Problem A (Kings in 2D Grid)

Setter: Suman Bhadra, Alternate Writer: Sabit Anwar Zahin, Shahed Shahriar

Editorial: You must notice that R*C <= 14. This makes the problem very simple. How many different grids are really possible? Let's divide our solution into a few cases.

Case 1 (R = 1, C = N or R = N, C = 1):
We have just one row or column. The kings will never be able to cross each other. So Bob will try to move his black king towards the white king as long as it's possible to make a valid move. And then he will just follow the reverse path.

Case 2 (R = 2, C = N or R = N, C = 2):
A similar solution to the first cases. Because the kings still can never cross each other.

Case 3 (R = 3, C = 4, or R = 4, C = 3, or R = 3, C = 5 or R = 5, C = 3):
Do backtracking. As there are maximum 14 distinct cells so we can also do bitmask dp. There are not many distinct configurations for this case. So one can also backtrack and precalculate the solution for all possible cases.

Time complexity: $O(2^{R*C})$ **or** $O(R * C)$

Problem B (Mysterious LCM)

Setter: Tarango Khan, Alternate Writer: Shahed Shahriar, Suman Bhadra

Editorial:
Hint 1: X can have a maximum of 15 distinct prime numbers in its prime factorization. Because the product of the smallest 15 prime numbers is close to 10^18.

Hint 2: For any number A, if A has a prime factor p which is not a prime factor of X then A will never contribute to our solution.

Hint 3: We can extend our previous hint a bit more. If A has a prime factor p and k is the maximum power such that p^k divides A then the maximum power of p in X must be >= k. Otherwise, A will never contribute to our solution.

From the above observations, we can easily discard the unnecessary numbers from the array. Now let's get to the solution.

Lets say X = p_1^e_1 * p_2^e_2 * .. * p_n^e_n, and maximum value of n is 15 (From our hint 1).

Now let's assign an n-bit mask to each index of the array. The mask will have j'th bit on, if the power of p_j in a_i is exactly e_j. That means if you take this number, you will get p_j^e_j in lcm.

Now the problem is converted to: "Given some masks. Find the minimum number of masks that ORs to (111….111), n 1s. Which is basically 2^n-1"

This can be calculated with DP or FWHT.

FWHT Approach:
You can do FWHT / OR Convolution n times on the count of the masks. And the first time you get a positive count of 2^n-1 mask print the iteration number. This has complexity O(n^2 2^n)

DP Approach:
Let dp[mask] = minimum number of masks you need to take to get a super mask of 'mask'.

Initially, dp[mask] = 1, if the mask is in the array. But you also need to mark 1 in all the masks that are sub masks of some masks from the array. (Confusing? That is if you are marking mask 1011, then you also need to mark 1010, 1001, 0011, 0010, 0001 etc)

Why? Because if you can make a mask selecting a single number from the array then definitely you can make their sub masks in a single move as well.

This can be done with SOS DP like thing in O(n*2^n). Basically, we can just propagate from the large masks to smaller masks.

Then the rest of the DP is obvious:

```
For each mask M:
    For each sub mask X of M:
        dp[M] = min(dp[M], dp[X] + dp[M ^ X]);
```

Our final answer is = DP[2^n-1]

Time complexity: $O(3^n)$

You can also do subset convolution on min-sum subring, with almost the same complexity as the previous approach: $O(n^2 * 2^n)$

Editorial credit: Rezwan Arefin

Problem C (Swipe Your Time Away)

Setter: Mohammad Ashraful Islam, Alternate Writer: Rafsan Jani

Editorial:
Let’s define an array V[][][] where,

V[r][c][0] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its left.
V[r][c][1] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its right.
V[r][c][2] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its down.
V[r][c][3] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its top.

This array V can be calculated iterating from the top left cell to the bottom right cell and then again from bottom right cell to the top left cell.

Now we can consider each of the cell (r, c) as the cross point (or center) of a plus sign and then find the maximum number of cells having the same color as C[r][c] which is connected to the cell (r, c) using our pre-calculated array V. The maximum one from all possible center of a cross sign is our final answer.

Time complexity: $O(R * C)$

Problem D (DarkCity, CrimsonCity of FlightLand)

Setter: Shahed Shahriar, Alternate Writer: Tarango Khan, Mehdi Rahman

Editorial: Let's say we want to calculate the minimum cost for a flight between city u and city v where the distance between u to v is P and our initial total fuel is F. If P is not an integer then obviously the nearest greater integer value is the length of our optimal path. Because we can always expand the straight line path in such a way that it becomes a curve.

