



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 603

TopCoder Competitions

SRM 603

View

Attachments (4)

Info

Browse Space

Added by [[rng_58]], last edited by vexorian on Jan 11, 2014 ([view change](#))Labels: (None) [EDIT](#)

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](#)

PairsOfStrings

Rate It

Discuss it

Used as: Division One - Level Two:

Value	500
Submission Rate	121 / 610 (19.84%)
Success Rate	103 / 121 (85.12%)
High Score	Blue.Mary for 484.75 points (5 mins 4 secs)
Average Score	276.54 (for 103 correct submissions)

B is a rotation of A

The described condition, that a string C exists such that: $A + C = C + B$ is usually a way to say that B must be a rotation of A . To verify this, assume that C has length k :

- $A_0 = C_0, A_1 = C_1, \dots, A_{k-1} = C_{k-1}$
- $A_k = B_0, A_{k+1} = B_1, \dots, A_n = B_{n-k-1}$
- $B_{n-k} = C_0, B_{n-k+1} = C_1, \dots, B_{n-1} = C_{k-1}$

It follows that: $A_{k+i} = B_i$ for each $i < n - k$ and that $B_{n-k+i} = A_i$ for $i < k$. So for the first k elements of B we have $B_i = A_i$ and for the remaining $n - k$ we have: $B_i = A_{i-(n-k)}$. This means that B is a rotation of A .

Counting the pairs

For the given n, k , count the number of pairs of strings (A, B) of length n using an alphabet of k characters such that B is a rotation of A .

One approach is to count the number of rotations B for each string A and add the totals together. There are too many strings A , but we can simplify this by separating A in groups such that each give a specific

number of rotations. Then you just need to calculate the number of rotations for a string that belongs to the group and the number of strings in the group.

The number of distinct rotations of a string depends on it being made of repetitions or not. For example, the number of rotations of string "ABCD" is 4: "ABCD", "BCDA", "CDAB" and "DABC" but the number of distinct rotations of "ABAB" is only 2: "ABAB" and "BABA". More so, a string like "BBBB" has only 1 rotation.

We can generalize, if string A is equal to a string X of d characters repeated $\frac{n}{d}$ times ($A = X + X + \dots + X$), then the number of rotations B is exactly d . So the algorithm idea goes as follows: For each d count the number of strings A that consist of the same d characters repeated $\frac{n}{d}$ times, multiply that result by d and the sum of these results is the total number of pairs.

Note that d must be a divisor of n , the divisors of $n \leq 1000000000$ can be extracted in $O(\sqrt{n})$ time with trial division.

The remaining challenge is to, given d , count the number of strings A composed of $\frac{n}{d}$ repetitions of a string of d characters.

Given d

Count the number of A s equal to $\frac{n}{d}$ repetitions of a string containing d characters. For each of d choices, we have k options. So maybe the result is: k^d ? Though it certainly finds strings with repetitions, there is an issue. Imagine $n = 8, d = 2, k = 2$: There are $k^d = 2^2$ ways to pick d characters: AA, AB, BA, BB. This translates to strings: AAAAAAAAAA, ABABABAB, BABABABA and BBBB BBBB. However, we shouldn't count AAAAAAAAAA and BBBB BBBB, because they are really strings that contain 8 repetitions ($d = 1$). We should count the strings whose *minimum* number of repetitions is $\frac{n}{d}$.

The solution to this is to notice: If we use the k^d formula, we will find the sum of results for all $d' \leq d$ which is a divisor of d . If we already knew the results for previous d' , we could just subtract: $k^d - s$, where s is the sum of results for all $d' < d$. This will find the exact result for a given d . All that we need is to process the divisors d in increasing order and save their results in a table. This needs $O(d)$ memory and $O(d^2)$ time.

Maximum number of divisors

The $O(d^2)$ complexity means that we should ensure that d will not be too high. What is the maximum number of divisors for a number $n \leq 1000000000$? Note that the $O(\sqrt{d})$ complexity applies only to the algorithm we normally use to extract all divisors and not to the number of divisors itself.

The number of divisors of a number can be calculated by first finding the prime factorization. If the prime factorization is $p_0^{r_0} \dots p_k^{r_k}$, then the number of divisors is: $(r_0 + 1)(r_1 + 1) \dots (r_k + 1)$. Thus we should maximize the product $(r_0 + 1)(r_1 + 1) \dots (r_k + 1)$. From this realization you can find some numbers below 1000000000 that have 700+ divisors using exponents of $2^3 \cdot 3^5 \cdot 5^7 \cdot 7^{11} \cdot 13^{17} \dots$. A possible way to find the maximum number of divisors is brute force for the exponents stopping when the exponents get too large for 1000000000. Another idea is to use some theory. It turns out that numbers that have a notable number of divisors are called [highly composite numbers](#) and we can find a [table](#) that will tell us that the worst number smaller than 1000000000 is: 735134400 with 1344 divisors. 1344*1344 is not too high a value so we are free to use the algorithm described above.

Code

Problems that ask you to return the result modulo 1,000,000,007 (or a similar number) usually do so to allow us to solve problems that have large results without actually using those large numbers in calculations. We can handle these numbers with modular arithmetic. Read [this recipe](#) for more info.

```

const int MOD = 1000000007;

// extract divisors of n:
vector<int> getDivisors(int n)
{
    vector<int> div;
    for (int i = 1; i*i <= n; i++) {
        if (n % i == 0) {
            div.push_back(i);
            if (n / i != i) {
                div.push_back(n / i);
            }
        }
    }
    sort(div.begin(), div.end());
    return div;
}

// Find x raised to the y-th power modulo MOD
long modPow(long x, int y)
{
    long r = 1;
    while (y > 0) {
        if (y & 1) {
            r = (r * x) % MOD;
        }
        x = (x * x) % MOD;
        y /= 2;
    }
    return r;
}

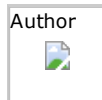
int getNumber(int n, int k)

```

Alternative solutions and additional comments.

<Place your comments here>

Next problem: [SumOfArrays](#)



By **vexorian**

TopCoder Member

Editorial feedback	Choose
I liked it.	<input checked="" type="checkbox"/>
I didn't like it.	<input checked="" type="checkbox"/>

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 11:00 Updated by vexorian | [Reply To This](#)

In "PairOfStrings" question :

$A+C=C+B \implies B$ is a rotation of A which you have proved for length of $C < \text{length of } B \text{ and } A$.

Will it remain valid even if length of $C > \text{length of } B \text{ and } A$??


 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
 $A_k=B_0, A_{k+1}=B_1, ..., A_n=B_{n-k-1}$

"

Maybe


$A_k=B_0, A_{k+1}=B_1, ..., A_{n-1}=B_{n-k-1}b$

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](#)

[Home](#) | [About TopCoder](#) | [Press Room](#) | [Contact Us](#) | [Privacy](#) | [Terms](#)
[Developer Center](#) | [Corporate Services](#)

Copyright TopCoder, Inc. 2001-2014