

Problem 9: X's and O's, Tic-Tac's and Go's

23 Points

Problem ID: `tictacgo`

Rank: 3

Introduction

You and Peater often get bored in your APUSH class, so you both make up games to pass the time. Tic-Tac-Toe gets boring quick, and so does Dots and Boxes, and so does Connect 4, and so does... a lot of other games. Yet, one game has remained interesting throughout hours and hours of history lectures: Tic-Tac-Go (a marriage between the games Tic-Tac-Toe and Go), where you try to get a certain number of pieces in a row—but you can also surround, capture, and convert your opponent's pieces. However, the process of converting pieces requires a lot of erasing and rewriting, so you want to write a program to automate the process!

Your task is to create a program that will output the final state of a Tic-Tac-Go grid after a given piece is placed. You do not need to account for win/lose states.

Program Input

The first line of the input from STDIN will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

- A first line consisting of grid dimensions and a target coordinate, separated by a colon. The two values are as follows:
 - The dimensions consist of two positive integers r and c separated by an \times , denoting the number of rows and columns, respectively.
 - The target coordinate consists of two numbers within parentheses separated by a comma. The numbers are as follows:
 - Two positive numbers r_T and c_T representing the row and column of the next X placed within the grid.
 - The coordinate $(1,1)$ represents the top left corner, with additional rows going down and additional columns going right. For example, the coordinate $(3,1)$ represents the position at the third row and at the very left.

- A grid with the given dimensions containing any of the following three characters:
 - An \times denoting a grid space occupied by your piece.
 - An \circ denoting a grid space occupied by your opponent's piece.
 - A dash – representing empty space.

Example Input:

```
5
2x2: (1, 2)
--
XO
```

```
4x5: (3, 3)
X-XOO
OXOOO
-X-OO
OXOOO
```

```
7x2: (6, 1)
-O
-X
O-
O-
OO
--
O-
```

```
5x6: (3, 4)
-----
--XO--
-XO-O-
--XO--
-----
```

```
5x4: (3, 3)
--O--
-OXO-
OX-XO
-OXO-
--O--
```

Program Output

For each test case, your program should output the final state of the grid after the next \times piece is placed. Your output should be created based on the following criteria:

- Same-side pieces directly adjacent to each other are defined to be part of the same clump.
 - Adjacent spaces are defined as the spaces directly above, below, to the right, and to the left of a piece.
- A clump of pieces is defined as surrounded when all pieces within the clump have their adjacent spaces occupied by opposition pieces, grid borders, or same-side pieces. In other words, a clump is surrounded when it is impossible for it to expand in size.
 - If a clump of pieces is surrounded (at least in part) by opposition pieces, all the pieces within the clump are converted to opposition pieces (\times to \circ , and vice versa).
 - All clumps made of the same piece should be converted simultaneously (at the same time).
 - The player who placed the last piece has priority in converting opponents' clumps, meaning any surrounded \circ clumps should be converted to \times pieces first, before the opposite happens.
- A turn ends (described as the final state of the grid) when no clumps on the grid are surrounded (at least in part) by opposition pieces. This may include the player's own pieces.

Example Output:

-X

XX

X-XXX

OXXXX

-XXXX

OXXXX

-O

-X

O-

O-

OO

X-

O-

--XO--

-XXXO-

--XO--

--O--

-OOO-

OOOOO

-OOO-

--O--

Problem Constraints

$T \leq 100$

$1 \leq r, c \leq 50$

Assume the input grid is already in a final state.

Assume the target coordinate corresponds to an empty space.