



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 604](#)

TopCoder Competitions



SRM 604

[View](#)

[Attachments \(8\)](#)

[Info](#)

[Browse Space](#)

Added by [[rng_58]], last edited by vexorian on Jan 28, 2014 ([view change](#))

Labels: (None) [EDIT](#)



Single Round Match 604

Wednesday, January 11th, 2014

[Archive](#)
[Match Overview](#)
[Discuss this match](#)

Match summary

It is the 604-th edition of TopCoder's Single Round Matches. This time the problem set was authored by **cg4ever**. This match was full of challenging problems. Division 1 easy needed some nice ad hoc maths, it is always impressive to watch coders like **Eryx** solve these messy problems in less than 2 minutes. The medium problem was a dynamic programming one that required good observations and analysis if you don't want it to get overcomplicated. **Petr** was significantly faster than everyone else in this problem precisely because he found the exact observation that simplifies everything quite quickly. The 1000 points problem was a beast of its own, requiring unusual Computational Geometry/Collision detection theory. This problem was **Egor's** success, winning just a bit more points in this problem than **Petr**. Also notable, we could catch **Gassa** submitting the first successful python solution for a division 1 hard problem. The consistently good performance in all three problems scored **Petr** yet another division win. Second place went to **Zlobober**, also with nice consistency. Third place, still with over 1000 points was **Egor's** to get. Division 2 was won by former gray coder **HwhiteTooth**, congratulations and welcome to division 1.

The Problems

[FoxAndWord](#) | [PowerOfThreeEasy](#) | [FoxConnection2](#) | [PowerOfThree](#) | [FoxConnection](#) | [FamilyCrest](#)

FoxConnection2

Used as: Division Two - Level Three:

Value	1000
Submission Rate	28 / 1435 (1.95%)
Success Rate	7 / 28 (25.00%)
High Score	Hwhitetooth for 729.61 points (18 mins 49 secs)
Average Score	572.37 (for 7 correct submissions)

Count the number of ways to pick k cities such that they are connected. The cities make a tree. It follows that you need to count the number of sub-trees of k nodes from a larger tree. This problem is a classical one that possibly already has explanations out there, but let's solve it with a dynamic programming that will also help explain the division 1 version.

If you are new to dynamic programming, try **dumitru's** tutorial: [Dynamic](#)

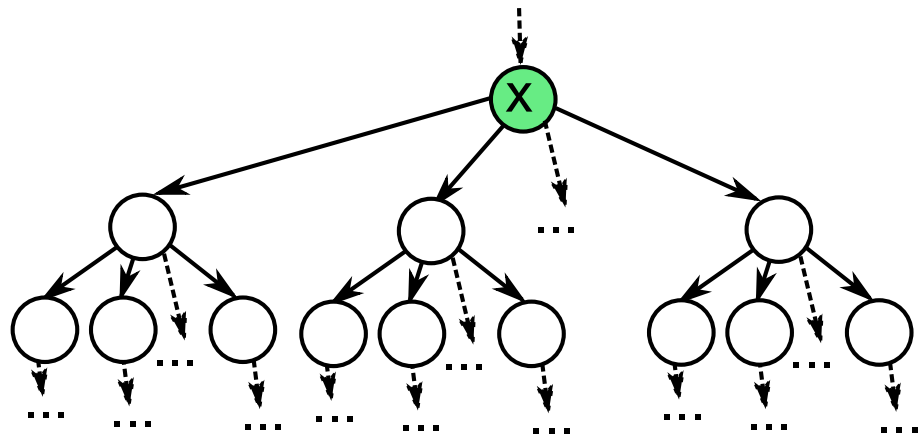
Given a root

We would like to find a sub-optimal structure in this problem. That is, a way to divide the problem into smaller subproblems. In problems where the graph is a tree, it is usually a good idea to start by [rooting](#) them, any arbitrary root is going to be fine, let's go with node 0 as a root.

The sub-tree of k nodes that we pick should have a root of its own. How about we use that as a starting point? There are $O(n)$ options for the root of the sub-tree. For each of them, count the number of ways to make a sub-tree that is rooted there.

