A. Almost Pattern Matching II

Problem Setter: Sabit Zahin
Tester: Shadman Shadab
Alternate Writer: Raihat Zaman Neloy
Category: String, hashing, suffix array, binary search
Expected no. of AC: 30+

Analysis:
We need to check for each substring of length **|P|** in **S** whether it matches with **P** or not efficiently. K is very small here, so we can get the lcp of **P** and the substring in **S** using suffix array or hashing. When the next mismatch occurs increase a count and repeat. Since **K** is small, we can break this loop whenever this count exceeds **K** and the overall complexity becomes: **O(|S|-|P| * K * log(|P|))**

There is another divide and conquer solution which can pass with some optimizations.

Another solution is possible with FFT and that would work for any **K**, but in practice FFT is quite slow. So there is a high chance such solutions will time out.

B. The Social Network

Problem Setter: Md. Shafiul Islam
Tester: Raihat Zaman Neloy
Alternate writer: Tanzir Islam, Shadman Shadab
Category: DSU, PBDS, Heavy Light Tricks
Expected no. of AC: 70+
Analysis:
Maintain an ordered multiset for every network and maintain a representative for the network. Initially every user will be in the different network and that user itself is the representative of that network. Let's simulate the queries one by one.
For **1 U T**, find the representative of U using DSU and insert the post in its representative set.
For **0 U V**, find U and V's representative. If representatives are different then merge the small set into the larger set. New representative of the merged set will be the representative of the larger set.
For **2 U L R**, find the representative of U and using ordered multiset find the numbers of posts that lies between the time period.

C. ICGeSi Standings

Problem Setter: Mohammad Ashraful Islam
Tester: Tanzir Islam
Alternate writer: Sourav Sen Tonmoy
Category: Greedy, Implementation, brute force

Expected no. of AC: 150+

Analysis:
Given three information:
1) Current ranklist
2) Submission details of each team.
3) Final ranklist

We have to determine whether such ranklist is possible or not!
Well, we can say such ranklist is possible only and if only we can generate such ranklist.

The idea is simple:

- Team with rank 1 will solve all the problems.
- Starting from i=2 to n,
  i^{th} team will solve as many as possible satisfying any of the followings:
  a) Solve of i^{th} team < Solve of $(i-1)^{th}$ team.
  b) Solve of i^{th} team == Solve of $(i-1)^{th}$ team and penaly time i^{th} team $\geq$ penalty time of $(i-1)^{th}$ team.
  If there is any conflict, then the ranklist is invalid.

You can also solve this problem by thinking in reverse direction.
Overall Complexity O (**K** $\times$ **N** $\times$ **M**).
It is was an implementation problem.

D. ICPC Standings

Problem Setter: Mohammad Ashraful Islam
Tester: Tanzir Islam
Alternate writer: Sourav Sen Tonmoy
Category: parsing + dp + sliding window
Expected no. of AC: 2-4

Analysis:
You can use stringstream for parsing which would make life a lot easier. After doing the tiresome parsing, now you have a problem that can be solved with dp along with sliding window technique. The pseudo code for getWay() function is given below:

```
func getWay(curTeam, lastMask)
  dp[team][lastMask] = 0
    for( m = 0 to (2^problem)-1)
      if(m is a worse mask for curTeam than lastMask of the  previous
         team)
        dp[team][mask] += getWay[team+1][m]
  return dp[team][curMask]
```

The getWay(curTeam, lastMask) function returns the number of ways to unfreeze the ranklist from the current team to the last team when the *(curTeam-1)-th* team got ac in their frozen hour submissions for the problems whose bits were on in the *lastMask*. Complexity of this dp solution is O(team * $2^{problem}$ * $2^{problem}$) which will obviously get TLE.
The next idea is, you can use a sliding window and write an iterative dp to drop one of the mask from the complexity. This is left as a task for the reader. Final complexity of judge solution: O(team * $2^{problem}$ * problem) per test case.
[hint: sort all the submasks for every team]

E. Palindrome Again? Arghh!

Problem Setter: Tarango Khan
Tester: Imran Bin Azad
Alternate writer: Abdullah Al Munim
Category: Greedy, Backtracking, Bit mask dp
Expected no. of AC: 30+ (Maybe setter was joking :P)

Analysis:
Any string **S** will be a palindrome only if $S_i = S_{N-i+1}$ for $(1 \leq i \leq \frac{N}{2})$. So every position is part of a pair where the partner of any position **i** is (**N-i+1)**. Let's name this pair as **palinpair** where **palinpair(i, j)** is basically the string that contains the characters at the positions **i** and **j**.

For example, let's say, **S = "abababab"**. So now,
palinpair(1,8) = ab
palinpair(2,7) = ba
palinpair(3,6) = ab
palinpair(4,5) = ba

Which can be rewritten as the following,
palinpair(1,8) = ab
palinpair(2,7) = ab
palinpair(3,6) = ab
palinpair(4,5) = ab

The order of the characters doesn't matter. So we sorted them for simplicity in our solution.

So how many different values the **palinpairs** can produce? There can be a maximum of 5 distinct characters in our string. So there can be total 10 palinpair values where the characters are not equal, which are **ab, ac, ad, ae, bc, bd, be, cd, ce** and **de**.

So before going to the original solution, let's count the frequency of each **palinpairs**. Let's consider a map **freq** where **freq[pr]** = the frequency of **palnpair** value **pr**.

Now we need a few observations.

