A Opening Ceremony

Author: Michael Xiang
Here, we will treat r_1 and r_2 as integers by converting A to 1 and Z to 26.

If $r_1 = r_2$, then the answer is just the distance from c_1 to c_2, which is $|c_1 - c_2|$.

If $r_1 \neq r_2$, then Magikarp must use a staircase. Magikarp can either move left, use the stairs, then move right to his seat, or move right, use the stairs, then move left to his seat. So, the answer here would be:

$$\min\left(c_1 + |r_1 - r_2| + c_2, (n + 1 - c_1) + |r_1 - r_2| + (n + 1 - c_2)\right)$$

B Ski Buddy

Author: Johnny Liu
You can brute force this problem. Simulate what happens if Edward were to drop down at each pole.

C Olympic Haircut

Author: Nick
Notice that n is very small, thus we can simulate every subset of barbers and take the best result of all subsets. A technique to do this efficiently is with a bitmask. Iterate i over $0 \ldots (1 << n) - 1$ (all numbers with n bits), and for each $j = 0 \ldots n - 1$ if bit j is active (1) in i, include j in this subset. From here, it is easy to sum up all elements in the subset, see if the subset is valid, and minimize our result with it if it is valid. The time complexity of this approach is $O(2^n n)$, as we must iterate over all 2^n subsets and processing each one takes $O(n)$ time.

Code: here

D Revenge of the (C/K)or(e)ys

Author: Warith Rahman
Because of how curling works, this problem is asking us the probability to have the first k of a list of $2n$ stones all be American. We can calculate the probability as

$$\frac{\binom{2n-k}{n-k}}{\binom{2n}{n}}$$

In the denominator, we're choosing n of the $2n$ locations to be American (and the rest to be Swedish). In the numerator, we have already fixed that k of the American stones have to all be at the beginning, so out of the remaining $2n - k$ spots we have to choose $n - k$ to be American. Note that the numerator could also be "$2n - k$ choose n", interpreted as the number of ways to put the Swedish stones.

Warith's Code: here

E Feed the Beast

Author: Michael Xiang
We can perform a binary search on the highest number of days t such that no building goes hungry in that time. This is since if we have a valid t, then all $t' < t$ are all also valid.

To check if a given t is valid, for every building i, we will need $\lceil \frac{a_i \cdot t}{x} \rceil$ boxes of food. Suppose the total amount of food required is f_t. t is only valid if $f_t \leq b$.

The upper bound on t is 10^{12}, and each check runs in $O(n)$, so the total time complexity is $O(n \log_2 (10^{12}))$.

My solution can be found here.

F Racing Game

Author: Dylan Smith

Since the course is a permutation, it can be divided into cycles of possibly different lengths. Let the cycle lengths be $l_1, ..., l_c$. Note that the chosen checkpoint k must be reachable from s, so we can choose any checkpoint in the same cycle as s. Also notice that when swapping the two elements in the permutation we will split the cycle into two, and the choice of k allows us to split the cycle into two cycles of any size we want (such that their sum is equal to the original cycle size). Now suppose we want all racers to finish at time x. This means that we want all cycle sizes to be x, and this is only possible if all cycle sizes are multiples of x, for which the number of operations required would be $\sum_{i=1}^{c} (\frac{l_i}{x} - 1)$. Notice that as x increases, this function decreases, so we want to choose the largest x possible, and because x must be a common divisor of l_i it suffices to choose the greatest common divisor of l_i. Thus, the answer is $1 + \sum_{i=1}^{c} (\frac{l_i}{g} - 1)$ where $g = \gcd(l_1, ..., l_c)$.

G The Veneto Relay

Author: Zafir Nasim

There are various ways to tackle this problem. We can iterate through all danger factor values, since they only range from 1 to 100. For each V we consider, we can do a dfs/bfs only considering edges satisfying $w \leq V$. If the number of milestone cities in one component is K (so all milestones are in one component), then we are done. We will do up to 100 dfs, giving $O(100(N + M))$ which will pass.

If the range of danger factors was wider, we could still pass with this approach by binary searching on V, giving $O((M + N) \log W)$ where W is the max factor. Another approach is to treat the problem like finding the minimum spanning tree. We can modify Kruskal's for example, using a disjoint-set data structure which also stores for each set how many milestone cities are in it. The idea is that we add edges in order of increasing weight, and we exit when one set/component contains all milestones. This passes in $O(M \log N)$ from the sorting.

H Ultimate Figure Skating

Author: Randy Guo

Observe that a program can be represented as a graph with elements as nodes and rules as directed edges, and the goal is to find the maximum weight path that starts at a given node. Since there is no limit on the length of the path and repeat edges count for zero points, every node in any strongly connected component (SCC) of the graph can be visited, allowing the graph to be condensed into a DAG of SCC's, called the condensation graph. This can be done in $O(n)$ time using Kosaraju's or Tarjan's algorithm. Then, we can perform DP on the condensation graph in topological order to find the sequence of SCC's that maximizes the total weight, also in $O(n)$ time. Thus, the overall time complexity is $O(n)$.

I Unfreeze Tag

Author: Edward

We can solve this problem via bitmask dynamic programming. In particular, let dp[$mask$][i] be the minimum time of unfreezing some subset of the people (represented by the bitmask) assuming that there are two people at position i initially who are unfrozen. Observe that any optimal solution to this subproblem involves sending one unfrozen person to unfreeze one part of the subset and sending the other unfrozen person to unfreeze the other part of the subset, so we can iterate over all possible splits of the subset and use the best split for dp[$mask$][i].

To compute the final answer, simply iterate through all people for the first person to unfreeze and choose the candidate that leads to the smallest time.

My solution can be found here, with a time complexity of $O(n^2 \cdot 3^n)$. Michael has a faster solution which runs in $O(n \cdot 3^n)$, which can be found here.

Note for Michael's solution: $dp[i][j]$ is the minimum time to start at position i and visit the bitmask j (j does not include i). $dp2[i][j] = \min_{k \subseteq j} (\max (dp[i][k], dp[i][j \setminus k]))$, aka the best way to have 2 people start at i and visit the bitmask j (again, j does not include i).