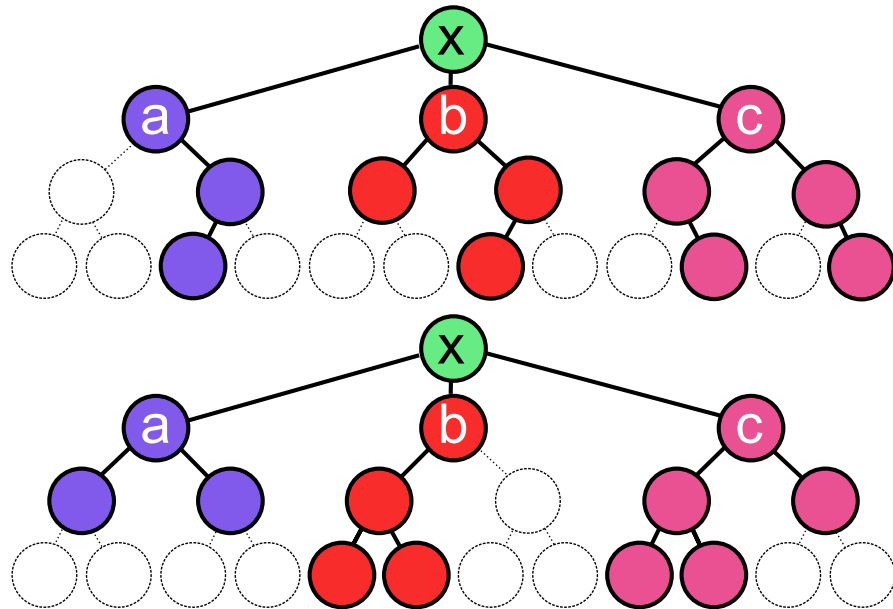
Sub-problems

We want to count the number of sub-trees of k nodes rooted at node x . Let's call the function that solves this $f(x, k)$.



It is a good thing trees are a recursive data structure. The node x will have some children and each of those children can be seen as a subtree. Of the k nodes that should be part of the sub-tree, one will be x , the remaining $k - 1$ nodes have to be distributed between these children.

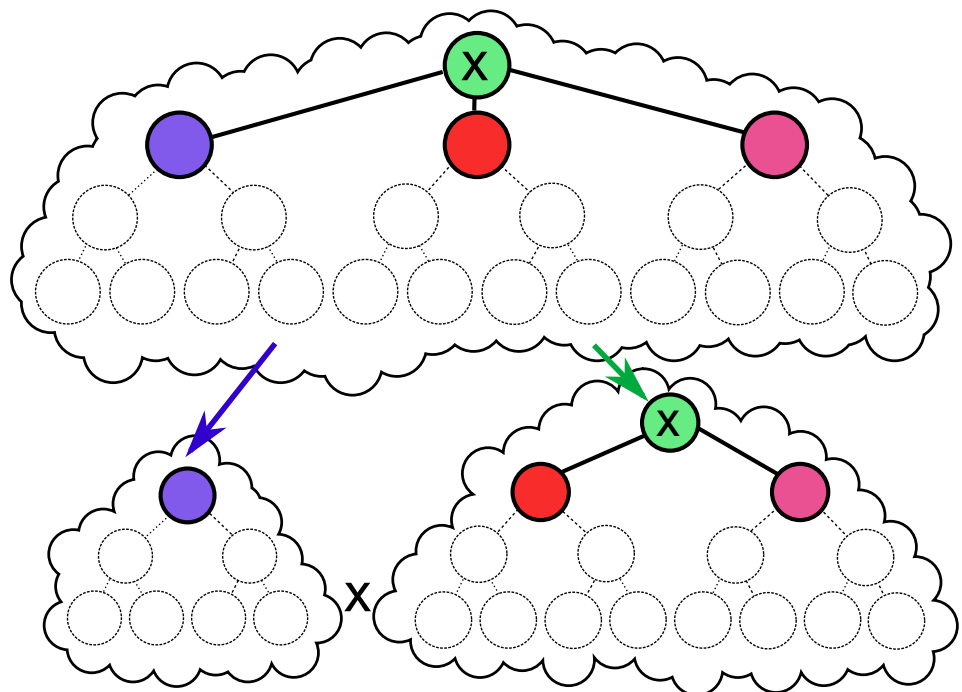
Consider a node x with 3 children: a , b and c . Assume we picked k_a , k_b and k_c as the number of nodes from a , b and c that will be used in the picked sub-tree. $k_a + k_b + k_c + 1 = k$. Any subtree rooted at a that has k_a elements will be a valid part of the result, same with b and k_b and c and k_c . We can find the number of ways to pick the nodes in each subtree as $f(a, k_a)$, $f(b, k_b)$ and $f(c, k_c)$, respectively. These choices are independent (the nodes we pick from subtree a won't alter the available options in b) so we can find the total number of ways to pick subtrees given k_a , k_b and k_c as: $f(a, k_a) \cdot f(b, k_b) \cdot f(c, k_c)$.



While this shows that we can divide the problem in smaller sub-problems, we have a new issue: How to iterate through all the ways to choose the number of elements that will go to each child subtree? We could nest a dynamic programming there, but an arguably simpler approach is to divide the sub-problems in a different way.

The left-most child

The trick here is not to distribute $k - 1$ between all the children, but only between the left-most child and the rest. Like this:



This time we solve a function $f(x, e, k)$ that returns the number of ways to have a subtree of k nodes using x as root and *without using the first e children of x* .

- If e is equal to the number of children of x , it means that we can only use the root, x . If k is equal to 1, there is exactly one way to make a subtree that has only one element (take x as the sole element of the subtree).
- Else we have at least one child. The left-most child will have index e in the list of children of x . Now we know that one of the k nodes will be the root, x . We can distribute some of the nodes in the sub-tree rooted at child e . We can try all the possibilities for the number of nodes that will belong to the subtree i . Let's say y is child # e of x : There are $f(y, 0, i)$ ways to distribute i nodes with this subtree (Because we can use all its children, so $e = 0$). There are $f(x, e + 1, k - i)$ ways to distribute the remaining nodes with the rest of the subtree rooted at x . For each i we can find the result as: $f(y, 0, i) \cdot f(x, e + 1, k - i)$
- There are some details to consider. $k - i$ shouldn't be 0, because we need the root x to be one of the picked nodes. Also, we are able to choose not to put any node in the child y . This means that the provided k to $f(x, e, k)$ can be 0. In this case, there is exactly one way to pick zero nodes, so the result is 1.

