ECOO 2013

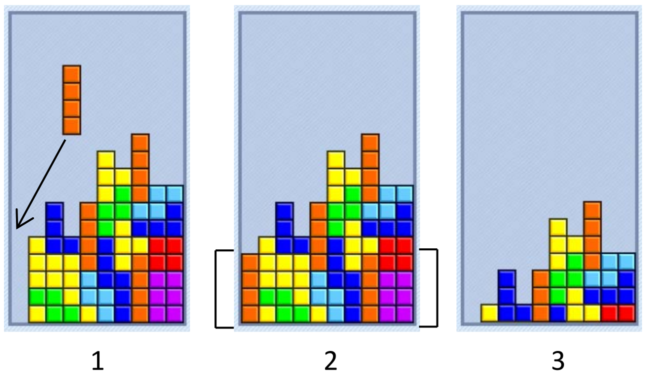
Programming Contest Questions

Regional Competition (Round 2)

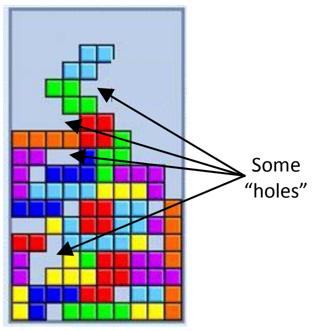
April 27, 2013

Problem 1: Upsetris

The game of Upsetris is played just like the classic game of Tetris, but with a twist. Just like in Tetris, puzzle pieces, each made of four square bricks, fall from above and the user has to decide where they will land to try and make full rows of blocks. The game ends when the pile reaches the top of the screen.



In the game shown at left, a straight vertical piece is falling. If the user drops it in the left most column (picture 1), she will make the bottom four rows full (picture 2), and the full rows will disappear allowing the pile to move down (picture 3).



If the player makes bad decisions about where to put pieces, they can end up with boards with "holes" in them as shown at right. These holes will eventually cost them the game.

In Upsetris, the "twist" (pun intended) is that the user has a one-time-only chance to "upset" the board by rotating it 180 degrees. When this happens, all the individual bricks separate and fall to the floor. This eliminates the holes and sometimes makes new rows.

Here's an example of the "upset" process in action:

```
|          |   |==========|   |          |   |          |   |          |
|          |   |O OOOOO OO|   |O OOOOO OO|   |          |   |          |
|          |   |OOOOO     |   |OOOOO     |   |          |   |          |
|          |   | OOOOO OO |   | OOOOO OO |   |          |   |          |
| O   OOO O|   |O OO O O  |   |O OO O O  |   |  OO      |   |          |
|O O  OOOO |   | OOOO  O O|   | OOOO  O O|   |  OOO     |   |  OO      |
|  O O OO O|   |O OOO   O |   |O OOO   O |   |O OOO     |   |  OOO     |
| OO OOOOO |   |          |   |          |   |OOOOOO OO |   |O OOO     |
|     OOOOO|   |          |   |          |   |OOOOOO OOO|   |OOOOOO OO |
|OO OOOOO O|   |          |   |          |   |OOOOOOOOOO|   |OOOOOO OOO|
|==========|   |          |   |==========|   |==========|   |==========|
      1              2              3              4              5
```

In the example above, the bricks are represented by capital O's and the floor is a row of equal signs. The original board (1) is rotated 180 degrees (2), then the floor of the board is moved to the bottom(3), then the pieces all settle to the new floor (4) making a single complete row which gets removed (5).

DATA11.txt (DATA12.txt for the second try) contains five different Upsetris boards, each with from 5 to 20 rows (not including the “floor”) and from 5 to 20 columns (not including the walls). You must simulate the “upset” operation and show the final result for each board. You must show the complete board in your output, including the “floor” row. You can display the boards under input control if you like (e.g. “hit a key for the next board”) but during judging, you will be allowed to scroll the output screen horizontally or vertically if necessary. The output must be in a fixed-width font (e.g. Courier). If you are unable to configure your IDE to use a fixed width font, you may copy and paste the output of your program into a text editor that uses a fixed-width font during judging. Note that all the boards in the sample input are the same size and have the same number of rows as columns, but this might not be true of the data sets you will be judged on. The side characters for the board are ASCII code 124.

