Florida High School Programming Series
2013 Inaugural Championship

Problem Writer's Analysis by Arup Guha

Problem 1: Fantasy Baseball

This problem was intended to be one of the more challenging problems in the set. The first major challenge the problem presents is reading through a rather lengthy set of rules. None of the rules require a complicated calculation, since both key calculations are simple divisions. Instead, this problem requires the solver to contend with String input, properly count at bats, plate appearances, hits, times on base, and properly rank all of the players.

An object-oriented design makes the logic in this program a bit easier to handle. In Java, one can create a player class that implements Comparable<player> and can define all of the comparison logic in the compareTo function, making use of Java's built in sorting function.

The only judging issue dealt with a blank line at the end of the judge's output file. The judges dealt with this manually, making sure that any team that matched all of the output text (minus that last blank newline) got a correct response.

Problem 2: Piece of Cake

This problem was meant to be one of the easier problems. Via calculus, one can prove that the shape of a square minimizes the perimeter of a rectangle, given that the area is fixed. However, it was expected that students would realize this by working with a few examples. Thus, the problem boils down to finding two positive integers x and y that multiply to the input value n, where the difference between x and y is as small as possible.

One tiny twist to this problem was that the possible range of n was big enough that a full brute force search that tests whether each number from 1 to n is a divisor of n times out, since n could be as big as 500,000,000. Thus, in order to solve the problem quickly enough, one must realize that she can stop testing after reaching the square root of n, since one of the two factors is guaranteed to be less than or equal to the square root of n. A slight optimization would start testing value starting at the square root of n, going down and stop as the first valid factor was hit. Either of these last two approaches solved the problem within the given time limit.

Problem 3: Coupon-Palooza

The intention of this problem was to reward students who realized that a greedy solution was viable, instead of trying a brute force solution (which is possible since there are only 10 total discounts at most). An absolute brute force solution would try all 10! orders of discounts and pick the best one.

The greedy solution is as follows: apply all of the percentage discounts in any order first. Then apply all the fixed dollar off discounts at the end. The reason for this is as follows:

Consider one collected percentage discount off, P and one fixed dollar discount Q, where $0 < P < 100$ and $Q > 0$. Let $M = (100 - P)/100$, the multiplicative factor that corresponds to a discount of percent P. Note that $0 < M < 1$. Let the original price of the item be X.

If we apply P then Q, our final price is $XM - Q$.

If we apply Q then P, our final price is $(X - Q)M = XM - QM$

These two values are the same except what is subtracted out. Since we know that $M < 1$ and $Q > 0$, it follows that $QM < Q$. Thus, a larger value is being subtracted out in the first calculation than the second. This intuitively should make sense. If we apply any subtractive discounts first, then our percentage off later gets diminished, since its applied to a smaller value. (You save more money if you get 20% off $100 than $90, for example.)

In the judging of this problem, there was one test case that all teams missed. When worked out exactly, the math for this case had a final answer that was some number of dollars and exactly 30.5 cents. According to the problem specification, this should be rounded up to 31 cents. However, all of the received solutions were printing out 30 cents. (The judge solution had 31 cents.) This is due to floating point error inherent in any calculations with doubles. Occasionally tiny errors cascade, as was the case in this problem. The judge solution combined all percentage discounts into one variable and all subtractive discounts into one variable before applying either to the original price an an effort to reduce cascading error. Even this isn't always going to work. A standard solution to this dilemma is adding a tiny epsilon, say 10^{-10} to the answer to be printed. The rationale behind this is that rarely, in any real life problems is the actual answer 5.34499999999999. But, occasionally the actual answer will be 5.345. Due to this bias, the trick of adding a tiny epsilon usually fixes most floating point errors of this nature.

After quick deliberation, Mr. Dencker decided that students who only had this one case wrong should get the question correct. The judging staff removed this one corner case from the test file and rejudged all submissions. In a collegiate contest, this type of change would not have been made, but I agree with Mr. Dencker's decision to remove this case for a contest of this level.

