Task Nopolynolife

Author:	Tamio-Vesa Nakajima
Attempted by:	69
Solved by:	53
Time to first solve:	15m

The task demands to consider a cyclic polygon, on which a game is defined. In this game, a move consists of specified cutting the polygon along a chord; the opponent then erases one of the resulting halves of the polygon, of his choosing. The roles then switch.

Observation 1: The losing polygon (i.e. three vertices) has an odd number of vertices.

Observation 2: If the polygon has an odd number of vertices, then no matter how we cut it, the opponent can choose from a polygon with an even number of vertices and a polygon with an odd number of vertices.

Observation 3: If the polygon has an even number of vertices, we can cut it in such a way that both of the resulting polygons have an odd number of vertices.

These imply that we win if and only if the initial polygon has an odd number of vertices.

Task Hiking

Author:	Vlad-Andrei Munteanu
Attempted by:	115
Solved by:	24
Time to first solve:	45m

In this task, you are given an undirected graph (not necessarily connected) with distances on its edges and you are asked to answer a series of queries in the form X Y P: Is there any not necessarily simple path between the nodes X and Y whose length has the given parity P?

Observation 1: We are interested in the parity of each edge only, and not in its actual value. Therefore, we can build a new graph which has 0 or 1 on its edges depending on the parity of the initial values of the edges. From now on, I will refer only to the new graph.

Observation 2: If there are two nodes, X and Y, which are connected by an edge of length 0, we can consider them as being only one node. This happens because I can go on any edge of length 0 without changing the parity of the path. Given this observation, we can compress the graph and obtain a new graph which has edges of length 1 only. And now the problem sounds like this: Given an undirected graph, you are asked to answer the given queries.

Observation 3: If the nodes X and Y are in different connected components, the answer is always "NO". This can be easily checked if we precompute the connected components of the graph.

Observation 4: If the nodes X and Y are in the same connected component,

we are interested in whether this connected component contains any odd cycles or not (i.e. whether it is bipartite or not). If there is at least one odd cycle, then we can get from any node to any other node using paths of both parities, otherwise only one parity is valid. To check this, we can use the well-known algorithm of bicolorating a graph so that any two neighbours have different colors. If we can bicolorate the connected component, then it is bipartite, else it is not bipartite.
Observation 5: Finally, for each query, we must check if the nodes are in the same connected components. If they are not, then the answer is "NO", else we must check whether the connected component is bipartite or not. This can be precomputed by using a bicolorating algorithm. If it is not bipartite, then the answer is "YES", else we must check the colors of X and Y. If they have the same color, then the only valid parity is 0, otherwise it is 1.

There are many ways to implement this solution, some of them shorter than the others, but all of them will be accepted. The final time complexity is $O(n + m + q)$.

Task Romeo

Author:	Teodor-Stelian Ionescu
Attempted by:	112
Solved by:	24
Time to first solve:	19m

In this task, you are given an array of length N containing distinct values and, considering all intervals of length K in it, you are asked to answer for each position in how many of these intervals it would be in the first half if it is sorted in a descending order. There are many solutions to this task, but we will present you a solution which would work on more general cases as well.
Notation 1: If a specific value is in the first half of an interval if it was sorted, we will say that the interval is relevant for that value. Obviously, if an interval is relevant for a value X, it is relevant for any value Y which is greater than X.
Observation 1: We want to find for each interval what is the smallest value X for which it is relevant. This can be done by using a Fenwick Tree and a parallel binary search. We will start by assuming that the required value for each intervals is 0 and we will process each bit from the most significant one to the least significant one. The maximum value in the input has 30 bits, therefore starting with the bit 30 is a good approach. When we are at the bit b, we will do the following:

1. Set the whole Fenwick Tree to 0.
2. Iterate through the intervals and set the bit b to 1.
3. Iterate through the intervals again and keep a pointer in an array which stores the values from the initial array and their the positions in sorted

order by the value. When you are at a specific interval, we will move the pointer to the right until we find a value which is greater than the current answer of our interval and for each value we are iterating through (i.e. all values which are smaller than or equal with the current answer), we will update the Fenwick Tree with 1 at the position of that value. Finally, we will get the sum of the current interval from the Fenwick Tree and if this sum is smaller or equal then $K/2$, we will keep the update on the bit b, otherwise we will not.