A complexity analysis: There are $O(n)$ (x, e) pairs, because there are at most $2n - 1$ such pairs in total. Consider that each (x, e) represents either a child node or the root when e is equal to the degree, so most nodes will appear in two (x, e) pairs, except for the tree's root, which will appear only once. So there are $O(nk)$ (x, e, k) tuples. Each call to $f(x, e, k)$ might need an $O(k)$ loop to pick i . We are using dynamic programming so that each (x, e, k) tuple is evaluated at most once. So the algorithmic complexity is $O(nk^2)$. This is even when we consider the $O(n)$ loop to pick the root of the subtree, regardless of how it calls $f()$, $f()$ will be evaluated at most $O(nk)$ times.

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.

```

long rootedWays(int x, int k, int e)
{
    // number of ways to have a tree of k nodes that is rooted at node x.
    // if we already processed e of its edges.
    long res = mem[x][k][e];
    if (res == -1) {
        res = 0;
        if (k == 0) {
            res = 1;
        } else if (e == degree[x]) {
            //processed all edges.
            res = (k == 1)? 1 : 0;
        } else {
            //how many must go to child #e?
            for (int i = 0; i < k; i++) {
                long p = rootedWays( g[x][e], i, 0 );
                long q = rootedWays( x, k - i, e + 1);
                res += (p*q) % MOD;
            }
            res %= MOD;
            mem[x][k][e] = (int)res;
        }
    }
    return res;
}

int ways(vector<int> A, vector<int> B, int k)
{
    this->n = A.size() + 1;
    fill(degree, degree+n, 0);
    dfsMakeTree(0, -1, A, B);

    memset(mem, -1, sizeof(mem));
}

```

Alternative solutions and additional comments.

<Place your comments here>

Next problem: [PowerOfThree](#)

Author



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](#))

in the problem:FoxConnection2

in the function:rootedWays.


for (int i = 0; i < k; i++)

why when i == k we choose to ignore?

is this situation already considered?

 Posted by ray007great at [Jan 17, 2014 00:17](#) | [Reply To This](#)


If we put all the k nodes in a child, then we wouldn't be able to use the root's node. The tree would be disconnected.

 Posted by vexorian at [Jan 17, 2014 08:39](#) | [Reply To This](#)


problem FoxConnection2

```
for (int i = 0; i < n; i++)
sum += rootedWays(i, k, 0);
//cout << sum << " " << i << endl;
```

When we consider all roots, are there any duplications?

 Posted by kaseo at [Jan 17, 2014 05:17](#) | [Reply To This](#)

The graph is a tree, and each rootedWays(i,k,0) counts the number of trees that contain i as a root. A tree can only have one root so we cannot count the same tree in two different calls to rootedWays(i,k,0)

 Posted by vexorian at [Jan 17, 2014 08:41](#) | [Reply To This](#)

In continuation to duplication of result for problem FoxConnection2

Lets take a tree: 1 <=> 2 <=> 3 <=> 4 <=> 5 <=> 6 <=> 7

For k = 4

tree rooted at 2 (1 <=> 2 <=> 3 <=> 4) : 1 as left child and 3 <=> 4 as right


tree rooted at 3 (1 <=> 2 <=> 3 <=> 4) : 1 <=> 2 as left child and 4 as right

are same which I guess will be repeated

Please spot some light

 Posted by jitendra_theta at [Jan 24, 2014 01:46](#) | [Reply To This](#)

In the function bool interesting(string s, string t), if s = a + b and t = b + a, so we can ignore all positions if s(p) is not equal to t(0).

 Posted by jiabailie at [Jan 21, 2014 04:40](#) | [Reply To This](#)

For FoxAndWord problem there is a much easier implementation.

When we compare 2 strings (let's call them A and B) to check if they form a pair, we don't need to generate all substrings of A. It's enough if we check that they have the same length and A+A contains B.

(How to add code to comment?)



Posted by zhiren at [Jan 22, 2014 17:35](#) | [Reply To This](#)

Could you check the code for FamilyCrest? I don't think it's working.



Posted by playlikeneverb4 at [Jan 24, 2014 13:23](#) | [Reply To This](#)

In problem FoxAndWord Div2 Level 1, where are taking into consideration that at kth step the Fox is suppose to move 3^k in one of the directions ?



Posted by akshayk0406 at [May 16, 2014 00:28](#) | [Reply To This](#)

First of all, Are you talking about div2 level 1 or div2 level 2?



Posted by vexorian at [May 16, 2014 00:38](#) | [Reply To This](#)



[Add Comment](#)

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

Copyright TopCoder, Inc. 2001-2014