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.

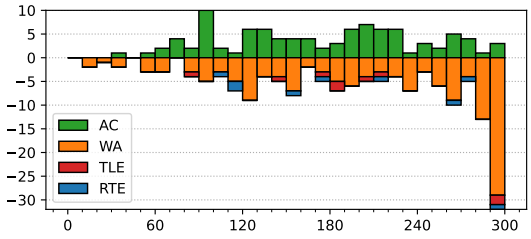
This leads to the following linear-time algorithm: for each connected component of the graph, pick an arbitrary button i, try all 3 possible values $x_i = 0, 1, 2$, and propagate the result. If an inconsistency is found (some equation is not satisfied), then this value of x_i is invalid. Otherwise, check 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

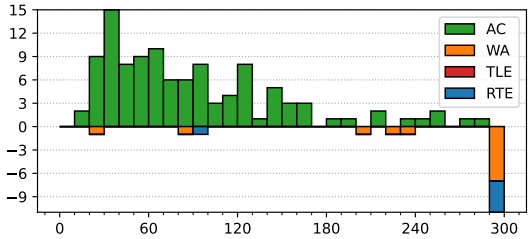
$$s_{i+1} \geq t_i + (t_i - s_i) = 2t_i - s \qquad \Leftrightarrow \qquad \frac{s_{i+1} + s_i}{2} \geq t_i$$

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.

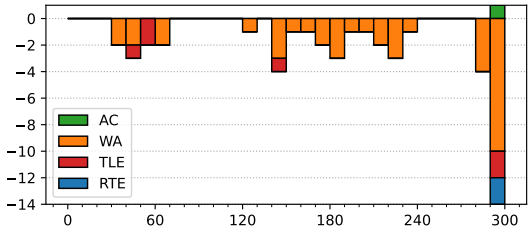


The surface of a spherical cap or a spherical segment is proportional to its height. In cartography, this is known as a cylindrical equal-area projection. Formally, the surface area of any band of height h is $2\pi Rh$ where R is the radius of the sphere. Each horizontal segment covers the same surface area during a whole rotation, and therefore also during a single step. The answer is simply the average of the numbers in the matrix.

Problem J: Bridging the Gap

Problem authors:
Walter Guttmann and Paul Wild

Solved by 1 team.
First solved after 292 minutes.
Shortest team solution: 907 bytes.
Shortest judge solution: 857 bytes.



There are a number of observations that can be made about the structure of the solution:

- Every time a group crosses backwards, the group is only one person
- Every time a group crosses forwards, the group is either:
  - the k slowest people and $c - k$ fastest people who will later cross back, or
  - some k of the fastest people, who will later cross back, or
  - all of the people left, assuming there are no more than c of them.

After every forward crossing except the last, we need to perform one back-crossing. So, we can instead pay for the back-crossing when we cross the fast walkers forward, and keep track of how many paid-for back-crossings we have accumulated.

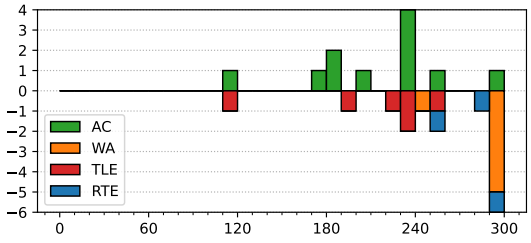
This naturally translates to a dynamic programming solution, where we calculate the cost of having the k slowest people already crossed, and l back-crossings earned. Naively, this is $O(n^3)$ — we have n^2 states, and up to n transitions out of each state. But, if you look closer, it never makes sense to accumulate more that n/c backcrossings (since that's already enough to cross everyone over), so the state space is $O(n^2/c)$. At the same time, the number of transitions out of a state is $O(c)$, so the actual cost after filtering out unreachable states and nonexistent transitions is $O(n^2)$.

While the resulting code is short, a number of judges found the implementation to be tricky to get right.

Problem K: Alea Iacta Est

Problem authors:
Martin Kacer and Arnav Sastry

Solved by 11 teams.
First solved after 113 minutes.
Shortest team solution: 1459 bytes.
Shortest judge solution: 1511 bytes.



This problem can be viewed as a graph problem and solved with either Dijkstra's algorithm or Bellman-Ford. Alternatively, it may be solved with a value iteration algorithm (which, as far as we could tell, was the approach taken by every single team solving this problem). Here we