Sample Input

```
|          |
|          |
|0         |
|0 00  0 0 |
|    0 0   |
| 0   0 00 |
|  0     00|
|0 0       |
|   0  0 0 |
|0      0  |
|==========|
|          |
|          |
|          |
|    0     |
|   0  00 0|
|   0     0|
|    0  0  |
|       00 |
```

```
|0         |
|   0     0|
|==========|
|          |
|          |
|          |
|          |
| 0   000 0|
|0 0  0000 |
|  0 0 00 0|
| 00 00000 |
|     00000|
|00 00000 0|
|==========|
|          |
|          |
|          |
|          |
|          |
```

```
|          |
|          |
|          |
| 0     0  |
|     0   0|
|==========|
|          |
|          |
|          |
|          |
|00 00 0000|
| 0 0000000|
|0 0000 000|
| 00000000 |
|0000 0 0  |
|0  000 000|
|==========|
```

Sample Output

```
|          |
|          |
|          |
|          |
|          |
|          |
|          |
| 0       0|
| 0 0   0 0|
| 000  00 0|
|==========|
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|0 0   0   |
```

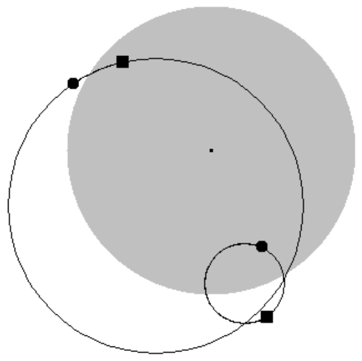
```
|0 0  00   |
|0000 00  0|
|==========|
|          |
|          |
|          |
|          |
|          |
|  00      |
|  000     |
|0 000     |
|000000 00 |
|000000 000|
|==========|
|          |
|          |
|          |
|          |
|          |
```

```
|          |
|          |
|          |
|          |
|0 0 0   0 |
|==========|
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|  0   0   |
| 00 000   |
|000 000 00|
|==========|
```

Problem 2: The Walking Dead

The zombie apocalypse has finally arrived. You were at school when it happened and now you are trapped in a portable classroom on the edge of a schoolyard that is swarming with the re-animated corpses of your former teachers and classmates. Fortunately, you saw this coming weeks ago and cunningly buried explosive devices in key locations. One of them is under the schoolyard. You can start the bomb's countdown timer from your smart phone. Right now the zombies are just milling around aimlessly. When the bomb detonates it will destroy some of them and the rest will start moving towards the sound, giving you the space you need to make it across the field.

Zombies always walk with a limp, so when they have nowhere in particular to go they end up walking in circles. When you trigger your explosive device, the zombies will continue to walk during the countdown and then when the bomb goes off, any zombie inside the circular blast area will be destroyed.



In the picture at right, the grey circle shows the area covered by the bomb blast and the outlined circles show the paths of two zombies. One is moving quickly (for a zombie) in a clockwise direction and is limping quite badly so it walks in a tight circle. The other is limping less and moving more slowly in a counter-clockwise direction. The black squares show their positions when the timer was started and the black circles show where they were when the bomb went off. As you can see one zombie was destroyed in the blast and the other escaped.

DATA21.txt (DATA22.txt for the second try) contains 10 different cases. Each case starts with a line containing four numbers Bx, By, R, and T separated by spaces. The location of the bomb (in metres relative to the center of the schoolyard) is given by the floating point numbers Bx and By ($-500.0 \le Bx, By \le 500.0$ – it's a big school, ok?). The blast radius of the bomb (in metres) is given by the floating point number R ($10.0 \le R \le 500.0$). T is an integer representing the number of seconds until the bomb goes off. This is followed by an integer N on its own line representing the number of zombies on the field ($1 \le N \le 1000$). This is followed by N lines, each representing a zombie using 5 floating point numbers separated by spaces: X, Y, Cx, Cy, and S. The position of the zombie is given by X and Y and the centre point of the zombie's circular path is given by Cx and Cy ($-500.0 \le X, Y, Cx, Cy \le 500.0$). S is the speed of the zombie in metres per second ($-1.0 \le S \le 1.0$, $s \ne 0$) where a positive speed indicates clockwise motion and a negative speed indicates counter-clockwise motion. All floating point numbers in the file have at most two digits after the decimal point.

