Member Count: 672,344 - August 26, 2014 [Get Time]

Hello, faijur | Logout

My TopCoder

Competitions
Overview
Copilot Opportunities
Design
Development
UI Development
QA and Maintenance
CloudSpokes
Algorithm
High School
The Digital Run
Submit & Review

TopCoder Networks
Events
Statistics
Tutorials
Forums
Surveys
My TopCoder
Help Center
About TopCoder

Member Search:
Handle: Go
Advanced Search

Dashboard > TopCoder Competitions > ... > Algorithm Problem Set Analysis > SRM 601 Search

TopCoder Competitions

SRM 601

Added by [[rng_58]] , last edited by vexorian on Dec 28, 2013 (view change)
Labels: (None) EDIT

Single Round Match 601
Sunday, December 22nd, 2013

Match summary

SRM 601, or the match right between SRM 600 and the last match of the year. It might not be the most special match this year, or the most special match this month, but it was a rare chance to have some topcoder fun on a Sunday. The problem set was provided by **Witaliy**. The focus seems to have been dynamic programming, as the 3 hardest problems were all dynamic programming ones with interesting twists. In division 1, coders were welcome with an ad hoc problem that needed some good observations to work. **tourist** broke the speed barrier in this problem, solving it in just a couple of minutes. The medium problem was both an easy dynamic programming problem and a complicated dynamic programming problem. It was all up to the coders' creativity at coming up with clever ways to reduce a problem to a simpler one. No surprise that a coder like **Petr**, who is very good at exactly that, solved the problem in only 8 minutes. The hardest problem was also a dynamic programming one, this one needed a magic trick to get rid of two complexity factors. Yet the problem was another triumph for **Petr**, even though this time other 9 coders solved it as well. Between the two speed records and the still very fast solution to 250, **Petr** won the division with a very comfortable lead of over 200 points. This is **Petr**'s 100-th division 1 win. This is also the fifth time **Petr** is mentioned in this commentary. Turns out SRM 601 was special after all!

Congratulations also to **al13n** (2nd place), **ikatanic** (3rd place) and **Hasan0540** (Division 2 winner).

The Problems

WinterAndMandarins | WinterAndCandies | WinterAndReindeers | WinterAndPresents | WinterAndSnowmen | WinterAndShopping

WinterAndSnowmen

Rate It Discuss it

Used as: Division One - Level Two:

Value	500
Submission Rate	72 / 767 (9.39%)
Success Rate	55 / 72 (76.39%)
High Score	**Petr** for 458.89 points (8 mins 39 secs)
Average Score	279.61 (for 55 correct submissions)

Simple and slow dp

Let's begin with a dynamic programming approach that is too slow but allows us to understand some ideas that will be useful in the final approach.

We can describe the final state by two numbers X and Y, where X is the xor value of the numbers in the first set and Y is the xor value of the numbers in the second set. We need $(X < Y)$. Initially, the xor values are $x = y = 0$ and we have to assign all positive numbers: $t \leq T = \max(N, M)$ to one (or none) of the sets. Consider a function $f(t, x, y)$, that solves the following sub-problem: We can use positive numbers $(1 \leq t)$, the initial xor value of the first set is x and the initial xor value of the second set is y. How many ways exist to assign the t integers to the sets such that the final xor values X and Y follow $X < Y$?:

- Base case: If $t = 0$, then there are no more numbers to use. The xor values, x and y will not change. If $(x < y)$ then there is exactly one way to do it (doing nothing) the result is 1. Else it is impossible to do it and the result is 0.
- Otherwise, we can decide what to do with number t:
  - If $(t \leq N)$, we can add t to the first set, which would change the xor value into $x = x \oplus t$(Where $\oplus$ is the xor operation). To count the number of ways to assign the remaining $t - 1$integers we can call $f(t-1, x \oplus t, y)$
  - If $(t \leq M)$ we can add t to the second set , its xor value becomes $(y \oplus t)$ and there are $t - 1$integers left: $f(t-1, x, y \oplus t)$
  - We can also choose to leave the integer outside both sets. The available integers are decremented: $f(t-1, x, y)$

  $f(t, x, y)$would be equal to the sum of those three partial results.

The recurrence relation is acyclic because it always calls $f(t-1, \ldots, \ldots)$. There are $O(\max(N, M))$ values for t. If the maximum number of bits of N and M is b, then there are $O\left(2^b\right)$ different values for x and y (Any combination of 1s and 0s is possible for the xor). Note that b is $O(\log(\max(N, M)))$, so it turns out that there are $O(\max(N, M))$ values for x and y. The total number of states for this recurrence is $O\left(\max(N, M)^3\right)$, which is too slow for $M, N \leq 2000$ We need something better.

