TopCoder Competitions

SRM 603

Single Round Match 603
Monday, January 6th, 2014

Archive
Match Overview
Discuss this match

Match summary

This was the first match of 2014. Most of the problems were contributed by **subscriber**. **lyrically** contributed the division 1 medium. Division 1 required us to find a simple, yet elusive solution. The medium problem was more complicated, it starts as a string counting problem, but requires some number theory so you can reduce it to dynamic programming or even more number theory to solve it entirely. Congratulations to **Blue.Mary** for getting the fastest score in such a problem. The hard problem required all sorts of tricks to find a solution that required some advanced math. This was **Egor**'s moment to shine: The fastest 1000 points problem, fast solutions to the other problems and 100 challenge points. **Egor** won the match with almost 200 points of advantage over **VArtem**. **VArtem**'s second place was still very impressive with ~250 more points than **cgy4ever**. The top 3 were the only coders to solve all three problems. Congratulations also to division 2 winner: **yuya_n**.

The Problems

MiddleCode | SplitIntoPairs | GraphWalkWithProbabilities | MaxMinTreeGame | PairsOfStrings | SumOfArrays

GraphWalkWithProbabilities

Used as: Division Two - Level Three:

Value	950
Submission Rate	28 / 878 (3.19%)
Success Rate	5 / 28 (17.86%)
High Score	**ImbaOvertroll** for 747.21 points (15 mins 43 secs)
Average Score	623.48 (for 5 correct submissions)

The results for each node

The problem has a recursive logic. We want to find the probability to win in a given node of the graph. However, in a single move, we should decide whether to stay in the node or to move somewhere else. This decision depends on which of the other nodes has a better probability (maybe staying in the original node is better). So we need to know their probabilities too.

The moves you made in the past, do not affect the probability, only the current position. In other words, the game is memory-less. This means that for each position we should be able to find a probability.

If there is a cycle in the graph, for example one between nodes x and y, the probabilities to win in x and y: $p[x]$ and $p[y]$ depend on each other. You need the probability of $p[y]$ to find out whether it is best to stay in x or move to y. You also need $p[x]$ to know whether to move to $p[x]$ or stay in y.

So the probabilities make recurrence relations that might be cyclic. If we only we could get rid of the cycles somehow... Here is an idea: Consider the probability to win in *at most* t steps.

- If t is very large, then the probability to win in *at most* t moves will be a close enough value to the probability to win without the movement restriction. Note that the required value is a floating point variable and we need the relative or absolute error to be smaller than 1e-9.
- If t is 0, we already know the results: It is impossible to win in 0 steps. We can set all probabilities to 0.
- If $t > 0$, we can use the results for $t - 1$. Calculate $p[i][t]$, the probability to win in at most t moves if you have reached i. Consider all your decisions:
  - Move to j. There is a probability w_j to win, a probability l_j to lose and a probability $1 - w_j - l_j$ that nothing happens in this step, but something can happen in a later step, so we need the probability to win in at most $t - 1$ steps: $w_j + (1 - w_j - l_j)p[j][t - 1]$.
  - Stay in i. The formula is the same but replacing j with i.

  We should pick the maximum result among all of those, this finds $p[i][t]$.

This time, the values depend on the values for $t - 1$, so there is no cycle anymore.

Number of moves

We want a number of moves M that is large enough so that the probability is close enough to the probability we want. It shouldn't be too large, because we need $O(nM)$ memory and time. Notice $z = p[i][1]$. When we calculate some $p[j][2]$, we will find $p[j][2] = w_i + (1 - w_i - l_i)z$. Then we might need this value when calculate some $p[k][3] = w_j + (1 - w_j - l_j)p[j][2] = w_j + (1 - w_j - l_j)(w_i + (1 - w_i - l_i)z$. And so and so. This original z will be multiplied by values of the kind $(1 - w_x - l_x)$ $M - 1$ times. Even if those $(1 - w_x - l_x)$ values were as large as possible (0.99). There will be a M for which the product becomes meaningless. Even for $M = 3000$, the value 0.99^{3000} is already much smaller than 1e-9. We can safely use $M = 3000$ as the maximum number of moves and it will also be good enough for the memory complexity.