Your job is to output an integer representing the number of zombies that will be destroyed by the bomb when it goes off. When determining whether a zombie is within the bomb blast area, you should round its position to two decimal places and consider the zombie destroyed if it is within the bomb blast or if it is exactly on the border. Don't worry about zombies bumping into each other as they walk, just assume that won't happen. Note that the sample data below consists of only 5 cases with boldfacing used to separate them visually. The real data files have 10 cases each.

Sample Input

```
0.0 0.0 130.0 500
2
50.0 -150.0 30.0 -120.0 0.3
-80.0 80.0 -50.0 -50.0 -0.1
-17.11 16.81 21.99 79
4
-7.91 22.79 -18.07 13.85 -0.13
-19.53 -19.43 -19.37 13.35 -0.21
10.65 -1.34 0.23 -16.52 0.2
-3.25 -24.61 16.21 18.87 0.17
1.48 21.27 20.24 13
12
-3.6 -20.42 23.02 -15.12 -0.16
19.95 -14.18 -6.0 -23.23 -0.18
5.15 13.06 20.26 11.81 0.26
4.16 -18.42 -13.43 16.23 -0.07
-24.79 12.28 4.11 9.55 -0.13
12.31 11.1 11.87 20.86 0.05
11.78 12.93 -7.82 -24.35 0.03
-14.71 -15.49 16.43 -3.9 0.15
14.31 5.03 -22.68 16.29 -0.05
-22.1 4.86 13.96 -7.16 -0.2
-21.16 -11.18 4.71 -7.64 -0.04
12.74 14.08 -1.88 -5.92 0.16
-13.83 4.62 21.45 61
16
-7.66 -3.24 1.33 -6.53 0.03
-19.0 -0.56 -13.26 24.93 0.03
-7.5 -20.3 -23.88 -5.15 -0.01
-12.09 -4.41 14.17 -0.82 0.08
-11.48 18.29 -1.8 7.82 0.15
20.57 13.01 -17.16 -3.08 -0.15
8.39 0.29 -8.25 -23.22 0.04
6.07 -16.38 -0.48 22.44 0.11
16.92 20.4 0.64 -16.19 0.02
-14.67 -1.2 -18.32 3.41 0.25
16.27 1.26 -2.92 18.78 -0.12
-9.25 1.74 -8.01 2.07 -0.07
10.88 17.68 6.2 3.94 0.23
21.08 -9.37 10.94 1.29 0.09
11.31 8.68 -17.95 9.54 -0.1
-1.46 21.11 13.39 9.04 -0.12
4.57 8.57 18.83 169
20
22.2 5.22 18.09 18.98 0.04
-24.75 -14.48 -24.14 -16.0 0.01
-5.22 -16.06 5.75 -3.2 -0.05
9.5 22.34 12.87 -24.34 0.06
-22.32 19.67 16.79 8.64 -0.02
4.46 10.22 14.43 22.25 -0.21
-15.7 -18.05 1.37 -5.16 0.19
-8.34 -6.63 -16.61 16.73 -0.07
22.78 -17.16 19.91 12.26 0.23
-14.35 18.54 1.5 -11.14 0.14
-5.22 9.95 21.32 5.24 0.08
1.61 9.98 -22.21 -2.42 0.22
-12.72 15.29 -3.63 -19.68 0.05
-3.74 9.51 -7.84 -3.44 -0.08
-20.14 -17.28 -0.82 2.43 -0.24
-11.01 -12.15 -0.81 11.95 -0.17
10.91 1.02 -13.54 18.59 -0.02
-6.2 7.71 18.18 -20.29 -0.08
-22.82 -24.88 23.3 -7.44 -0.13
20.95 -16.7 7.61 -23.1 -0.06
```

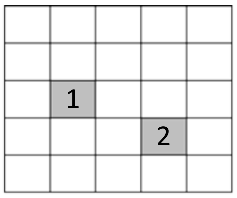
Sample Output

```
1
1
4
7
8
```

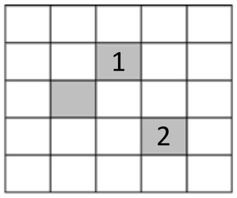
Problem 3: Last Robot Standing

You are entering your robot into a simple game of strategy. The game is played on a grid and the robots take it in turns to move one space at a time (including diagonally) around the grid. When a robot moves into a square, that square is taken and nobody else can move into it for the duration of the game. When a robot is boxed in and cannot move on its turn, it is out. The winner is the Last Robot Standing.