4. Notice that, if the initial array of intervals is sorted by the answers of the intervals, then the intervals whose updates were done are still sorted and the intervals whose updates were not done are also still sorted. Now we can merge them in a new big array containing all the intervals so that we keep it sorted for the next bits.

5. Reset the Fenwick Tree for the next bit.

At the end, we need to add 1 to each answer so that we get the smallest value for which its interval is relevant.
Observation 2: Notice that we have the array of intervals sorted by the answer and the array of values and their positions sorted by the value. Now what we have to do is to reset the Fenwick Tree and to iterate through the array containing the values and the position and keep a pointer in the array containing the intervals. When we are at a specific value, we will move the pointer to the right until we find an interval whose answer is greater than the current value and for each intervals we are iterating through (i.e. the ones who are relevant for the value), we will update the Fenwick Tree with 1 at the left end of the interval. Now the answer for the value is the sum on the interval $[position-k+1, position]$ in the Fenwick Tree.

There are many other (simpler) solutions which were used by many of you during the contest, but this one is more general and it makes use of the tricks and strategies which may be new for some of the participants and this is why we chose to present it instead of others. The final time complexity is $O(n*log^2(n))$

Task Phone

Author:	George Mărcuș
Attempted by:	122
Solved by:	9
Time to first solve:	68m

This problem seemed easy, but it had some tricky corner cases which caused a few wrong attempts even for the strongest teams.

You are given a string S of length N. You need to check if it contains X consecutive numbers in order, given that there was a prefix and/or a suffix

erased from the string so the first and the last number have at least 1 digit remaining.

Obviously if X is 1 the answer is always YES. Even if S starts with zeros, there could have been a non-zero digit prefix erased.

The evil case was when X is 2. Most of the time, the answer is YES. For example if $S = 1234$ the numbers can be 2341 and 2342. If S only contains zeros ($S = 0...0$) then the answer is NO because we can't start the second number with a zero. So we just have to check that there is at least one non-zero digit, right? Wrong! When $S = 1000$, the answer is NO. Because the first digit always belongs to the first number, we can only start the second number somewhere after the first digit. So we have to check if there is a non-zero digit starting from the second digit.

The rest is just careful implementation. Because N is very small, we don't care about efficiency. We know that $X > 2$, so there are at least 3 numbers in S. We will try every possible value for the second number by trying every index combination where $2 \leq i \leq j \leq N - 1$ (the value is the number $k = S[i..j]$). Then we keep going to the right to find the next number $k + 1$ and so on.

Finally, when we reach the end we just have to check the string $S[1..i - 1]$ is a suffix of $k - 1$, that the suffix of S after the last number is a prefix of the last number + 1 and that we find exactly X numbers.

Congratulations to the team that solved this problem very elegantly in Python, it was a very good decision!

Task Cubeon

Author:	Tamio-Vesa Nakajima
Attempted by:	74
Solved by:	8
Time to first solve:	27m

This problem is a classical example of the *divide and conquer* dynamic programming optimization. A good explanation of this can be found here: `https://cp-algorithms.com/dynamic_programming/divide-and-conquer-dp.html`

Task Power

Author:	Costin-Andrei Oncescu
Attempted by:	87
Solved by:	8
Time to first solve:	49m

In this task, you are given a linear recurrence of size K and you are asked to answer a sequence of queries X: What is the X^{th} value of the recurrence?

Notation 1 Let B_{i} be the column vector containing the values of the recurrence with indexes between i and $i + k - 1$ in this order from top to bottom.
Observation 1 Any linear recurrence can be solved using matrix exponentia-

tion. We can find a matrix A such that:

$$A * B_i = B_{i+1}$$

. Obviously, we have

$$A^{n-1} * B_1 = B_n$$

.