Smaller condition

It would be great if we could remove one of the factors so that we have a $O\left(\max(N, M)^2\right)$ solution. The problem right now is the $X < Y$ which forces us to remember two numbers in the state. Here is a key observation: If $X < Y$, then the binary representations would be like this:

```
          i
Y = abcdef1??..?
X = abcdef0??..?
```

There should be a bit position i such that the two numbers have equal bits in the higher bit positions, Y has 1 in the i-th position and X has 0 in that position. The remaining bit positions can have any value.

How about, we try all candidates for position i, there are only $O(\log(\max(N, M)))$ of them. For each of those candidates count the number of ways to assign the numbers to sets such that the xor values X and Y follow the property described above.

A single xor

At first the above condition seems unhelpful and more complicated than the initial problem. The second key observation is to think about the xor between Y and X:

```
            i
  Y = abcdef1??..?
  X = abcdef0??..?
X^Y = 0000001??..?
```

There are now two conditions:

- The xor value $X \oplus Y$ must follow the pattern: 0000...1. It must have all zeros until the i-th bit which should have 1.
- The i-th bit of X must be 0.

This is very nice because applying a xor to one of X or Y will also apply the xor to $X \oplus Y$. Consider the current xor value $x \oplus y$ and we decide to apply a xor to x $(x = x \oplus t)$.

$$(x \oplus t) \oplus y = x \oplus t \oplus y = x \oplus y \oplus t = (x \oplus y) \oplus t$$

That's because the commutative and associativity properties of the Xor operation.

This will allow us to solve the problem using a simple dynamic programming. Consider $f(t, z, b)$ where t is the number of positive integers available. z is the current xor $x \oplus y$ and b is the value of the i-th bit of x.

- Base case: If $t = 0$, then there are no more numbers to use. Result is 1 if and only if z contains only zeros for bit positions higher than i and 1 for bit i. $b = 0$.
- Otherwise, decide what to do with number t:
  - If $(t \leq N)$, we can add t to the first set, which would change the xor value into $x = x \oplus t$ Which in place means that $z = z \oplus t$ Note also that the **i**-th bit changes. Let k be the **i**-th bit value in t, then $b = b \oplus k$. We can solve sub-problem by calling $f(t-1, z \oplus t, b \oplus k)$
  - If $(t \leq M)$, we can add t to the second set , $(z = z \oplus t)$ and there are $t-1$ integers left. b does not change: $f(t-1, z \oplus t, b)$.
  - We can also choose to leave the integer outside both sets.: $f(t-1, z, b)$.

  $f(t, z, b)$ would be equal to the sum of those three partial results.

Some more optimizations

Note that we must do that $O\left(\max(N, M)^2\right)$ dynamic programming once for each bit position **i**. This gives us a total complexity: $O\left(\max(N, M)^2 \log(\max(N, M))\right)$, which can be very tight. We should make sure to optimize as much as possible.

For a given **i**, the values of the bit positions smaller than **i** do not matter. We can actually ignore them altogether and remove the first **i** bits from all numbers in the dp. This means that the total number of bits in the xor value **z** will be reduced by **i**. This actually reduces the complexity. For example, without this optimization, we will estimate 11 * 2001 * 2048 * 2 states (11 bit positions, 2001 integers, 2048 variations of z, 2 bits for b). With the optimization we have: 2001 * 2048 * 2 + 2001 * 1024 * 2 + ... + 2001 * 1 * 2 = 2001 * 2 * 4096. In effect, the complexity becomes $O\left(\max(N, M)^2\right)$ with a constant factor.

Code

The following c++ code takes 0.3 seconds in the worst case:

```
static const int MOD      = 1000000007;
static const int MAX_BITS = 11;
static const int MAX_N    = 2000;
long dp[MAX_N + 1][1 << MAX_BITS][2];

int i, N, M;

int rec(int t, int z, int b)
{
    long res = dp[t][z][b];
    if (res == -1) {
        if (t == 0) {
            // the limit
            if ( (z == 1) && (b == 0) ) {
                res = 1;
            } else {
                res = 0;
            }
        } else {
            // xor number t to neither A or B:
            res = rec(t-1, z, b);
            // xor number t to A:
            if (t <= N) {
                res += rec(t-1, z ^ (t >> i), b ^ ( (t >> i) & 1 ) );
            }
            // xor number q = p+1 to B:
            if (t <= M) {
                res += rec(t-1, z ^ (t >> i), b);
            }
        }
        res %= MOD;
        // save it
        dp[t][z][b] = res;
    }
    return (int)res;
```

Alternative solutions and additional comments.

<Place your comments here>

Next problem: WinterAndShopping

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.

Posted by vexorian at Dec 24, 2013 10:34Updated by **vexorian** | Reply To This

Hi,

Please, could you tell me whether this style of editorials where each problem is on a separate page is TC default, or it is how you like to design editorials. Thanks.

