells is otherwise identical to an already-completed row (or column), the incomplete cells must be filled with the opposites of their counterparts in the complete row (or column).

0	1	1	0	1	0	1	0	→	0	1	1	0	1	0	1	0
0	1			1	0	1	0		0	1	0	1	1	0	1	0

- Finally, you may be able to identify cases where a cell cannot legally contain a particular value. If a row or column has all but one of its required number of 0's or 1's, you can put the last remaining number in any open space to see if it is a valid move. In the example below, the row already has three of the four required 0's in place. Placing the final 0 in the rightmost cell is not possible, as filling the remaining cells with 1's would cause three 1's to appear in a row. Therefore, the rightmost cell must be a 1. When resorting to this method, be sure to only use the five rules listed in the Problem Background section to validate the current state of the puzzle. Do not attempt to recursively solve the remainder of the puzzle based on that guess if you can't immediately disqualify the entry.

0	0	1	0	1				→	0	0	1	0	1	1	1	0
0	0	1	0	1				→	0	0	1	0	1			1

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a positive even integer, **N**, representing the length of one side of the puzzle.

- N lines, each containing N characters, representing the puzzle itself. Characters may include the digits 0 or 1, representing pre-filled cells, or a period (.), representing a blank cell.

```
3
4
0.1.
..1.
...0
0...
4
0..1
0...
...0
...1
6
......
..0..1
..11..
....0.
.0...1
0....0
```

Sample Output

For each test case, your program must print the results of your program's effort to solve the given puzzle using the methods outlined above.

If a solution was found, your program must print:

- N lines, each N characters long and containing only 0's and 1's, representing the solved puzzle.
- A line containing one of the two following phrases:
    - "Solved with simple logic" if the puzzle was solved only using methods listed in the "Simple Logic" section above, or
    - "Solved with complex logic" if the puzzle required use of one or both methods listed in the "Complex Logic" section above at least once.

If the provided methods were not sufficient to come to a solution, print a single line with the phrase "Unable to solve with the provided logic".

```
0011
1010
1100
0101
Solved with simple logic
Unable to solve with the provided logic
101001
010011
101100
110100
001011
010110
Solved with complex logic
```