Problem A: Tree Flip

Author: Md Mahbubul Hasan
Special Thanks: Pritom Kundu, S. M. Shaheen Sha

First let's think how we would have solved it if the limit was not so high. We will go down from the root to leaves in BFS fashion (or DFS would also work) and if we find a 1 node, then we will make an operation at that node. This is the optimal way. Another way to think about it is, suppose we have 1 at a node x and all the other nodes are 0. To make the whole tree zero, we actually need to perform operations at all the nodes in the subtree of x (including x). Now just XOR all the corresponding subtrees of all the 1 nodes, you will know which nodes to perform operation at. So, if the root is u, the answer is the number of nodes v such that the xor of values from u to v is 1.

We can efficiently solve the problem using this last observation. First "centroid decompose" the whole tree. Then, for the query for a node x as root (Update 2), check the nodes in the path from x to the root in the centroid-decomposition tree. Say we are at y. Suppose we know the xor of values from x-to-y (this is easy to obtain using a segment tree on the euler tour of the original tree). Then we will need to find out how many nodes in the tree rooted at y have xor-path 1 from y. Depending on the xor value of the path x-to-y, you might need to actually find the xor-path 0 value in the tree rooted at y. All these are details to the implementation. But the main idea here is that we will need to maintain yet another set of segment trees for each of the components in the centroid decomposed tree to perform all these queries efficiently. Once you figure out how to do the Update 2 operation efficiently, you will be able to find out the efficient way to perform Update 1 operations.

Problem B: Binary Sort

Author: Rakibul Rushel
Special Thanks: Arghya Pal

This problem can be solved by dynamic programming. First observation is that, the answer will not be greater than 3*n, as the top 30 bits are empty in each number($^{30}C_0+^{30}C_1+^{30}C_2+^{30}C_3 >= 1000$).
dp[i][j] = What is the minimum number we can achieve on i'th position if we make at most j operations on (1…i). We initialized the dp table with INF. If we have calculated some state dp[i][j], then we will try to calculate other states from it. If a[i+1] > dp[i][j], then we don't make any operation on a[i+1] otherwise,

```
for each k=0 to 59,
    let y be the smallest number which is greater than dp[i][j] and achievable from
    a[i+1] by making at most k operations, then we update
        dp[i+1][j+k] = min(dp[i+1][j+k], y)
```

How to find out y? We leave that as an exercise to the reader but it can be done in O(number of bits) amortized complexity. So total complexity is O(n*3*n*60) = O(180*n*n)

Problem C: Network

Author: Arghya Pal
Special Thanks: Md Mahbubul Hasan

The solution is the application of component dp mentioned here. Basically we sort the towers by their radius in decreasing order. Then we place the towers one by one. We place the first tower and it's a component with span 0. Like this we will place the towers. There are three options to place a tower:

- We create its own component. In this case, the span will not increase.
- We can place it at the side of one of the components. That means, the new tower should be able to communicate with the side tower of the component. Since we are placing the towers in decreasing order of radius, the radius of that side tower does not matter. If the radius of the current tower is R, then we can place the new tower, 1 unit apart, 2 unit apart … R unit apart. If there are c components, we can place the tower in 2c ways and the span increases by [1, R].
- We can place it between two towers. If there are c towers, then there are c*(c-1) choices to combine components. And the additional span can be [2, 2R]. For each of them we can find out the number of ways to place the tower.

This way we can perform dynamic programming. However, it's too slow and we can use cumulative sum technique to make the dynamic programming faster. Intended complexity is RN^3.

Problem D: Aspect Ratio

Author: Shahriar Manzoor
Special Thanks: Md Mahbubul Hasan

One of the easiest problems of the set. Just grab a pen and paper and lay out some relevant equations!

Problem E: Overtake

Author: Mohammad Ashraful Islam
Special Thanks: Md Mahbubul Hasan, Pritom Kundu

This is an annoying problem. However if you think carefully you may find an easy implementation for it. Let A be the slower one of the bus and car, B be the faster one. There are three cases:

- A and the truck do not overlap, drive the truck till it touches A, check if at that moment if the B is behind the truck, in such case, change the lane and drive as fast as possible until it crosses A.
- B and the truck do not overlap, drive the truck till it touches B. Next check if it already crossed A, if not go with the speed of B until it crosses A. At that moment, switch the lane and drive as fast as possible until it crosses B.
- B and the truck overlaps. Let B drive and cross A, if it did not already. Since it overlapped the truck in the beginning the truck can always tailgate the B until it crossed A. At that point it will switch the lane and cross B as fast as possible.

These are the three ways the truck can overtake the other two vehicles. If none of them passes, then the answer is impossible. And it will be easy to get AC if you use Fraction class, but double with eps also passes.

Problem F: 2x2 Flip

Author: Md Mahbubul Hasan
Special Thanks: Pritom Kundu, Rafid Bin Mostofa, Tariq Bin Salam

What properties make the matrix solvable by 2x2 flips only? If the xor of values in each of the rows and each of the columns are zero, then they are solvable by 2x2 flips only. How to prove that?

