Problem A: Find the Winning Move

Source file: `win.{c, cpp, java, pas}`
Input file: `win.in`
Output file: `win.out`

4x4 tic-tac-toe is played on a board with four rows (numbered 0 to 3 from top to bottom) and four columns (numbered 0 to 3 from left to right). There are two players, *x* and *o*, who move alternately with *x* always going first. The game is won by the first player to get four of his or her pieces on the same row, column, or diagonal. If the board is full and neither player has won then the game is a draw.

Assuming that it is *x*'s turn to move, *x* is said to have a ***forced win*** if *x* can make a move such that no matter what moves *o* makes for the rest of the game, *x* can win. This does not necessarily mean that *x* will win on the very next move, although that is a possibility. It means that *x* has a winning strategy that will guarantee an eventual victory regardless of what *o* does.

Your job is to write a program that, given a partially-completed game with *x* to move next, will determine whether *x* has a forced win. You can assume that each player has made at least two moves, that the game has not already been won by either player, and that the board is not full.

The input file contains one or more test cases, followed by a line beginning with a dollar sign that signals the end of the file. Each test case begins with a line containing a question mark and is followed by four lines representing the board; formatting is exactly as shown in the example. The characters used in a board description are the period (representing an empty space), lowercase *x*, and lowercase *o*. For each test case, output a line containing the (*row*, *column*) position of the *first* forced win for *x*, or '#####' if there is no forced win. Format the output exactly as shown in the example.

For this problem, the *first* forced win is determined by board position, not the number of moves required for victory. Search for a forced win by examining positions (0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), ..., (3, 2), (3, 3), in that order, and output the first forced win you find. In the second test case below, note that *x* could win immediately by playing at (0, 3) or (2, 0), but playing at (0, 1) will still ensure victory (although it unnecessarily *delays* it), and position (0, 1) comes first.

Example input:

```
?
....
.xo.
.ox.
....
?
o...
.ox.
.xxx
xooo
$
```

Example output:

```
#####
(0,1)
```