Statistics

Ma

SRM 152
Thursday, June 26, 2003

Arch
Norr
Disc

Match summary

SRM 152 went pretty smoothly, with submission rates close to their long-term averages. The division 2 hard problem wa exception to this, and a great many people submitted it, but few were successful. **SnapDragon** started on a new streak, round after his previous steak of 3 SRM's was broken in SRM 151. **LunaticFringe** was a distant second, 126 points beh place, **tomek** continued to be perfect, successfully submitting all 3 problems for his 4th straight competition. In division 2 edged out the competition by an equally large margin, getting high scores on all three problems. **jpo**, in fifth place was th scoring new comer, with 1298.59.

The Problems

FixedPointTheorem Discuss it

Used as: Division-II - Level 1:

Value	250
Submission Rate	193 / 219 (88.13%)
Success Rate	191 / 193 (93.78%)
High Score	**Veloso** for 246.60 points

There's not a whole lot to this one. Basically just follow the instructions. You are given a simple function: F(X)=R*X*(1-X) with X = 0.25, and R is an input. Then, you repeatedly set X = F(X), 200000 times. After this, you simply do 1000 more it take the maximum minus the minimum of those iterations. There wasn't really anything tricky here. One thing to note is t range of R ensures that F(X) will always be between 0 and 1, exclusive. Since it starts in this range, we can show this by Assuming that it X is in the range 0 to 1, exclusive, we want to show that F(X) is also in this range. The lower bound is p since both X and (1-X) are positive. The upper bound comes from the fact that X*(1-X) <= 0.25 for all 0<X>1, and the fac So, since X is bounded thus, you don't have to worry about overflow or underflow, as the problem states. It turns out tha few values in the input range that will not converge in 200,000 iterations (and hence are not allowed), but they will conve eventually. On the other hand, for larger values of R, above 3.569, F(X) may not converge at all, but instead forms an in repeating series.

LeaguePicks Discuss it

Used as: Division-II - Level 2:

Value	500
Submission Rate	175/219 (79.91%)
Success Rate	131 / 175 (83.55%)
High Score	**Veloso** for 492.59 points

Used as: Division-I - Level 1:

Value	250
Submission Rate	130 / 132 (98.48%)
Success Rate	124 / 130 (98.48%)

High Score	**SnapDragon** for 248.41 points

This problem is about a TopCoder fantasy league, an idea which vorthys takes credit for. In particular, there is a draft wh get to choose who they have on their fantasy team (dibs on SnapDragon). The order in which the friends in this problem people is the same as the way competitors are assigned to rooms in TopCoder tournaments. Namely, you order everyor down the list, and back up the list, repeatedly until everyone is drafted. This problem asks, given your place in the list, th number of people playing in the league, and the total number of people to be drafted, which picks will you get. The most to do this is to iterate over all picks, and keep track of whose turn it is. Start by going up, and reverse direction each time the end of the list. Since there are at most 1600 draftees, this is plenty fast, are pretty simple to code. Another way to do that you start with the pick numbered the same as your position. friends - position people then go after you, and each go and then you get the next pick. So the next pick is 2 * (friends - position) + 1 later. After that, the people before you go, s pick is 2 * position - 1 later. If you add these two quantities alternately, you will get all the right numbers, and you simply when you get to a number that is bigger than the number of draftees.

ProblemWriting

Discuss it

Used as: Division-II - Level 3:

Value	1000
Submission Rate	99/ 219 (45.21%)
Success Rate	16 / 99 (16.16%)
High Score	**Veloso** for 786.09 points

This problem involves checking that a given string conforms to a certain grammar (which is also used in the division 1 ha returning an error message if it does not. A lot of people seemed to struggle with the notation in this problem, so before analysis, it might be helpful to read about Backus Naur Form.That said, the simplest way to implement this is to write a s machine. In a state machine, there are a number of different states, and we simply iterate through all of the characters in changing states based on what those characters are.There should be 4 states in our state machine. The first state, whic s_0, is our start state. It indicates that the next character in the string should be a digit. So, if we are in state s_0, and encou we advance to state s_1. If we are in state s_0, and encounter anything other than a digit, then an error message should be From s_1, there are three different state transitions, depending on the character encountered:

1. If the character is a '.', then move to state s_2.
2. If the character is an operator, then we advance to state s_3.
3. Otherwise, return an error message

Now, state s_2 turns out to be exactly the same as state s_1 in terms of advancing to other states (there is an important dif which we will get to later).If we are in state s_3, then there are two cases based on the character encountered:

1. If the character is a '.', then stay in state s_3.
2. If the character is a digit, then we advance to state s_1.
3. Otherwise, return an error message

That is it for our state transitions. Now, the distinction between states s_1 and s_2 is that the machine must be in state s_1 a characters have been read, or else an error message should be returned. The idea of a state machine is a very useful o would behoove all coders to become familiar with it. Anyhow, here it is in code:

```
int S0 = 0, S1 = 1, S2 = 2, S3 = 3;
String ops = "+*-/";
if (dotForm.length() < 1 || dotForm.length() > 25)
              return "dotForm must contain between 1 and 25 characters, inclusive.";
```

```
boolean good = true;
int STATE = S0;
for (int i = 0; i < dotForm.length(); i++) {
      char curr = dotForm.charAt(i);
      good = true;
      if (STATE == S0) {
         if (Character.isDigit(curr)) STATE = S1;
         else good = false;
      } else if (STATE == S1 || STATE == S2){
         if (curr=='.')STATE = S2;
         else if (ops.indexOf(curr)!=-1) STATE = S3;
         else good = false;
      } else if (STATE == S3) {
         if (curr=='.') STATE = S3;
         else if (Character.isDigit(curr)) STATE = S1;
         else good = false;
      }
      if (!good) return "dotForm is not in dot notation, check character "+i+".";
} if (STATE!=S1) return "dotForm is not in dot notation, check character "+(dotForm.leng
return "";
```

QuiningTopCoder Discuss it

Used as: Division-I - Level 2:

Value	500
Submission Rate	77 / 132 (58.33%)
Success Rate	48 / 77 (62.34%)
High Score	**LunaticFringe** for 351.53 points

In this problem you had to perform a simulation of a machine executing a fairly simple programming language. There we different cases, and it looks overwhelming at first glance. However, if you are methodical, and follow the directions well, hard. One way to make it a little more manageable is to write two functions, push(int n) and pop(), to handle all of your s management. Doing all of the stack work for each case is a good way to make your code harder for everyone, including read. Other than that, it was mostly just a matter of following directions carefully. The character printing could also be ma more orderly by writing a method to handle that, but its not as important as the stack functions. As an interesting exercis a Quine in your language of choice. Some languages are can be done with many fewer characters than others.

DotNotation Discuss it

Used as: Division-I - Level 3:

Value	1000
Submission Rate	13 / 132 (9.85%)
Success Rate	8 / 13 (61.54%)
High Score	**SnapDragon** for 714.58 points

The best approach to take to this problem is recursion with memoization. Start with the entire expression, and determine expressions within the expression could be the dominant ones. Then, for each potentially dominant operator, split the ex two parts around that operator and its dots, a left and right operand, and recursively call your function. Your recursive m return a list of all of the possible values that an expression could be evaluated to. So, when the lists for the left and right operator have both been evaluated, you can take every possible pair of values from the left and right, and evaluate the Then, you add all of these results into your return list. So, the pseudocode (with memoiziation) goes something like this:

```
recurse(string expression){
        if(recurse(expression) has already been calculated)
            return cached value of recurse(expression);
        list ret;
        foreach(possible dominant operator){
            string left = left operand of operator;
            string right = right operand of operator;
            foreach(value l in left){
                foreach(value r in right){
                    if(l op r is not in ret)
                        add l op r to ret;
                }
            }
        }
        save ret to cache;
        return ret;
    }
```

Now, the main part that is left out is how to determine which operators might be dominant. The simplest way to determin just try them all. For each one, count how many dots there are before and after the operator, and then check all of the ot operators before and after the operator in question to see if the operator in question can be dominant. You also have to careful about your data structures. In particular, you don't want to be doing a linear search of ret every time you want to number to it. Once you have your recursive function written, you simply return the size of the list it returns when called o expression.

By **lbackstrom**
TopCoder Member