Observation 2 We can use fast exponentiation in order to find A^{n-1} faster. However, this is still too slow.

Observation 3 We can recompute A^{2^k} for all k between 0 and 60. Now we have to compute the product of maximum $log(n)$ matrices of size $K * K$ and a vector of size K. Note that if we multiply a vector with a matrix, the time complexity is $O(k^2)$ and the result will be a vector. Therefore, if we multiply the vector with each matrix at a time instead of multiplying the matrices between themselves and the resulting one with the vector at the end, the time complexity will be $O(Q * K^2 * log(n))$ instead of $O(Q * K^3 * log(n))$ which is small enough. The answer will be the first element in the resulting vector.

Task Shiroeseq

Author:	Tamio-Vesa Nakajima
Attempted by:	50
Solved by:	7
Time to first solve:	119m

The problem asks us to find, for a string S and Q patterns P_i, the longest subsequences of S where each P_i appears as an anagram at all positions in S. Let $M = |P_1| + ... + |P_Q|$

Observation 1: To solve all patterns of length L in $O(|S|+Q)$, use a 2-pointers approach to calculate tables for the frequencies of characters in all sub strings of length L of S. Hash these, and see which hashes coincide with the hash for the frequency tables of each P_i of length L. It is now easy to solve these patterns.

Observation 2: In order to exist X different lengths of patterns, there must be $\Theta(X^2)$ characters in these patterns (i.e. one for a pattern of length 1, 2 for a pattern of length 2, ..., X for a pattern of length X).

Observation 3: The previous observation implies that there are at most $O(\sqrt{M})$ different lengths of patterns.

Observation 4: Putting these ideas together, by solving each of the lengths of patterns separately as described in observation 1, we get an $O((|S|+Q)*\sqrt{M})$.

Task Arugaktus

Author:	Tamio-Vesa Nakajima
Attempted by:	24
Solved by:	3
Time to first solve:	67m

The task asks us to count the number of K colorings of some graph G that contains no even length cycle, for some values of K.
Observation 1: As G contains no even length cycles, it is not possible for any two cycles in G to contain a common node or edge.
Observation 2: The previous observation tells us that all the biconnected components of G are either single edges, or cycles.
Observation 3: Suppose that we know x, the number of K colorings for some connected collection of cycles and nodes not in cycles. If there is some node not in any cycle, nor in the collection, then, after adding the node to the collection, there will be $(K-1)*x$ colorings. This occurs since the new node can be colored in $K-1$ ways.
Observation 4: Let $\delta(i)$ be the number of ways in which we can color a cycle of i nodes with K colors, so that one particular distinguished node of the graph does not have some particular color. Note that $\delta(i)$ can be calculated by dynamic programming; also, we can deduce that $\delta(i) = \frac{(k-1)^2}{k}((-1)^i + (k-1)^{i-1})$. If we, as before, add a cycle of i nodes to a collection that can be colored in x ways, then the new collection will be colorable in $x * \delta(i)$ ways.
Observation 5: This implies, by induction, that if there are cycles of sizes $s_1, ..., s_m$, and the graph has n nodes, then there are $K * (K-1)^{n-s_1-...-s_m-1} * \delta(s_1) * ... * \delta(s_m)$.
So, it is sufficient to simply find the sizes of the cycles using a depth-first search, and then to compute the value described above.

Task Subfind

Author:	George Mărcuș
Attempted by:	38
Solved by:	2
Time to first solve:	160m

This was one of the easiest problems of the contest, but it was only solved by 2 teams. Maybe the long problem statement scared everyone.

In this problem you are in a 30×30 grid. You control a 2×2 square and you need to catch the enemy 2×2 square by only querying its position at most 30 times.

The solution is to first go to the center of the map so your top-left cell is at $(15, 15)$ without using your radar at all. This will make sure that the worst case scenario is as best as possible. The Manhattan distance to the enemy from this position is at most 28.

