Statistics

SRM 303
Thursday, May 18, 2006

Match summary

With 843 registrants, this single round match again had a healthy amount of participants, even though no money prizes were at stake. Unfortunately, some of the top-ranked competitors were missing out, as they were busy winning the internet problem solving contest. Congratulations to **reid**, **tomek**, and **John Dethridge** for this impressive feat! (and in the individual division the top 3 were TopCoders as well: congrats to **Petr**, **antimatter**, and **liympanda**.)

In this SRM the division 1 coders faced a tricky-to-code 250, a tricky-to-see 500 and a pretty straightforward 1,000. The division 2 coders had the tricky 250 of division 1 as the medium and an interesting problem concerning a combination of primes and palindromes as the 1,000. The tricky 500 of division 1 lured a large number of coders (including yours truly) into coding a solution taking the wrong (greedy) approach. As always, the challenge phase is exciting with this kind of problem. With 158 challenged problems in division 1, this SRM ranks within the top 10 of matches with the most challenges.

In the end, **marek.cygan** took home the win. **Burunduk1** came in second and, with 5 challenges on the 500, **Eryx** took the bronze. In division 2, newcomer **wojtekt** secured the win with a fast 1,000 and 3 correct challenges. **marting** was a close second, with **Protean** coming in third.

The Problems

Segments

Rate It Discuss it

Used as: Division Two - Level One:

Value	250
Submission Rate	239 / 471 (50.74%)
Success Rate	44 / 239 (18.41%)
High Score	**dubna** for 238.13 points (6 mins 24 secs)
Average Score	150.68 (for 44 correct submissions)

Although coding a few 'if's' to handle the different configurations of line segments may seem tempting at first, boundary cases make such an approach hard to get right (as is reflected by the success rate of this problem). In this problem, one such a boundary case consists of a line segment that actually is just a point, which lies on another segment. Many if-based solutions return "SEGMENT" in this case, instead of "POINT". An easier solution is to write code that naturally handles all possible cases.

With a few min's and max's we can easily determine the overlapping region of the two line segments, and then see if this is a point, a segment, or if there is no overlap.

In code:

```
int left = max(min(s1x1, s1x2), min(s2x1, s2x2));
int right = min(max(s1x1, s1x2), max(s2x1, s2x2));
int top = max(min(s1y1, s1y2), min(s2y1, s2y2));
int bottom = min(max(s1y1, s1y2), max(s2y1, s2y2));

if(top > bottom || left > right)
  return "NO";

if(top == bottom && left == right)
  return "POINT";

return "SEGMENT";
```

SpiralNumbers

Rate It Discuss it

Used as: Division Two - Level Two:

Value	500
Submission Rate	157 / 471 (33.33%)
Success Rate	44 / 157 (28.03%)
High Score	**marting** for 433.23 points (11 mins 31 secs)
Average Score	290.09 (for 44 correct submissions)

Used as: Division One - Level One:

Value	250

Submission Rate	279 / 310 (90.00%)
Success Rate	178 / 279 (63.80%)
High Score	**gevak** for 244.92 points (4 mins 6 secs)
Average Score	161.42 (for 178 correct submissions)

This problem asks for a simple simulation of a spiraling sequence of numbers. However, with a maximum number of steps of more than 2 billion, a naive solution will time out. Observing that the lowest odd square number lower than N gives the last number of the last completed square can be used to start the search at a much higher number. This reduces the runtime from O(n) to O($n^{1/2}$), which easily runs in time.

Maybe an even simpler approach is to simulate the spiral using bigger steps than 1, as the spiral takes 1 step right, then 1 step down, then 2 steps left, then 2 steps up, then 3 steps right, etc. In pseudocode:

```
cur = 1
direction = right
size = 1
x = 0, y = 0

while(true) {
  repeat twice {
    if(N - cur > size) {
      update x and y with (size) steps in the current direction
      direction = rotate(direction)
      cur += steps
    }
    else
    {
      update x and y with (N-cur) steps in the current direction
      return "(x,y)";
    }
  }
  size ++;
}
```

PrimePalindromic

Rate It Discuss it

Used as: Division Two - Level Three:

Value	1000
Submission Rate	18 / 471 (3.82%)
Success Rate	8 / 18 (44.44%)
High Score	**wojtekt** for 711.42 points (19 mins 52 secs)
Average Score	565.63 (for 8 correct submissions)

This problem can be solved by simply checking all integers in the requested range one-by-one whether they are prime palindromic. To check this for a certain integer, the list of prime factors of that integer needs to be determined, and then one can simply try every permutation of these factors to see if they form a palindrome. Checking all permutations may seem slow, especially with a maximum number of prime factors that can be 13 (for 8192=2^{13}). But as all these factors are the same, there is only one true permutation instead of 13!. The number with the maximum number of permutations is 7560 (=2^3*3^3*5*7), and this number has only 1120 permutations. With such small numbers of permutations, the factor-finding and palindrome-checking routine don't even need to be sophisticated to let a solution run in time.