Code

```
static const int MAX_TURNS = 3000;
double prob[50][MAX_TURNS+1];
double findprob(vector<string> graph, vector<int> winprob, vector<int> loseprob, int Start)
{
    int n = graph.size();
    for (int i = 0; i < n; i++) {
        // 0.0 probability to win in at most zero turns:
        prob[i][0] = 0.0;
    }
    for (int k = 1; k <= MAX_TURNS; k++) {
        for (int i = 0; i < n; i++) {
            prob[i][k] = 0.0;
            for (int j = 0; j < n; j++) {
                if ( graph[i][j] == '1' ) {
                    // can move from i to j.
                    double q = winprob[j] / 100.0 + prob[j][k-1] * (100 - winprob[j] - loseprob[j]) / 100;
                    prob[i][k] = std::max(prob[i][k], q);
                }
            }
        }
    }

    return prob[Start][MAX_TURNS];
}
```

Alternative solutions and additional comments.

There is a interesting discussion at the forums with more approaches and explanations: forum thread.

<Place your comments here>

Next problem: MaxMinTreeGame

Author

By **vexorian**

TopCoder Member

Editorial feedback	Choose
I liked it.	✔
I didn't like it.	✔

Comments (Hide Comments)

As it's in the Wiki, there's a possibility to improve it. It can be language correction, wording improvement or additional explanation in some parts, your additional comments, description of alternative solutions, etc. If you want to improve the wording of editorial writer or correct some language error, please feel free to put your change over the original text. And if you wish to add a comment or describe another approach, there's a section for this at the bottom of each problem.

Before editing, please be sure to check the guidelines.

You can also add a comment in this comment section. When posting a comment thread, make sure to specify the problem you are talking about.

Corrections by: xkirillx and felikjunvianto

Posted by vexorian at Jan 08, 2014 11:00Updated by **vexorian** | Reply To This

In "PairOfStrings" question :

A+C=C+B ==> B is a rotation of A which you have proved for length of C < length of B and A.

Will it remain valid even if length of C > length of B and A ??

Posted by nilmish.iit at Jan 09, 2014 12:53 | Reply To This

Yes, it will valid. For ex,
aaaccccc
ccccbbb

c1 = a1, c2 = a2, c3 = a3
c4 = c1 = a1, c5 = c2 = a2

Thus we can rewrite c as a1 a2 a3 a1 a2 .. and so on
Since b is suffix c, b will be some of rotation of a.

Posted by rotozoom at Jan 09, 2014 13:38 | Reply To This

Yes. And can show it.

Although it is best if we prove that B being a rotation of A is sufficient for there to be a string C of length smaller than n...

Posted by vexorian at Jan 09, 2014 13:39 | Reply To This

In GraphWalkWithProbabilities

The TURNS_NUM will be sufficient to be equal to number of nodes (50 max) if to account the best strategy. When we stay in the same node, the best strategy for the END-node assumes we should stay here until end of the game. This implies we should not pass the same node twice and 50 turns is sufficient to reach fartherst node in graph. Code change will look like

```
for (int k = 0; k < N + 1; k++) {
        for (int i = 0; i < N; i++) {
          if (k == 0) {
            P[i][k] = 0.L; continue;
          } else {
            P[i][k] = winprob[i] / (winprob[i] + looseprob[i]);
          }
          for (int j = 0; j < N; j++) {
            if (i == j || graph[i][j] == '0') continue;
            P[i][k] = max(P[i][k], winprob[j] + passprob[j] * P[j][k-1]);
          }
        }
      }
```

Posted by lev.rumyantsev at Jan 10, 2014 09:27 | Reply To This

PairsOfStrings:
"
B is a rotation of A
Ak=B0,Ak+1=B1,...,An=Bn−k−1

"

Maybe

Ak=B0,Ak+1=B1,...,An-1=Bn−k−1b

because we use 0-index.

Posted by nik_storm_2010 at Jan 12, 2014 09:19 | Reply To This

The Div-1, Level-2 solution doesn't seem to work for the example-4(n=100, k=26). Can someone pls explain why..

Sorry, my bad, a silly mistake.. the soln works fine!

Posted by ujraaja at Jan 18, 2014 04:08Updated by **ujraaja** | Reply To This

SplitIntoPairs:

all the vectors(say A,neg,nonneg) are "int" containers. But while getting max ans min element, we are using "long". Why?

I know the input range is huge. But why aren't we storing in "long" containers itself?

Please help me understand.

Posted by k4v1r4j at Mar 16, 2014 04:42 | Reply To This

Add Comment