1. If both characters of a **palinpair** value is same then we can ignore them. They can never contribute in the swap operations. For example, **palinpairs** having the values **aa**, **bb**, **cc**, **dd** and **ee** will never contribute.
2. For any palinpair value **pr**, if the frequency of that value is **x (x is even)** then we need exactly $\frac{x}{2}$ swap operations to make the characters equal for all the **palinpairs** having initial values **pr**. For example, if **ab** occurs 2 times then we need just **one** swap operation to convert those **palinpair** values (**ab**) to **aa** and **bb**.
3. For any palinpair **pr**, if the frequency of that painpair is **x (x is odd)** then we can just separate one of them for now (to solve for them later) and solve for the rest **(x-1) palinpairs** using the same observation 2.

If we are able to apply our above observations correctly, then we are now only left with few painpairs having frequency exactly one. And in every operation, we have a choice to make a swap operation between the characters of any two of these **palinpairs**. Our target is to make the characters equal for each of them.

This can be solved using a simple bitmask dp approach. Our DP state is just a **mask** where the **i'th** bit is marked **one** only if the characters of the **i'th** palinpair value is not the same. And DP[mask] defines the minimum number of swaps required to make **mask = 0**. Which basically means, making the characters same in all possible palinpair values.

In every state, we can select any two **palinpairs** and make a swap between any two characters of those **palinpair** values. And we always keep storing the optimal result from all those states.

Note: For odd palindromes you also have to handle the case where the character at the middle position can also be part of a swap operation.

F. Elevators

Problem Setter: Abdullah Al Munim
Tester: Tarango Khan
Alternate writer: Imran Bin Azad
Category: Ad-hoc
Expected no. of AC: 20+

Analysis:
Notice that we can solve the problem separately for each jump. Alice should make the jump the first time she can.

Also observe that Alice can make the jump within **max(2*h[i], 2*h[i+1])** time, as the two elevators will be at the same level at least once in this time. We can get the time by some case analysis.

G. Pairs Forming GCD

Problem Setter: Sourav Sen Tonmoy
Tester: Abdullah Al Munim
Alternate writer: Md. Shafiul Islam
Category: Number Theory
Expected no. of AC: 80+

Analysis:
Firstly, given **N** and **G**, let's talk about how can we compute number of pairs **(X, Y) (1 ≤ X ≤ Y ≤ N)** where **GCD(X, Y)** is equal to **G**. Essentially, we are looking for all **(AG, BG)** pairs where **(1 ≤ AG ≤ BG ≤ N)**, and **(A, B)** are co-primes.

For every **B** in range **[1, N/G]** number of **A** values is **φ(B)**. We can precompute Euler's totient values. And then number of pairs of integers **(X, Y) (1 ≤ X ≤ Y ≤ N)**, for given **N** and **G**, where **GCD(X, Y)** is equal to **G**, will be cumulative sum of totients till **N/G**.

For a given **N**, it can be proved that, the number of such pairs will be non-increasing for increasing values of **G**. Hence, we can apply binary search to find the maximum **G**, where number of pairs is greater or equal to **P**.

H. Have You Heard of Cricket?

Problem Setter: Tanzir Islam
Tester: Md. Shafiul Islam
Alternate writer: Sabit Zahin
Category: Simple Math, Case Analysis / Binary Search
Expected no. of AC: 3-4

Analysis:
This problem had two solutions, an O(1) inequality solving approach and O(log(N)) binary search approach.
For the O(1) approach, if X bats first, assume that X scores run_x in first innings and it is optimal to assume y scores 0 runs and loses 10 wickets in the 2nd innings. Then you just need to formulate the inequality where x has a higher NRR and solve it to get the value of run_x.
If Y bats first, assume that Y scores run_y in the first innings and it is optimal to assume x scores (run_y + 7) runs to win in the 2nd innings in 0 balls. [Just assume that team Y keeps on bowling no-balls and when X and Y have equal score, X scores 7 runs in the next no ball]. Then formulate the inequality and solve for run_y.
For the 2nd approach, you can binary search on the score of the first innings and see if X can still qualify from there. It is easy to do the binary search when x is batting first. When Y is batting first, depending on the input, the binary property can be of two shape. But for one of those shapes, the answer is always either 0 or -1. So handle the 0 case, and if you can't find a solution for the 0 case, then do a binary search for the other shape. Next thing to note is,

you will face precision error if you use double or even long double since the answer can be more than 10^{13}. So you need to do integer calculation, but intermediate calculation is going to cause overflow for long long! So use 128 bit integers to escape these errors!
It can be proved for the given limits, 128 bit integers would always suffice.
To understand why there can be two shapes of the binary property, you need to analyze the original inequality. And if you can analyze the original inequality, you'd probably prefer to solve it using the O(1) approach.
N.B. For O(1) approach calculation with long long is enough. 128 bit is only needed for the binary search approach.

I. Almost Forgot to Welcome

Problem Setter: Hasnain Heickal
Tester: Imran Bin Azad
Alternate writer: Sabit Zahin
Category: Adhoc
Expected no. of AC: 0 (:P)

Analysis:
This was considered the hardest problem of the contest as any kinds of spelling mistakes, forgetting newline or wrong case of characters lead to WA.

J. Good Things Come to Those Who Wait

Problem Setter: Hasnain Heickal
Tester: Imran Bin Azad
Alternate writer: Sabit Zahin
Category: Adhoc
Expected no. of AC: 500+

Analysis: Second hardest problem in the set. As we are given all the proper divisors of N, we need to find the minimum and the maximum of those. Then answer will be min*max.