Now let's see how the fuels get consumed in each step following the procedure explained in the problem statement.

After the first unit distance, airplane consumes $C1 = \frac{(A+F)}{D}$ amount of fuel. So the remaining fuel is $(F - C1)$.

After the second unit distance, airplane consumes $C2 = \frac{(A+F-C1)}{D}$ amount of fuel. So the remaining fuel is $(F - C1 - C2)$.

After the third unit distance, airplane consumes $C3 = \frac{(A+F-C1-C2)}{D}$ amount of fuel. So the remaining fuel is $(F - C1 - C2 - C3)$.

This way, after all the unit distances are covered, there might still be some fuels left. But if this really happens then did we really need to carry these extra fuels? No. So in the optimal process, the remaining amount of fuel must be zero after all the unit distances are covered.

From the above calculations, we can construct an equation for F which eventually gives us the solution, $F = \frac{(X * A)}{(D - X)}$, where $X = ((1 - R^P) * D)$ and $R = (1 - 1/D)$.

So the minimum cost for a single flight $= F * C + B$.

We have to calculate this cost for the flight between all pair of cities and then we can just run a Dijkstra or Floyd Warshall algorithm which will find the minimum total cost to reach the city N from the city 1.

Time complexity: $O(N^2 * log(N))$

Problem E (Consecutive Letters)

Setter: Shafaet Ashraf, Alternate Writer: Md. Imran Bin Azad, Rafsan Jani

Editorial: The problem can be solved using the union-find data structure or STL set. We can consider a substring of the same characters as a single component. For an update operation, we can just divide or merge the components connected to that specific position and update their parents accordingly. We'll also need to maintain the size of each components to answer our queries.

The detailed editorial can be found in the following blog post:
http://en.shafaetsplanet.com/problem-solving-consecutive-letters-mist-inter-university-contest-2019/

Problem F (Palindromadness)

Setter: Sourav Sen Tonmoy, Alternate Writer: Sabit Anwar Zahin

Prerequisite: Palindromic Tree [Paper, Tutorial (EN), Tutorial (BN)]

Editorial: Firstly let's discuss an $O(N^2)$ solution. Build the palindromic tree. Then generate and store the frequencies of all unique palindromes (read nodes) [Section 2.4, Proposition 3 of the original paper]. Now for each node v, traverse to all those nodes u, who are reachable from v via suffix link chains and tree parental chains. Here, obviously, palindrome stored at node u is a substring of palindrome stored at node v. So, we can now add $freq[u] * freq[v]$ to $f(length(u))$. Of course, we will also have to add $freq[v] * freq[v]$ to $f(length(v))$.

Now, how can we decrease the complexity, and move towards the desired solution? For that let's rewrite the previously discussed $O(N^2)$ solution. Previously we iterated over v, and looked for u. Now, we will iterate over u, and look for v. Who are v nodes? The nodes, from which, node u can be reached via suffix link chains and tree parental chains.

We can reach all nodes v, from node u, by checking the nodes in the subtrees of those nodes w, who marked u as their longest suffix palindrome. We will sum up the frequencies of palindromes of each node under all the subtrees beforehand. Then for all those w nodes, we will add $freq[u] * \Sigma freq[p]$ to $f(length(u))$, where p is a node under the subtree of node w. To avoid overcount, we will mark discovery and finishing times of all nodes. We will sort all those w nodes, by discover time. Then we can easily check whether the current w node is already under the subtree of a previously checked w node. In this solution, we will also have to add $freq[u] * freq[u]$ to $f(length(u))$.

Let's talk about correctness. Let, x be a node, who marked u as its longest suffix palindrome. Let, y be another node, who marked x as its longest suffix palindrome. Also let x not be reachable from y via parental chain. So are we missing out all those nodes under node y, to mark as node v? No, it can be observed that there exists at least one node p, which can be reached via parental chain from node y, where node p marks node u as its longest suffix palindrome.

Time Complexity: As each node marks exactly one node as its longest suffix palindrome, so over all u nodes, the total number of w nodes will be N. We are sorting these w nodes, using discovery time, which costs $O(N\ log\ N)$. And that is the complexity of this solution. As you can see, the complexity is mostly dominated by the sorting algorithm used. By using a linear sorting algorithm, we can even reach a linear time solution for this problem.

