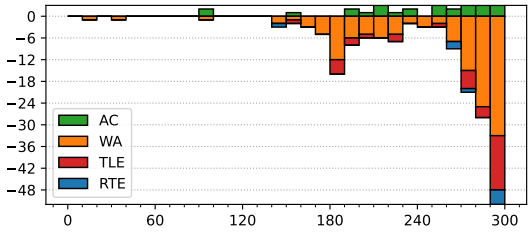
Problem F: Tilting Tiles

Problem authors:
Paul Wild and Martin Kacer

Solved by 26 teams.
First solved after 90 minutes.
Shortest team solution: 3662 bytes.
Shortest judge solution: 2240 bytes.



Once we have made at least one horizontal and at least one vertical move, the tiles will always be flush in one of the corners of the grid. At this point, we are no longer able to change the outline the tiles to any other shape, other than changing which of the four corners the tiles are anchored in. The only change possible from here is to permute the labels by cycling through these four configurations using the moves right-down-left-up repeatedly. One repetition of these four moves induces a permutation on the tiles, and the configurations reachable from here are determined by the powers of that permutation, plus some extra moves if we want to anchor in another corner.

In total, the reachable configurations are 1) some configurations reachable in 0 or 1 steps that are not anchored in a corner, and 2) some configurations that are all reachable along at most four long cycles that are each determined by a permutation.

If we fix one of these permutations, we are now left with a string theory problem: Given strings s and t, and a permutation p, is there some number n such that after n-fold application of the permutation p to the string s we get the string t?

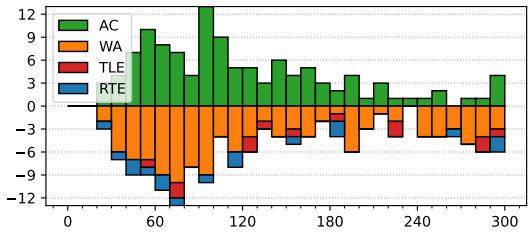
To solve this problem, we decompose the permutation into disjoint cycles, and look at each cycle independently. Using a linear-time string matching algorithm we either find that no solution for that cycle exists, or obtain a requirement in the form of a linear congruence $n \equiv a \pmod{m}$.

Finally, we need to check whether the system of linear congruences has a solution. This is a somewhat standard application of the Chinese Remainder Theorem, but the solution may potentially be huge. Rather than constructing it directly, one can show that it is enough to check each pair of congruences for consistence. As there are only $O(\sqrt{|s|})$ distinct moduli, this too can be done in linear time.

Problem G: Turning Red

Problem authors:
Jakub Onufry Wojtaszczyk and Arnav Sastry

Solved by 117 teams.
First solved after 21 minutes.
Shortest team solution: 1733 bytes.
Shortest judge solution: 1341 bytes.



This was a relatively easy problem. Let us model the three colors as red $= 0$, green $= 1$, and blue $= 2$. Then the effect of pressing a button is that all the lights controlled by the button are incremented by 1 modulo 3.

Let x_i be the (unknown) number of times we press button i. Then the requirement that a light ℓ controlled by buttons i and j must turn red becomes the equation $c_\ell + x_i + x_j = 0 \pmod{3}$, where c_ℓ denotes the initial color of this light. Note that if either x_i or x_j is known, then this equation lets us immediately calculate the value of the other one. This means that

if we consider the implied graph where two buttons i and j are connected if they control the same light, then based on the value x_i for one button we can propagate and calculate the values within the entire connected component of that button.

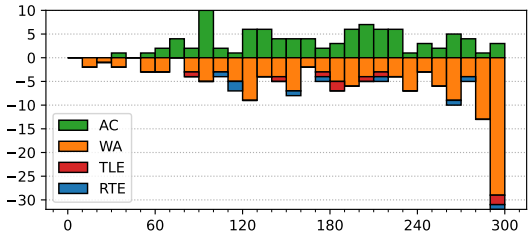
This leads to the following linear-time algorithm: for each connected component of the graph, pick an arbitrary button i, try all 3 possible values $x_i = 0, 1, 2$, and propagate the result. If an inconsistency is found (some equation is not satisfied), then this value of x_i is invalid. Otherwise, check the total number of button presses (sum of x_i's in the component). If all 3 possible choices of x_i are invalid then there is no solution, otherwise pick the one that leads to the smallest number of button presses within this component.

In the problem there could also be some lights that were controlled by a single button, leading to an equation of the form $c_\ell + x_i = 0 \pmod 3$ which immediately determines x_i. One could special-case the handling of these and propagate their values first, but it is probably less error-prone to simply include them in the inconsistency check in the above algorithm instead.

Problem H: Jet Lag

Problem authors:
Jakub Onufry Wojtaszczyk and Federico Glaudo

Solved by 96 teams.
First solved after 33 minutes.
Shortest team solution: 669 bytes.
Shortest judge solution: 246 bytes.



This problem, which has absolutely no real-world inspirations, can be solved with a greedy approach with a short implementation.

One method to solve this problem is to first ignore the restriction that you cannot sleep while rested. This allows you to sleep during some intervals $[e_i, b_{i+1}]$, where $e_0 = 0$. After sleeping during one such interval, we can be awake until $e_i + 2(b_{i+1} - e_i)$, so we greedily sleep during these intervals if they allow us to stay up later than we currently can.

After having a list of sleeping intervals from the above process, we need to modify them such that the beginning of one sleep interval is not in the k minutes after the end of the previous one. To do so, say two consecutive sleep intervals are $[s_i, t_i]$ and $[s_{i+1}, t_{i+1}]$. Then we need

$$s_{i+1} \geq t_i + (t_i - s_i) = 2t_i - s \qquad \Leftrightarrow \qquad \frac{s_{i+1} + s_i}{2} \geq t_i$$

Thus we set t_i to the mean of s_i and s_{i+1} if needed.

Problem I: Waterworld

Problem authors:
Walter Guttmann and Yujie An

Solved by 110 teams.
First solved after 16 minutes.
Shortest team solution: 171 bytes.
Shortest judge solution: 90 bytes.