ECOO 2013
Programming Contest Solutions and Notes

Regional Competition (Round 2)

April 27, 2013

Sam Scott, Sheridan College (sam.scott@sheridancollege.ca)

Problem 1: Upsetris

Recommended Approach

If you try to actually simulate the “upset” process, you will get bogged down. All you really need to do is count the number of bricks in each column. To simulate the deletion of full rows, find the column with the smallest number of bricks and subtract this number from all columns. Then you have to remember to output the columns in the reverse order because the board is rotated 180 degrees.

Solution to DATA11.txt

```
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                0 |
|                0 |
|  0  0   0  0   0 |
| 00  0 0 00 0   00|
| 00 0000 00 0 0 00|
|00000000000 000000|
|00000000000 000000|
|00000000000 000000|
|==================|
```

```
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|          0     |
|          0     |
|       0  0  0  |
|000 0000000  0 0|
|================|
```

```
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|=======|
```

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

```
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|        00|
| 0 0   000|
| 0 0 00000|
| 000 00000|
| 000000000|
| 000000000|
|==========|
```

Solution to DATA12.txt

```
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|0   0   |
|00  0000|
|========|
```

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

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

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

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

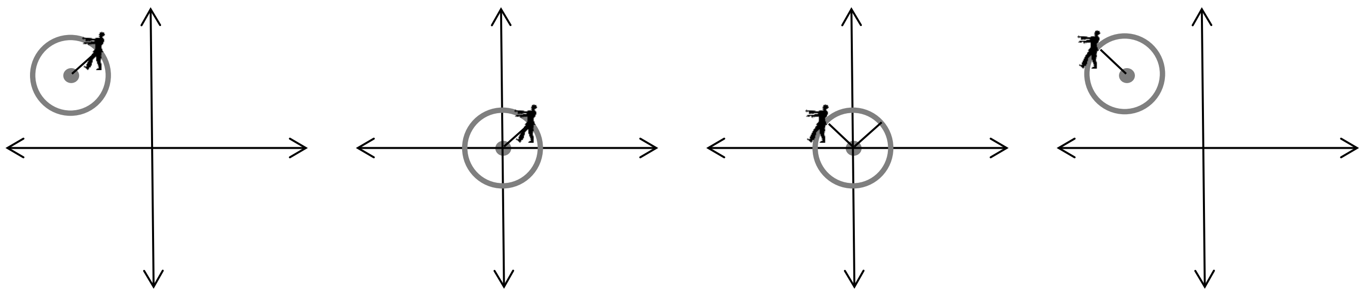
Problem 2: The Walking Dead

Recommended Approach

You have the centre of each zombie's circle and their current position. You can get the radius of the circle by computing the distance between these two points with the Pythagorean theorem. Then you can get the distance a zombie walks by multiplying the countdown time by its speed. The tricky part is then figuring out where the zombie will end up.

First compute the circumference of the circle ($C=2\pi r$). Then divide the distance walked by the circumference. This gets you a number that represents how far around the circle they went (e.g. if you end up with 0.5 then they walked half way around the circle, which is 180 degrees or π radians).

Now translate the circle to the origin by subtracting Cx and Cy from the zombie's position. You can figure out the tangent of the current angle the zombie is making with the x-axis, then use arctan to get the angle. Now add the angle they walked and use sin and cos on this new angle to get x and y coordinates representing where they end up. Because you have translated to the origin, you have to translate back by adding Cx and Cy to the zombie's final position.



Once you have the final position of the zombie, round it to two decimal places and compute the distance to the bomb. If this distance is less than or equal to the blast radius, the zombie is destroyed.

When using arctan to get the angle, your system will probably always return an angle that is between 90 and -90 degrees, which would always put the zombie in quadrant I or IV of the Cartesian plane. Therefore, if your zombie is in quadrants II or III to the left of the y-axis (i.e. to the left of the circle centre before translation) then you will have to add or subtract 180 degrees (π radians) to get the real angle. Also, depending on the language you are coding in, you may have trouble with 90 degree or -90 degree angles and you might have to treat them like a special case.

The Test Cases

Although the question was very clear on when to judge that a zombie is within the blast radius, it was decided that for the test cases every zombie should be clearly in or out. In no test case did a zombie end up within 0.02 m of the border. Each set of test cases contains a couple of cases where there is at least one zombie starting with the same X coordinate as its circle centre, putting it at either 90 or -90 degrees.