Our solution runs well under 1s. We offered 10s time limit to encourage any other interesting approaches, with slightly higher complexity. Unfortunately, nobody came close to a valid solution during the onsite contest or the online replay.

Problem G (Decode The Alien Message)

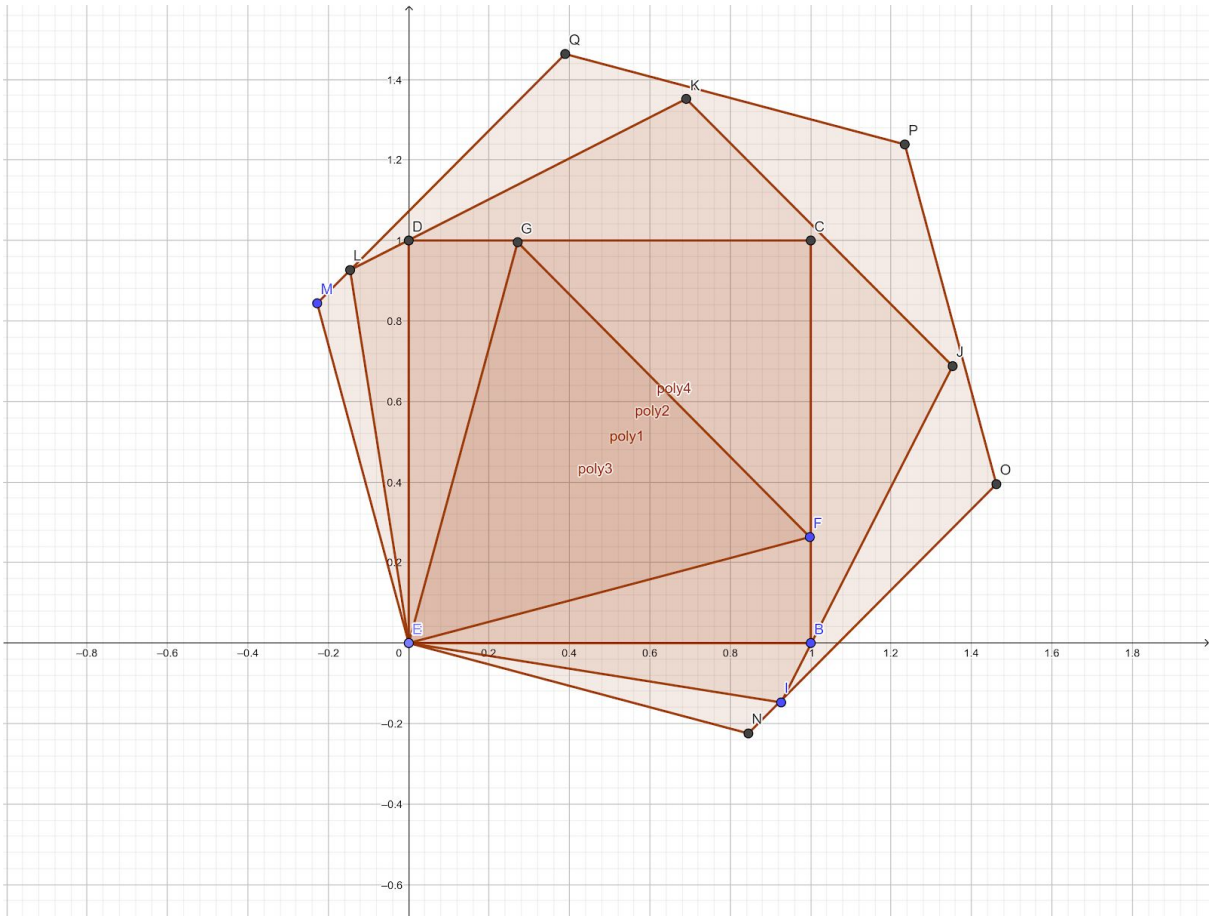
Setter: Mehdi Rahman, Alternate Writer: Muhammad Ridowan

Editorial: Very straight forward. Just make a list of the binary values and erase the trailing zeroes.

