Contest Analysis

Round 1A 2010

A. Rotate

B. Make it Smooth

C. Number Game

Contest Analysis

Questions asked 1

Submissions

Rotate

11pt	Not attempted **2076/2436 users** correct (85%)
12pt	Not attempted **1855/2071 users** correct (90%)

Make it Smooth

12pt	Not attempted **509/954 users** correct (53%)
24pt	Not attempted **319/482 users** correct (66%)

Number Game

16pt	Not attempted **680/1091 users** correct (62%)
25pt	Not attempted **244/450 users** correct (54%)

Top Scores

rng..58	100
Pipi	100
cgy4ever	100
rem	100
XiaoZiqian	100
qizichao	100
exod40	100
GarnetCrow	100
hos.lyric	100

Overview | Problem A | Problem B | Problem C

This is a relatively straightforward simulation problem -- the problem statement tells you what to do, and you just need to do it.

Well, except for one fun point: The name of the problem is *Rotate*, and in the statement we talk about the rotation a lot. However, that is the one thing you do *not* need to implement! Rotating 90 degrees clockwise and pushing everything downwards has the same effect as pushing everything towards the right without rotating. As long as you push the pieces in the correct direction, it doesn't matter whether you actually do the rotation. Any **K** pieces in a row will be the same in these two pictures, and your output will be the same too.

So, a simple solution to this problem looks like this:
(1) In each row, push everything to the right. This can be done with some code like the following:

```
    for (int row = 1; row < n; ++row) {
      int x = n-1;
      for (int col = n-1; col >= 0; col--)
        if (piece[row][col] != '.') {
          piece[row][x] = piece[row][col]; x-
        }
      while(x>=0) {piece[row][x]='.'; x--;}
    }
```

(2) Test if there are **K** pieces in a row of the same color. There are tricks that can be done to speed this up, but in our problem, **N** is at most 50, and no special optimizations are needed. For each piece, we can just start from that piece and look in all 8 directions (or we can do just 4 directions because of symmetry). For each direction, we go **K** steps from the starting piece, and see if all the pieces encountered are of the same color. The code -- how to go step by step in a direction, and how to check if we are off the board -- will look similar in many different programming languages. We encourage you to check out some of the correct solutions by downloading them from the scoreboard.

ACRush	100