First observation is that it's the same if you were allowed to flip corners of some rectangle, that is: (r1, c1), (r1, c2), (r2, c1), (r2, c2). Why? Just flip 2x2 from (r1, c1) to (r2 - 1, c2 - 1) and you will get it. So, for all the 1 cells which are not in the first row or first column let’s perform (1, 1), (1, c), (r, 1), (r, c) type operations. If we obtain the entire matrix zero then it’s solvable by 2x2 flips, otherwise not. Now, if we perform a 1x1 flip, that would flip one row, and one column. So actually, we can independently think about rows and columns. For example, for rows, we have r0 0s and r1 1s. We can perform at most K flips. We need to make all of them 0. How many different ways to do it? DP could be one solution, but the limit is too much for DP.

I hope you liked the deduction so far because the rest of the part is some boring FFT / exponential generating function stuff. Suppose A = Sum of ai/i! * x^i, similarly B is also defined. Here ai denotes the number of ordered ways of doing something (similarly bi). Note here A is an exponential generating function. Then if we multiply A and B, we “almost” get number of ordered ways to do A and B so that they can be interleaved (well… you will need to multiply by i! to get that). So for each “1” row, we can have some exponential generating function P (to be specific, it will be: p_2i = 0, p_(2i+1) = 1). Something similar for “0” row Q (to be specific, qi = 1). And since we have r1 1s, and r0 0s, we need to calculate: P^r1 * Q^r0 and the first K term will give us the number of ways to perform K flips to get all 0. Similarly for columns. And finally if we dot product these two polynomials (remember we considered both axes independently) and sum up the first K terms we will get the answer.

Fun Fact: At the last moment one of the judges wrote a too fast K^2 dp solution. Hence at the last moment the author decided to increase the limit of K. However, the problem statements have to be printed. So we omitted the limits from the problem statement to give us enough time to play with the limits and the limits were broadcasted via clarifications.

Problem G: Jump if you can

Author: Md. Muhiminul Islam Osim
Special Thanks: Md Mahbubul Hasan, Pritom Kundu, Raihat Zaman Neloy, S. M. Shaheen Sha

In these types of problems it's quite obvious that we will need to do binary search on the initial energy. If with x initial energy we can reach the destination we can definitely reach the destination with higher initial energy. That's the motivation behind binary search. Now we know our initial energy, we need to figure out if we can reach the destination with cost no more than input c. It is obviously dijkstra, however it's not clear how we can handle jumping back. We need a couple of observations. It is enough to consider only following types of jumping back:

- You are at a node u, you go to v where u-v cost is the least cost edge adjacent to u. You now jump back to u and have full energy. You can ignore the fact that you have just visited v.
- You are at a node u, you go to v where u-v is an adjacent edge to u. You now jump back to u and then jump back to v and have full energy.

It is enough to keep the state of (where are you at, what's your energy level). You don't need to keep the whole history as long as you try the above two strategies to re-energize yourself.

Fun Fact: The tricky case for this problem (the third sample) was discovered about 7 hours before the main contest. And the judges were so generous that they shared the case with the contestants but they were so evil that they did not share the explanation of it!

Problem H: Product Manager

Author: Mohammad Ashraful Islam
Special Thanks: Md Mahbubul Hasan
One of the easiest problems of the set. If you are unsure, try a few examples by hand and you will see that, whatever the order is, the required time is the same. It's because the required time is constrained by the largest value in the input list. Hence the answer is simply n!.

Problem I: Creating Triangles

Author: Mohammad Ashraful Islam
Special Thanks: Md Mahbubul Hasan, Md. Muhiminul Islam Osim, S. M. Shaheen Sha

First of all, level 0 triangle has 4 rows, level 1 has 8 rows (height just gets doubled) and so on. This way we will easily know if there is the Kth row in the n'th level triangle. Next, for n'th level triangle if the K’th row is in the bottom half, that means, it’s also the K’th row in (n-1)’th level triangle. Otherwise K’th row contains double stars of the corresponding (n-1)’th level triangle row. Hence we can recursively solve the problem. Every time we are reducing n by one. So we are solving each case in O(n) time.

Problem J: Sort it

Author: Pritom Kundu
Special Thanks: Rafid Bin Mostofa

Lemma: Let f(P, i) (0 <= i <= n) be the array constructed by replacing the i smallest integers in P by 0, and the rest by 1. Then P will be sorted if and only if all of f(P, i) is sorted after all operations. Proof is left as an exercise.

This allows us to work with binary arrays instead of permutations. there are only 2^n binary arrays of size n. For each of them, we can find out in O(q) if they are good. Let S be the set of all good binary arrays.

Then P is good if and only if f(P, i) belongs to S for all i. This allows us to do a O(2^n n) dp to count the number of good perms.