Problem H (Triangle Inside Rectangle Inside Pentagon)

Setter: Muhammad Ridowan, Alternate Writer: Mehdi Rahman, Mohammad Maksud Hossain

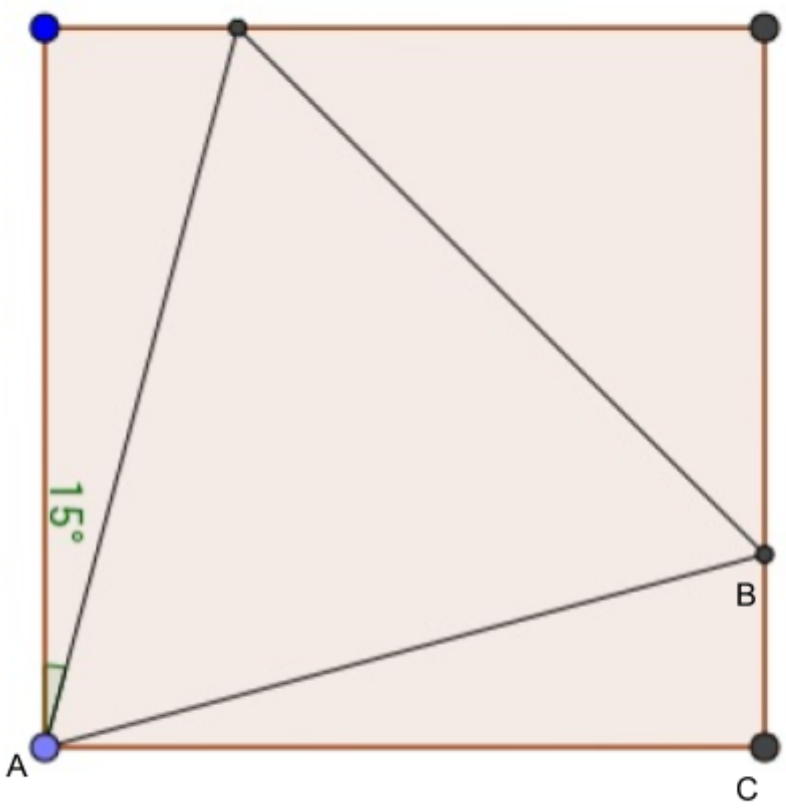
Editorial:



The optimal result is when the polygons have the same corner and placed in such a way that the internal polygon is just in the middle of that angle of the outer polygon. Prove of it is more of intuitive thinking that fitting like this will use the least space. Prove for triangle inside square and square inside pentagon can be found here

1. https://math.stackexchange.com/questions/59616/find-the-maximum-area-possible-of-equilateral-triangle-that-inside-the-given-squ
2. http://ken.duisenberg.com/potw/archive/arch98/981113sol.html

Now as finding minimum N-gon to contain a N-1-gon, let think their sides are s_{N-1} & s_N and angles a_{N-1} & a_N.



In the image

- <BAC = $(a_N-a_{N-1})/2$
- <ACB = a_N
- <ABC = Pi - $(a_N-a_{N-1})/2$ - a_N
- AB = s_{N-1} & AC = s_N

Now using sin rule of triangle, we can say

- AC/Sin<ABC = AB/Sin<ACB
- AC = AB * Sin<ABC /Sin<ACB
- $s_N = s_{N-1} * \text{Sin}(\text{Pi} - (a_N-a_{N-1})/2 - a_N) / \text{Sin}(a_N)$
- $s_N = s_{N-1} * \text{Sin}((a_N-a_{N-1})/2 + a_N) / \text{Sin}(a_N)$

As $a_N = \frac{(n-2)\pi}{n}$

So we can pre-compute that for N-gon, what is the multiple needed with the side of the triangle. Then for each query O(1) answer.

The area can be calculated by several means, a direct formula is $\frac{1}{4}ns^2\cot\left(\frac{\pi}{n}\right)$

Problem I (Fibonacci Power Sum)

Setter: Sabit Anwar Zahin, Alternate Writer: Sourav Sen Tonmoy

Editorial: The problem boils down to a linear recurrence, which can be derived using combinatorics (similar to expanding binomial coefficients). You should be able to figure it out after trying for **K=2 and C=1** in pen and paper.

For more details, check the following OEIS link, since the recurrence coefficients are basically rows of the signed fibonacci triangle.

Or if you're feeling lazy, you can simply use the Berlekamp-Massey algorithm :)

Time Complexity: $O(K^3 * log(N))$

Fun fact: Many contestants thought the recurrence would be of size **C*K** because of the constraints. **C** is actually irrelevant. The constraints were only kept relatively small with a large TL so that setter's Python solution can pass, and not to mislead the contestants.