Statistics

Match Editorial

Archive
Printable view
Discuss this match

SRM 239
Monday, April 18, 2005

Match summary

tomek strikes again! Out of the three coders who managed to solve all the problems, **tomek** took a comfortable lead of almost 300 points. Ten minutes before the end of the coding phase, he was the only one to have a submission for the 1000 point problem. **Ruberik** lighted the 'matches' with blazing speed and claimed the second spot. **Zis** came in third, more than 100 points behind **Ruberik**. The remaining top ten spots were also taken by reds. The challenge round was quite a trill, especially for the 'MatchCounting' problem, where 151 challenging attempts had been made. As usually, the system tests claimed even more solutions. But if you happen to be in the same room with **natori**, system testing can become pretty boring. With the help of 325 points he gained from challenges, **natori** climbed his way up to the fourth place.

In Division 2, the top performer was **wstfgl**, followed by **Titoz** and **ILoveYou**. The easy problem was a little work this time, but in the end most coders managed to solve it. As for the hard, the scores of those who managed to solve it were quite high, few coders breaking the 900 points barrier.

The Problems

Barbecue

Rate It Discuss it

Used as: Division Two - Level One:

Value	300
Submission Rate	192 / 245 (78.37%)
Success Rate	138 / 192 (71.88%)
High Score	**mouryaravi** for 298.11 points (2 mins 16 secs)
Average Score	200.99 (for 138 correct submissions)

This problem can be reduced to a standard "find the *max* problem", but first you have to build the array. One idea is to give each person a score and then, exclude the person with the highest score. One vote received counts towards exclusion more than any number of votes given, so we can do something like:

```
for (i = 0; i < n; i++) score[i] = 0;
for (i = 0; i < voter.length ; i++)
{
    score[voter[i]]++; score[excluded[i]] += 100;
}
```

Then, we just return the index of the person with the highest score.

```
maxscore = score[0]; out = 0;
for (i = 0; i < n; i++)
{
    if(score[i] > maxscore)
    {
        maxscore = score[i]; out = i;
    }
}
return out;
```

ATaleOfThreeCities

Rate It Discuss it

Used as: Division Two - Level Two:

Value	500
Submission Rate	144 / 245 (58.78%)
Success Rate	112 / 144 (77.78%)
High Score	**zylum** for 463.42 points (8 mins 6 secs)
Average Score	350.78 (for 112 correct submissions)

Used as: Division One - Level One:

Value	250
Submission Rate	210 / 210 (100.00%)
Success Rate	188 / 210 (89.52%)
High Score	**radeye** for 247.12 points (3 mins 4 secs)
Average Score	217.27 (for 188 correct submissions)

The first thing to do is to determine the optimal distance of connecting a city with another. In order to do this, you compute the distance between every subway station in one city and every subway station in the other. Then, you keep the one that is minimal. Here is the function that computes the 'distance' between two given cities, written in Java:

```
double go(int[] a, int[] b, int[] c, int[] d)
{
    double ret = 1e9;
    for (int i= 0; i<a.length; i++)
    {
        for (int j = 0; j<c.length; j++)
        {
       double d1 = a[i] - c[j];
       double d2 = b[i] - d[j];
       ret = Math.min(re , Math.sqrt(d1 * d1 + d2 * d2));
   }
   }
   return ret;
}
```

As there are three cities, you must find the optimal connecting distance between each of them:

```
double d1 = go(ax,ay,bx,by);
double d2 = go(ax,ay,cx,cy);
double d3 = go(bx,by,cx,cy);
```

Then, you compare the three distances already found to determine the two whose sum is minimal.

```
return Math.min(Math.min(d1+d2,d1+d3),d2+d3);
```

Some of the solutions that failed on this problem didn't find this minimum correctly when two of the distances were equal.

