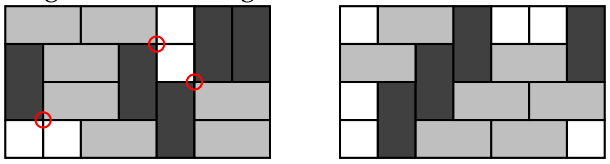
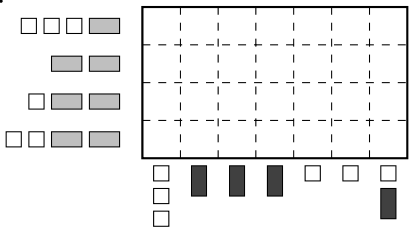
Tomoku

Tomoku is a type of logic puzzle developed by Alejandro Erickson. The idea of the puzzle is to reconstruct the positioning of some tatami mats that cover the floor of a room, given partial information about their position and a condition that no unlucky corners should be formed. What does that all mean? Tatami mats are rectangles or squares which, for the moment, have dimensions 1×1 or 1×2. Obviously they can be used to cover the area of any rectangle that has side lengths which are whole numbers. Two examples, covering a 4×7 rectangle are shown below:

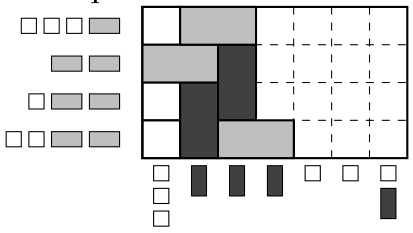


The first of these would be considered unlucky because there are places where four rectangles meet (shown by the red circles). The second has no such place and so would be considered to be an acceptable arrangement of the mats.

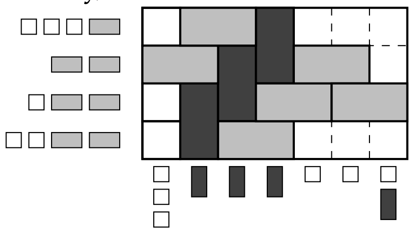
So that's what determines whether an arrangement is acceptable or unlucky. What do I mean by partial information? In the Tomoku puzzle, the partial information is that you know, for each row and each column of the room, how many mats of each type are wholly contained in that row or column. In the case of the second example above here's how we'd show that:



Given such a partial description it's often possible to work out most, or all, of the arrangement. For instance we can see that the three single mats in the first column must be in the first, second and fourth rows and therefore there must be a horizontal mat starting in the third row of the first column. Then in the second column the vertical mat must be in the bottom two rows. In the third column, if the vertical mat were in the top or bottom two rows we'd wind up being unlucky, so it must be in the middle. We can also fill in another horizontal mat in the top and bottom rows starting in the second and third columns since there are no squares in those columns. So, so far we have:



Now the vertical mat in the fourth column must fill the top square since we only have square mats left for the top row and they can't go in the fourth column. In the second row there are four squares remaining and we need to use two horizontal mats so that finishes that row. Now the remaining horizontal mat in the third row can't go at the right hand end or we'd be unlucky, so we have:



The rest is easy – in the bottom row the square mat must be in the lower right corner or we'd be unlucky and there's only one place for the last vertical mat to go. So we can see that the original configuration was the *only* one that was given by the information provided.

Problem Statement

Given a partially described configuration of mats determine whether it arises from an acceptable arrangement, and if so show one such arrangement.

Task

Input will come from `stdin` and will consist of a sequence of "scenarios" (partially described configurations) separated from one another by blank lines. There may also be some lines beginning with the hash character, #. These are comments and should be ignored entirely.

Each scenario consists of four consecutive lines which give the configuration. The first line contains the number of squares in each column from left to right, separated by spaces. The second line consists of the number of vertical mats in each column, separated by spaces. The third line contains the number of squares in each row from bottom to top, separated by spaces. The fourth line consists of the number of horizontal mats in each row, separated by spaces. So the number of columns and rows is to be inferred from the number of entries in the first (or second) and third (or fourth) lines.

There should be no other input to a scenario – but how your program behaves on malformed input is irrelevant.

The output for each scenario is to echo the input for that scenario (not including comments), followed by a blank line and then the required "answer" for the scenario. The answer is either the text “Not achievable” (exactly that text!) or a representation of a configuration that meets the description. In such a representation use the text characters `o` to represent single squares, `--` to represent horizontal mats, and vertically aligned `|`s to represent vertical mats.

For the scenario considered above (and one other) we could have the following input:

```
# A random comment
# Example 1
3 0 0 0 1 1 1
0 1 1 1 0 0 1
# This should be ignored
2 1 0 3
2 2 2 1

# Note multiple blank lines are ok
# Now comes another scenario
2 2
0 0
2 2
0 0

# Extra blank lines at the end shouldn’t cause problems
```

And this should produce the following output exactly:

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

o--|oo|
--||--|
o||----
o|----o

2 2
0 0
2 2
0 0

Not achievable
```

(3 points, Group)