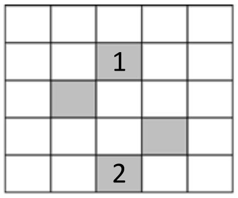
Here's an example of the game in action on a small grid with 2 robots. The positions of the robots are marked 1 and 2 and the shaded squares are taken and cannot be used again by either robot.



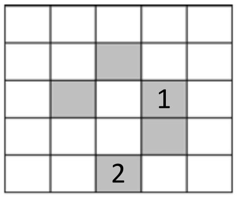
Start



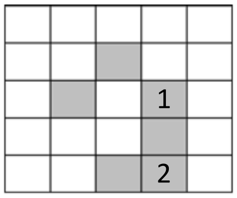
A



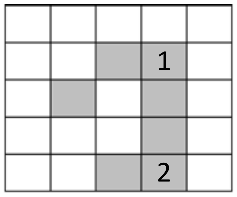
B



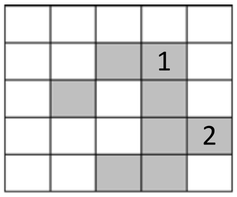
C



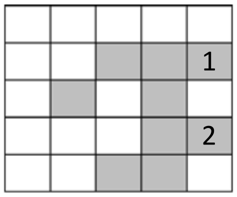
D



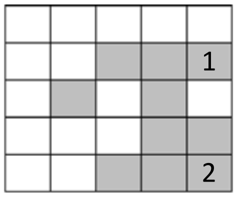
E



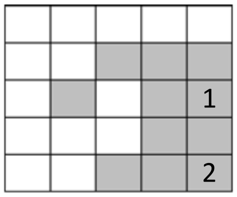
F



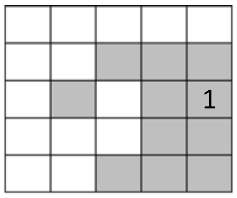
G



H



I



J

In this example, Robot 1 is the Last Robot Standing. Note that Robot 2 was boxed in on Move H, but was not removed from the game at that point. Any robot that is able to move on its turn is safe for that turn. It is only on its next move (Move J) that it is removed from the game. Note that if Robot 2 had started out just one square to the left and used the same set of moves, it would have won.

8	1	2
7	▨	3
6	5	4

The robots in the above game are choosing their moves from a hardwired "strategy" that tells them the order of moves to try. There are 8 possible moves, numbered clockwise from 1 to 8 as shown at right. A robot's strategy can be represented as a list of numbers. Robot 1's strategy is 2481377746543, and Robot 2's strategy is 62437428513.

On Robot 1's first turn (A), it uses move 2, which is up and right. Then on its next turn (C) it uses move 4 (down and right). Then on its next turn (E) it tries move 8 (up and left) but is blocked, so it uses the next move in its strategy, which is 1 (up). If a Robot ever runs out of moves in its hardwired strategy, it simply starts again from the beginning. Note that this can only work if each of the 8 moves appears in each strategy at least once.

Using your evil hacking skills, you have found out the strategies of the other robots. Since you get to pick your starting position and you're moving last, you can figure out a winning start position.

DATA31.txt (DATA32.txt for the second try) will contain 10 cases. Each case starts with a line of 3 integers R, C, and N. R and C are the number of rows and columns on the playing surface ($4 \le R,C \le 10$) and N is the number of robots in the competition ($2 \le N \le 16$). Your robot is the Nth one. Following this are N-1 lines containing the integer starting position for each of the other robots (row then column, numbered from 1 upwards), then N lines containing the strategy for each robot (including your own). During the game, the robots take turns in the order they appear in the file, with your robot moving last.