Problem 4: Five Dollar Deal

This problem was the banger in the set. I expected every team that came to the contest to solve this question. Just multiply the input by five!

Problem 5: Late Night Wake Up

I do have a four month old daughter and for the first three months of her life, my wife and I had to negotiate wake up times, hence the motivation for the problem. In some ways, this problem is similar to the baseball problem. No individual calculation is very difficult, but there are multiple rules to invoke and some non-trivial String processing steps. A good program design helps simplify the solution to this problem. One key problem that arises is converting a time to an integer, representing the number of minutes past 11pm. Once a function/method is written to do this task, then the problem becomes easier to solve.

I created judge data to try to thwart any greedy solution short of calculating all the necessary values for each set of wake up times. I had cases where one person got the longest stretch of sleep but didn't sleep as much as the other person. I had cases where each wake up stretch was shorter than the other competing schedule but the other competing schedule had more sleep. I had cases where one wake up schedule was "losing" against the other until the very last wake up. I had cases where the longest stretch of sleep was from the beginning to the first wake up and cases where the longest stretch of sleep was from the last wake up to 8am. For each case, I added and subtracted one minute of sleep from one of the two schedules to test the boundary conditions.

In short, this question requires the solver to carefully consider a program design that easily handles these multiple boundary conditions. Every calculation comes down to correctly calculating the number of minutes in any given time interval. Once this subproblem is solved correctly, for each wake up schedule, the total number of minutes up and the longest stretch of sleep must both be calculated. Then the required comparisons must be made.

Problem 6: Origami

This problem is a (much) easier version of the 2011 South East Regional Problem "Folding", where up to 20 folds similar to the folds in this problem are applied to a piece of paper. The logic for following 2 folds turns out to be easier but still has multiple cases to consider. In particular, on each axis, there are two cases to consider: either the fold is made less than halfway along the paper, or more than halfway. (The corner case of folding in half can be included in either of the two cases previously listed.) Once the student works out the math for both cases, this simply has to be split into four separate cases since there are two folds. The key calculation is simply the absolute value between where the 0-edge of the paper ends up (twice the x or y value of the folding axis) and the opposite edge.

Problem 7: Maximum Stock Return

This problem was meant to be the most difficult in the set. Not only does the problem require reading through several rules, but the easiest implementation requires recursion, which is a difficult topic at this level. In a nutshell, the problem requires a calculation of the maximum gain from investing stocks over a 15 day period. At the beginning of each day, if one has just cash, they may either keep the cash or buy one of the two stocks. Alternatively, if one owns stock, they may either keep that stock, switch to the other stock, or just sell their stock. Thus, the total number of possible actions is 3^{14}, since we have three choices for all days except the last one, when we must sell.

In designing the recursive function, one must pass into it all information related to the state of the computation. This includes the current amount of money, the current amount of each of the two stocks owned (Technically, we know that at least one of these will be 0.), and the current day of the simulation. The base case of the recursion is if we're on the last day. Otherwise, depending on our situation, we make recursive calls corresponding to what would happen for each of our actions and simply take the maximum value of each of these calls. As looking at both judge solutions indicates, occasionally, recursion requires giving a function access to a fair number of parameters.

Problem 8: Up-wards

The inspiration for this problem was a counting problem in the 2012 Jordanian contest that asked to count the number of strings in alphabetical order of n letters, where n was the input. The answer is based on the formula for combinations with repetition. I decided to adapt the problem to make an easy string question, so that students would have to check to see if a word had its letters sorted in alphabetical order. I decided to add the condition of no repeats so that the students couldn't just call a sort function to sort the letters and compare that to the original. Needless to say, regardless of that particular decision, this problem boils down to using a simple loop. The only border conditions I could think of were strings of 1 letter, strings with a pair of repeated letters and a few longer strings with all letters in order except the last two.