Posted by boba5551 at Dec 27, 2013 20:59 | Reply To This

For the Java brute force algorithm in WinterAndMandarins, is the time complexity up to O(N^3)?

Posted by kctong529 at Dec 24, 2013 13:23Updated by **kctong529** | Reply To This

Hi! In the Div 1 250, there is an O(N) approach, where N is the size of the apple/orange vector. In essence, we start with the maximum possible answer and then remove all the cases which are not possible. We can skip the O(m) loop as the manner in which the values get subtracted for each bag are in an arithmetic progression. So, we can just use a formula instead of the O(m) loop. This solution passes the system tests taking 0.022 seconds for the worst case. I just thought readers should know.

```
class WinterAndPresents {
public:
        long long getNumber( vector <int> apple, vector <int> orange ) {
                long long res = 0;
                long long lim = apple[0] + orange[0], n = apple.size();
                for (long long i = 0; i < n; ++i)
                        lim = min(lim, 1LL * apple[i] + orange[i]);

                res = (lim * (2LL * (n + 1) + (lim - 1LL) * n)) / 2LL;
                for (long long i = 0; i < n; ++i) {
                        if (lim > orange[i])
                                res -= ((lim - orange[i]) * (lim - orange[i] + 1)) / 2I
                        if (lim > apple[i])
                                res -= ((lim - apple[i]) * (lim - apple[i] + 1)) / 2LL;
                }
                return res;
        }
};
```

Posted by zeforce at Dec 25, 2013 15:05Updated by **zeforce** | Reply To This

I have also worked out this solution, and it's not that complicated.

I don't know how to insert math formulas or code snippets here; I put it on this gist, and my derivation is in the comments:

https://gist.github.com/tomtung/8254466

I think the key is the following. Observe that we need to sum over both x (from 1 to M) and x (from i from 1 to N). We put the sum over x outside to loop through, so that the sum over x inside can be computed analytically using arithmetic progressions.

Posted by tomtung at Jan 04, 2014 23:08 | Reply To This

In Div 1 250 Analysis, I think there is a typo: m_O = Σmin(orangesi,X).

Posted by pp2441 at Dec 26, 2013 10:52 | Reply To This

On WinterAndSnowmen:
There is possible typo: "communitativity" property => "commutative" property.

1. I could not find "edit" button to edit...

Posted by yoshiokatsuneo at Dec 27, 2013 16:05 | Reply To This

On WinterAndPresents:

I'm just not getting this problem. Take one of the examples provided:

(2, 2, 2)
(2, 2, 2)
Returns: 16

So I can pick X to be 1, 2, 3 or 4.

If I pick 1, then I pick one piece of fruit from each bag. I have three pieces of fruit with which to construct gifts. So I could potentially have:

- 3 apples
- 2 apples, 1 orange

- 1 apple, 2 oranges
- 3 oranges

That means the following gifts are possible:

apples	oranges
0	1
1	0
0	2
1	1
2	0
0	3
1	2
2	1
3	0

for a total of 9. But of course the selection of 1 for X was arbitrary, I could have picked 2 and by the same procedure arrived at a total of 27 possible gifts.

Among other things, I am confused by the statement "you will choose any positive integer X such that there are at least X pieces of fruit in each bag". However, the choice of X makes a difference to the result, does it not?

I must be looking at the problem description incorrectly. Can anyone enlighten me? Thanks in advance.

Posted by ellipsis17 at Dec 27, 2013 19:59 | Reply To This

Formal:
You want to count the total number of pairs (O, A) such that there is one integer X for which the following condition can be true:

- For each (i < n):
  - takenOranges[i] + takenApples[i] <= X
  - takenOranges[i] <= orange[i]
  - takenApples[i] <= apple[i]
- Sum of all takenOranges[i] = O
- Sum of all takenApples[i] = A

—
With X = 1, the following are possible: (O = 0, A = 3), (O = 1, A = 2), (O = 2, A = 1), (O = 3, A = 0).

For X = 2: (O = 0, A = 6), (O = 1, A = 5), (O = 2, A = 4), (O = 3, A = 3), (O = 4, A = 2), (O = 5, A = 1), (O = 6, A = 1).

For X = 3: (O = 3, A = 6), (O = 4, A = 5), (O = 5, A = 4), (O = 6, A = 3)

For X = 4: (O = 6, A = 6)

—
In total, there are 16 pairs.

Posted by vexorian at Dec 27, 2013 20:50 | Reply To This

Thanks, that makes sense.

Posted by ellipsis17 at Dec 27, 2013 22:15 | Reply To This

The solution for Div 2 1000 times out with java if the dp array is defined as dp[2501][2501][2] .It works only if array is defined as dp[2][2501][2501]

Posted by agastya at Dec 29, 2013 04:14 | Reply To This

I like that you add python code.

Posted by vicfred at Jan 05, 2014 21:51 | Reply To This

Add Comment