Your job is to output a list of starting positions that will lead to a guaranteed win for your robot. If there is no such starting position, you should simply output the word "LOSE". Your output must be formatted exactly as shown below with no extra spaces or punctuation, but the order you output the starting positions for each case can be different from that shown. Note that the sample input below only contains 4 cases, but the data files will contain 10 cases each.

Sample Input

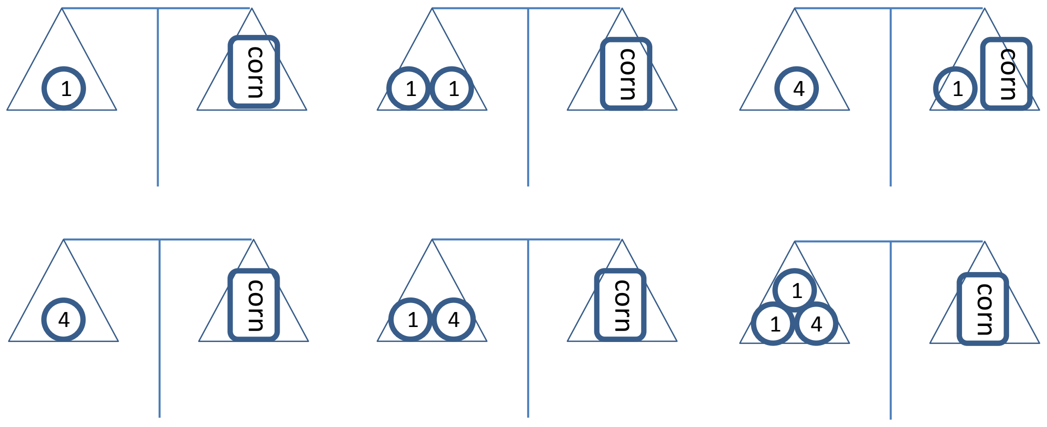
```
5 5 2
3 2
2481377746543
62437428513
3 3 3
1 1
2 2
45362718
45362718
87654321
5 5 3
1 1
3 3
81726354
45362718
12345678
10 10 8
1 3
1 8
3 1
8 1
3 10
8 10
10 3
12345678
87654321
18273645
54637281
21436587
78563412
16372485
73265841
```

Sample Output

```
(1,2) (1,3) (2,3) (2,4) (3,1) (4,5) (5,2) (5,5)
(3,3)
(3,2) (4,2)
(1,9) (4,9) (4,10) (5,7) (7,1) (7,2) (7,6) (7,7) (9,4)
```

Problem 4: Breaking Rocks

Farmer Jane needs to be able to measure out corn to feed her cows, but all she has to help her is a primitive balance scale and a 6 kilogram rock. With this rock she could use the balance scale to measure out 6 kg of corn, but she often needs to measure out smaller quantities. She figures out that if she breaks the rock into three pieces, where two of them are 1 kg and the third is 4 kg, then she can measure out all integer quantities of corn from 1 to 6, as shown below.



Farmer Jane is happy now, but the situation gets her thinking. She knows she could have broken the rock into 1, 2, and 3 kg pieces and this would also have worked. But things are not so simple for other numbers. For example, there are 15 ways to break a 12 kg rock into 4 integer pieces but only 9 of them let you measure all integer weights from 1 to 12. She wonders if there could be some kind of algorithm to determine how many combinations work for a given size of rock and a given number of pieces…

DATA41.txt (DATA42.txt for the second try) contains 10 test cases. Each test case consists of two integers (**P** and **R**) on a single line separated by a space. The integer **P** gives the number of pieces to break the rock into and the integer **R** gives the original size of the rock. For all test cases, $1 \leq R \leq 100$. For the first 5 test cases $3 \leq P \leq 5$ and for the next 5 test cases $6 \leq P \leq 10$. Your job is to output a single line for each test case indicating the number of ways you can break up the rock into **P** integer-sized pieces so that all possible integer weights from 1 to **R** can be measured on a balance scale.

Sample Input

```
3 6
4 12
4 30
5 40
5 5
6 25
7 55
8 65
9 75
10 85
```

Sample Output

```
2
9
5
137
1
154
5749
28051
121108
474402
```