UCF Local Contest — September 3, 2016

Don't Break the Ice

filename: `break`
(*Difficulty Level:* `Medium`)

Mr. Pennybags is enthralled with his new board game which involves knocking out tiny plastic ice blocks from a square grid. Pennybags was never very good at games and would like to practice his breaking strategies without having to reset the board. He has asked Unified Coders For Games (UCF Games) to develop such a tool. Pennybags wants a program to take in the description of a board and a sequence of moves and determine the number of invalid moves.

The Problem:

Given the dimensions (number of rows and columns) of a square game board and a list of moves, determine the number of attempted moves that would knock out an ice block that is no longer in the board. Note that when an ice block is knocked out, other blocks may fall out of the board as well. More specifically, an ice block will fall unless it is in a complete row (the row contains all its ice blocks) or it is in a complete column (the column contains all its ice blocks). Note also that if the fall of block B_1 results in the fall of block B_2, then other blocks may fall as a result of block B_2 falling.

The Input:

The first input line contains a positive integer, *t*, indicating the number of game boards. This is followed by the data for each game. The first input line for each game contains two integers (separated by a space): the dimensions (length and width) of a square game board (between 1 and 50 inclusive) and the number of attempted moves in Pennybags' strategy (between 1 and 100 inclusive). This is followed by the row and column (separated by a space) for each attempted move, one move per line. Assume that the input *values* are valid, e.g., the row/column for an attempted move will always be a cell of the game board.

The Output:

For each game board, output "`Strategy #`*`b`*`: `*`i`*" where *b* is the game board number (starting with 1), and *i* is the number of invalid moves. Leave a blank line after the output for each game board.

Sample Input:

```
3
4 5
1 1
1 2
4 1
4 2
1 1
4 4
1 3
2 4
1 4
4 4
3 3
1 1
2 2
3 3
```

Sample Output:

```
Strategy #1: 2

Strategy #2: 1

Strategy #3: 0
```

Explanation of the Sample Input/Output:

In Game #1, “4 2” and the second “1 1” are invalid.
In Game #2, “1 4” is invalid.

UCF Local Programming Contest — Sept 3, 2016

Errata

Don't Break the Ice

Use "breakice" for Filename instead of "break" ("break" is a keyword and causes problems with class names in Java).

Dot the i's and Cross the T's

Assume that the x and y coordinates in the input will have at most three digits after the decimal point. This will limit the accumulation of partial errors while performing the intermediate operations.

Jedi and the Galactic Empire

The input section specifies a limit of "less than 1,000,000" for blaster shots reaching the Jedi; this should be "less than or equal to 1,000".