Solution to DATA21.txt

```
11                        82                        74
60                        286                       65
68                        50
25                        30
```

Solution to DATA22.txt

```
34                        62                        361
65                        195                       38
40                        0
22                        171
```

Problem 3: Last Robot Standing

Recommended Approach

The only way to do this is to simulate the game for each possible starting position and then keep track of the ones that lead to a win for your robot. Use a two-dimensional array for the board but don't forget to re-initialize it for each run.

Keeping track of the players can be a bit tricky. You need some way to keep track of each robot's position (an array or set of arrays will do this) as well as where each robot is at in using its strategy. One way to do this is by using an array that indicates the current position inside the strategy of each robot. Another way is to represent the strategies as strings, then remove each move as you make it and concatenate it onto the end of the string again.

The fact that robots are not removed until their next turn is not really important. It makes no difference to the outcome of the game if you remove the robots on the turn they box themselves in or on their next turn as long as you apply the rule consistently.

Solution to DATA31.txt

```
(6,4)
(1,1) (2,3) (4,5)
(3,1) (4,2)
(3,3) (4,4) (6,1) (6,5) (9,3)
(1,7) (2,8) (2,9)
(1,9) (3,3) (3,4) (5,1) (5,5) (5,9)
LOSE
(3,1)
(1,3) (5,2) (5,4) (5,6) (6,1) (6,2) (6,3)
LOSE
```

Solution to DATA32.txt

```
(4,3) (5,6) (5,7) (6,3) (6,5) (6,6) (6,7) (7,3)
(2,6)
(3,4) (5,3) (5,4) (6,2) (6,3) (6,4) (7,2) (7,7)
(2,4) (2,5) (3,4)
(2,3) (2,4) (3,4) (4,4)
(1,1) (1,3) (2,2) (2,3) (3,1) (3,2)
(1,1) (1,2)
(7,1) (8,2) (8,3)
(2,7) (4,7) (5,6)
(3,4) (3,5)
```

Problem 4: Breaking Rocks

The Basic Idea

The basic approach here is to generate all the possible ways to split the rock, then check each combination to see if it works. You can use a recursive algorithm to generate all the possible splits, and it's a good idea to only generate the pieces in ascending order because for this problem 1 1 3 is the same combination as 1 3 1.

For the example of a 12 kg rock split into 4 pieces, the possible splits are:

> 1 1 1 9, 1 1 2 8, 1 1 3 7, 1 1 4 6, 1 1 5 5, 1 2 2 7, 1 2 3 6, 1 2 4 5, 1 3 3 5, 1 3 4 4, 2 2 2 6, 2 2 3 5, 2 2 4 4, 2 3 3 4, and 3 3 3 3

Checking each combination for whether it works is where things get a bit tricky.

An Inefficient Solution

You could use a recursive backtracker to test all the possible ways to place the pieces on the balance. Each piece could be on the left, on the right, or not on the balance at all. For each combination, the difference between the two sides of the balance is the amount of corn you can weigh. In the example above, there are 81 possible combinations for each set of 4 pieces. You keep track of which amounts of corn are covered as you generate the possible combinations and a split is good if it covers all the numbers from 1 to R.

This approach works, but the problem is that the number of combinations you might have to check grows exponentially (3^P). Depending on your code you will probably only solve some of the test cases within 30 seconds if you use this method.

An Observation

If you start generating and testing splits for any large problem (e.g. 10 rocks and 100 kg) you might start to see a pattern emerge. The first thing you might notice is that you always need a rock of size 1. Without this rock, you could never weigh the quantity **R**-1 (where R is the size of the original rock). So every split you generate should always start with 1, and this makes things faster.

But the bigger pattern that emerges is that the nth piece never seems to get larger than 3^{n-1}. So a split that starts 1 3 9 27 … could work, but a split that starts 1 3 10 29… could never work. You can use this fact to limit the splits that you try and this might help you get one more test case before your time is up.

An Efficient Solution

But if you think more deeply about the structure of the possible splits, it could lead you to an even more efficient solution that would be easily fast enough to complete all of the test data within 30 seconds. Can you figure it out? Or do you have a completely different approach that works? Post your ideas to the appropriate forum at compsci.ca!

Solution to DATA41.txt

```
0
2
1
12
131
1935
14384
78027
355183
1430193
```

Solution to DATA42.txt

```
3
11
45
52
108
2014
15063
82928
379105
1335694
```