DivisibilityCriteria Rate It Discuss it

Used as: Division Two - Level Three:

Value	1000
Submission Rate	44 / 245 (17.96%)
Success Rate	29 / 44 (65.91%)
High Score	**prayanks** for 959.73 points (5 mins 52 secs)
Average Score	733.71 (for 29 correct submissions)

In this problem you are told a few things you probably know about the usual divisibility properties of a number when divided by 2, 3 and 7. Then you are asked to find a general template for quickly assessing whether or not a number is divisible with a given prime, P. This problem may look daunting at a first glance, but as it can be seen from the first example, the construction of such a 'criterion' is not that difficult and you can work your way out from there. But there is an even simpler and more elegant solution:

```
int ret[] = new int[N], mult = 1;
for (int i = N-1; i ≥ 0; i--, mult = (mult * 10) % P)
{
    ret[i] = mult;
}
return ret;
```

MatchCounting Rate It Discuss it

Used as: Division One - Level Two:

Value	500
Submission Rate	151 / 210 (71.90%)
Success Rate	80 / 151 (52.98%)
High Score	**Ruberik** for 473.63 points (6 mins 46 secs)
Average Score	279.96 (for 80 correct submissions)

This problem proved to be a little unusual, since it required a different set of skills. But as with all the problems, it is helpful to have an overall look first.
A few observations can be made:
- not all the digits are being used! Take for example the numbers 28 and 58. Both 2 and 5 can be represented using 5 sticks, so if you are able to represent the number 2, you are also able to represent the number 5. After a short analysis, you find out that only the numbers 1, 2, 6, 8 and 0 are going to be used.
- numbers 1, 2, 6 and 8 always appear in increasing order from left to right. Take for example the numbers 12 and 21. If 21 cannot be represented, neither can 12.
- the number to return may have up to 19 digits. A digit can use at most 7 matches and you can have 128 matches at your disposal.
Keeping these in mind, an optimized recursive solution should work.

It is also possible to solve this problem using dynamic programming. You can try all the values for the last digit (call it x) and then, you have n - matches[x] left for the rest of the number. You now try to minimize the number that uses all the n- matches[x] matches.

Another interesting approach is to consider a number X and determine the number of matches needed to be able to represent all the numbers between 1 and X. If there are n or less, you can increase X, otherwise you should decrease it. This leads us to a binary search

algorithm, see Cosmin.ro's implementation.

The brute force method times out very quickly, but it may prove very useful as once you have a few more results, you can easily look for a pattern and the problem can be reduced to something like this:

```
long nr=0;

digits = n / 7 + 1;
prefix = n % 7;
if (prefix == 0) nr = 10;
if (prefix == 1) nr = 1;
if (prefix == 2) nr = 200;
if (prefix == 3) nr = 20;
if (prefix == 4) nr = 2;
if (prefix == 5) nr = 6;
if (prefix == 6) nr = 8;
while(nr < 1000000000000000000L) nr = nr * 10 + 8;
size = 19;
while (size > digits) { nr /= 10; size--; }

return nr;
```

Don't forget that intuition is a powerful weapon... There is no guarantee that a very thoughtful dynamic programming approach is less error prone than a good guess!

HiddenTriangles

Rate It Discuss it

Used as: Division One - Level Three:

Value	1000
Submission Rate	12 / 210 (5.71%)
Success Rate	4 / 12 (33.33%)
High Score	**tomek** for 706.19 points (20 mins 10 secs)
Average Score	548.50 (for 4 correct submissions)

A good observation to make is that with any three given segments, you can form at most one triangle. From this, the first idea that comes to mind is to iterate through all the possible sets of three segments and count whether or not it is possible to make a triangle. To do this, you take all the three combinations of two out of three segments and check their point of intersection. If you get an intersection in all the cases and the intersection points do not coincide, a triangle has been found!
But ... it may happen that you need more than three segments to find a certain triangle. This is the trickiest part of the problem and one way to solve this is to 'extend' each segment to its maximum possible length. This means to merge the segments that have the same orientation and share at least one point (two segments may have one segment in common or one may even be contained in the other). This merging process has to be applied until no more merges can be made, otherwise you may miss the cases when there are a lot of segments lying on the same 'line'. Also, make sure to delete the segments that have been used for merging, or you may risk counting the same triangle more than once.

Because of the relatively small constraints, it is also possible to consider a 400*400 grid and take every possible segment of length 1 to check whether it can be drawn or not from the initial given segments. This approach is feasible as you know from the problem statement that a segment can either be parallel or form a 45 degree angle with one of the axes. There are a lot of ways to go from here, see tomek's implementation for a clean solution.

By **supernova**
TopCoder Member

'puff piece' on the 'wisdom of crowds' and how all we need to do is post...Read More

View More

Get Connected

Your email address Submit