15295 Spring 2019 #14 Problem Discussion
May 1, 2019

This is where we collectively describe algorithms for these problems. To see the problem statements follow this link. To see the scoreboard, go to this page and select this contest.

A. Jigsaw Puzzle

First, you calculate the dimensions of the matrix. Suppose we have N tiles. There are three cases:

1. n=m=N=1: the tile is 0 0 0 0.
   For the following 2 cases, no tiles should be 0 0 0 0.
2. n=1 (or m=1, but that's the same case): 2 tiles of 0 0 0 x (x>0), and N-2 tiles of 0 x 0 y (x!=y and x>0 and y>0).
   For the following case, no tiles should have three 0's and no tiles should have "separated" two 0's.
3. n>1 and m>1: 4 tiles of 0 0 x y, and L tiles of 0 x y z, and all other tiles of x y z w (x>0, y>0, z>0, w>0, distinct).
   Thus, we have n+m-4 = L/2 and nm = N. It is not hard to calculate n and m in this way. If n and m are not integers, then the board is impossible.

When we have fixed n and m, we only need to consider case 2 and 3 because case 1 is obvious.
For case 2, we pick one of the tiles of 0 0 0 x. Since for $\forall$ x>0, there are exactly 2 tiles containing x, we can uniquely find the other one. Then, we know that this tile should be 0 x 0 y, or the board is impossible. Find the other tile containing y, and continue until we visited an invalid tile (previously visited or reaches 0 before iterating all tiles) or reaches y=0 for the last iteration. If we successfully reaches the end, we can output the solution. Note that considering 1*N board is enough, because N*1 board can always be rotated to a 1*N board. This is not the case for a n*m board (n>1 and m>1).
For case 3, we need 2 possibilities: n*m board and m*n board (because flipping not allowed). We also fix one corner tile of 0 0 x y. Using the same process, construct the first row. Using the same process, construct the first column. Using the same process, for every row, construct this row using the first element of the row and check if two adjacent rows are valid. If the whole processes ss for any board check is valid, we can output the answer for that board. If both processes didn't work out, we will output impossible.
C++ solution (If anyone has a prettier solution or solution in another language, welcome to add the link below):
http://codeforces.com/group/KIrM1Owd8u/contest/243825/submission/53650600
"I know my solution is very UGLY and some vectors are unnecessary. "
---Yucheng

B. Battle Royale

C. Game Night

Let nA be the number of A's in the string, with nB and nC defined analogously. Suppose that

for each contiguous block (considering the string as cyclic) we knew countA[i], the number of A's in the block starting at position i. (And we knew the analogous arrays countB[] and countC[]. These are all very easy to compute in O(n) time.)

Now any solution must end up with the block of A's starting at some point i, then the block of B's starting at j = (i+nA) mod n, and the block of C's starting at k = (j+nB) mod n. We can compute the cost a solution ending with this pattern. It's

nA-countA[i] + nB-countB[j] + nC-countC[k]

It's also possible that the order is ACB instead of ABC. So we compute this option as well, and take the minimum of them over all starting points i. This is the solution. It's O(n) time.

D. Kingpin Escape

A couple of observations to start with. You have a tree. Ignoring the case when n=2 (handle this separately), there exists a node of degree at least 2. Pick any such node as the root of the tree. With this root, all the leaves are located at the bottom of the tree.

The goal is to add the minimum number of edges so as to ensure that the graph is bi-connected. That is, it is such that if any edge is removed the graph is still connected. Therefore after adding these extra edges there can be no nodes of degree 1 (leaves), because these nodes can be lopped off from the graph by removing one edge. So if there are L leaves, then we have to add at least ceiling(L/2) additional edges just to eliminate the leaves. And in fact we can do it with that many additional edges, ensuring that our solution adds as few edges as possible. (I.e. it's optimal.) (Note that the location of the hideout is irrelevant.)

We're going to force bi-connectivity by making sure that every edge of the tree is part of a cycle. A very elegant way to do this is via a DFS. The DFS at a node u is going to connect up all the leaves in the subtree rooted at u except possibly two of them. These are reserved to ensure that higher edges in the tree are on cycles. The base case is if vertex u is a leaf, in which case the DFS returns the list [u]. Here are the rules for combining two lists of length 1 or 2 to generate a new list of length 1 or 2. The two lists come from two different subtrees of a node u.

[a,b] [c,d] -> output edge (a,c) and return the list [b,d]
[a] [c,d] -> output edge (a,c) return the list [d]
[a] [c] -> return the list [a,c]

The rest of the cases are symmetrical. (This rule is applied sequentially on all children of u one by one to give the result for node u.) Once we get to the root of the tree if we have a list of length 1, we add an edge from it to the root, and if we have a list of length two, we connect those two nodes by an edge.

This this very elegant way to solve the problem via DFS was Corwin's.

---Danny

E. In Case of an Invasion, Please...

F. Entirely Unsorted Sequences

G. Heretical … Möbius

Modulo 4: After some observation, note that there must be some remainder of 4 such that every index with this remainder is a 0 in the string (e.g. s[3,7,11,...,199]=0 always). This corresponds to the case when we start with a “1mod4” (like mu[1]=:s[0]). So we sort of have some information about n, namely what mod 4 it is.
What about the false positives? They don’t really happen, at least in so short (10^9) a time. In order to have a “wrong remainder”, like s[2,6,...]=0, we have to have 50 consecutive non-multiples of 4 (step=4), each of them being not square-free. What is the chance of this? The probability of a non-multiple of 4 to be non-square-free is roughly 0.2 (roughly everywhere). So with odds 0.2^50 we have a wrong remainder. So we don’t.

We want to generalize this to p^2, so that we know more information about n. The “50” generalizes to 200/p^2, and the 0.2 generalizes to $\frac{(1-\frac{6}{\pi^2})p^2-1}{p^2-1}$. Thinking about a binomial distribution, the number of wrong remainder is roughly $f(p) = (\frac{(1-\frac{6}{\pi^2})p^2-1}{p^2-1})^{\frac{200}{p^2}}(p^2-1)$, with a very small StdDev. f(2)=3e-35, f(3)=1e-10, f(5)=9e-3, f(7)=1, f(11)=26, f(13)=56.
Stopping here, if the string given is real, and we superficially treat every possible remainder (those corresponding to an AP of zeros, including the right & wrong remainders) equally, then we expect about 1*1*1*2*27*57=3078 possible “remainder codes” (something looking like <<1 mod 4, 3 mod 9, … mod 169>>). How large can this number be in the reasonable world (generated from some real mobius subsequence <1e9)? With the small stddev we can *confidently* say that it’s at most 10000. If encountered a string that results in more than this # of possibilities (e.g. ‘0’*200 results in 9e8), just throw it away and say that it’s made up.

Now, we’ve got enough information about n, at a cost of 10000 potential ‘fake news’. For each remainder code, use CRT to solve for a unique n up to 2^2 * … * 13^2 = 9e8. Then it’s O(200*50) to deal with that: we compare mu(n~n+199) and s[0~199], and there is an asymptotically O(cbrt(n)/ln(n)) simple algorithm to check sq-free-ness, namely: return false if it’s a perfect square (!=1), o/w check factors until cbrt(n).

So we do a total of ~10000*200*50=1e8 computations. But because we care about the smallest possible n, could maintain a min and reject everything larger before calculating the mu’s, making the “10000” much less. The number 10000 can really be anything in [8845, 1000000] for the program to be AC.

--Fei