Knights

Rate It Discuss it

Used as: Division One - Level Two:

Value	500
Submission Rate	128 / 310 (41.29%)
Success Rate	16 / 128 (12.50%)
High Score	**Ying** for 433.60 points (11 mins 29 secs)
Average Score	312.28 (for 16 correct submissions)

With knights attacking eachother on a chess board, a graph representation immediately springs to mind. We're asked to find a minimal set of nodes (knights) to remove so that no edges (attacking knights) in the graph remain. This is exactly the vertex cover problem. But hey, isn't this the same thing as the 250 problem from the wildcard round of the last TCO? Yes, almost, but in the wildcard problem the graph was guaranteed to be a tree. For a tree, a simple solution can repetitively pick nodes that are attached to a tree leaf, and this is guaranteed to yield a minimal cover. We could try the same thing in this problem too, but in most cases, we'll end up with a connected component, which is hard to handle. As the referenced wikipedia page mentions, minimal vertex cover is an NP-complete problem in general graphs. It is equivalent to a maximum matching in the same graph (as the maximum matching automatically selects the minimum number of edges from which one of both attached nodes should be removed). This translation is the key to this problem, together with the observation that the graph in this problem is by no means general. A knight that stands on a white square can only attack knights on black squares and vice versa. This means that the graph is bipartite. Therefore, the maximum matching can be easily calculated. See the topcoder algorithm tutorial about this subject for more information ([part 1][part 2]).

FourBears

Rate It Discuss it

Used as: Division One - Level Three:

Value	1000
Submission Rate	13 / 310 (4.19%)

Success Rate	6 / 13 (46.15%)
High Score	**Eryx** for 607.16 points (26 mins 49 secs)
Average Score	548.59 (for 6 correct submissions)

Probably the main reason why so few people solved this 1000 is the 500 took many people too much time. The question is how to search all possible paths the four bears could take to get together. It is easy to see that once two bears are in the same square, they will always travel together, because then they have to clear fewer passages. Therefore, all possible solutions are checked when we try to join the left two bears at all possible locations, and join the right two bears at all possible locations. (Except in some cases, it is better to join the top two bears and bottom two bears, and let those two pairs meet. This exception was already demonstrated by test case #2.) In pseudocode:

```
result = MAX
foreach meet1 in squares:
  foreach meet2 in squares:
    value  = distance(bear1, meet1) + distance(bear2, meet1) +
             distance(bear3, meet2) + distance(bear4, meet2) +
             distance(meet1, meet2) + correction(meet1, meet2);
    value2 = distance(bear1, meet1) + distance(bear3, meet1) +
             distance(bear2, meet2) + distance(bear4, meet2) +
             distance(meet1, meet2) + correction(meet1, meet2);
    result = min(min(value,value2, result)

return result;
```

The "correction" function mentioned above is needed to make sure that when either meet1 or meet2 are rocks, the clearing of those rocks isn't counted double. To have all the distance(x,y) functions ready, we do a Floyd-Warshall. The initial state of the distance matrix must be carefully constructed so that the constraint that bears can't move vertically on the first and last columns is taken care of.

By **mathijs**
TopCoder Member

Twitter

Follow

Recent Blog Posts Updated

Apr 23 @timmhicks – Tim Hicks Happy Hump Day topcoders! We are excited to announce that we will be releasing a new look for the very popular /tc by...Read More

Apr 23 Do you ever find yourself hitting “send” on an email and wondering if it’ll arrive in the recipient’s inbox? Sending email has become so ubiquitous, simple and...Read More

Apr 22 @ClintonBon – Clinton Bonner We know what you’re thinking. Great, another ‘puff piece’ on the ‘wisdom of crowds’ and how all we need to do is post...Read More

View More

About topcoder

The topcoder community gathers the world's experts in design, development and data science to work on interesting and challenging problems for fun and reward. We want to help topcoder members improve their skills, demonstrate and gain reward for their expertise, and provide the industry with objective insight on new and emerging technologies.

About Us

Get Connected

Your email address Submit