Now we will use the radar before every move. Let's assume we are at cell $(r1, c1)$ and the enemy is at cell $(r2, c2)$. We will calculate the horizontal and vertical distances as $\Delta r = |r1 - r2|$ and $\Delta c = |c1 - c2|$.

At each step we will decrement the one that is larger. For example if $\Delta r > \Delta c$ then we will move up or down in order to decrement Δr. Imagine the case when we are at $(20, 10)$ and the enemy is at $(1, 11)$ and you can see why we always need to consider the larger Δ.

So because in the worst case we will have to chase the enemy to a corner, we will catch the enemy with at most 28 radar uses and at most $28 + 28$ moves.

Task Village

Author:	Tamio-Vesa Nakajima
Attempted by:	10
Solved by:	2
Time to first solve:	190m

The task asks us to assign values to a matrix such that every value falls into some specified index, and such that the sum of each column and row is some specific value.

We will build an instance of the CIRCULATION problem that is equivalent to this task. More precisely, if our matrix has N rows and M columns, the sum of row i must be x_i, the sum of column j must be y_j, and that the value of the element at position (i, j) is between $a_{i,j}$ and $b_{i,j}$, we build a graph G with nodes $S, T, Row(1), ..., Row(N), Col(1), ..., Col(M)$. To denote an edge from x to y with capacity c and flow requirement r, write $x \to_r^c y$. Add the following edges:

- $T \to_{-\infty}^{\infty} S$
- $S \to_{x_i}^{x_i} Row(i)$ for all $i = 1, ..., N$
- $Row(i) \to_{a_{i,j}}^{b_{i,j}} Col(j)$ for all $i = 1, ..., N$, j $= 1, ..., M$.
- $Col(j) \to_{y_j}^{y_j} T$ for all $j = 1, ..., M$

If we find any circulation f that satisfies these restrictions, and let $f(Row(i) \to Col(j))$ be the value on the row i and column j of the matrix, it is easy to see that the resulting matrix satisfies all the conditions imposed on it. All that remains is to find a valid circulation ; this is a classical problem whose solution can be easily found online.

Task Petrick

Author:	Patrick-Cătălin-Alexandru Sava
Attempted by:	33
Solved by:	1
Time to first solve:	196m

The task tests more the quality of the way the contestants code. The reason why I decided not to give a chess table configurations which contain pawns was to simplify the problem (e.g to avoid giving the orientation of the table, a history of the game, etc) and to make it "codable" during the contest. I let also a reasonable time-limit in order not to ask the participants to put the accent on optimizations (both official model solutions run in 0.00s, while the time-limit is 1 second). Regarding the solution of the problem, I would make some key-observation which would enable the coder to build small components of the game in order to be plugged together in the end:
Observation 1: Object-Oriented style is welcomed in this type of problem, where you are able to have an abstract class (Piece), which is inherited by the classes corresponding to the actual pieces (King, Rook, Knight, Bishop, Queen). Each piece has two key properties (the transition array – e.g the directions in which it goes and a property which is responsible to see whether that certain piece can or cannot go more than one cell in that direction). Alternatively, it might be a third property, the color of the piece.
Observation 2: Checkmate for black means that the one who plays with the white piece was able to make a valid move and to check each position in which the king can go as a response to his move. On the other hand, for the one who plays with the black pieces, this means that he is unable to make any valid move which would eventually protect his king or will move the king itself to another unchecked position.
Observation 3: Having a look at the above-mentioned observation we can realize that the way we have to check all of these sub-problems is kind of similar, fact which makes the presence of the class Table to be more than reasonable here. Class Table will contain another class - Cell class. The latter will contain a field responsible to see whether a certain field is empty or not and also a field which will hold an instance of one of the classes representing pieces or a null pointer.

Taking into account the observations above, the participants would have reduced significantly the amount of lines of code they had to write in order to get accepted on this task.

Task Akanemat

Author:	Tamio-Vesa Nakajima
Attempted by:	1
Solved by:	0
Time to first solve:	N/A

The task asks us to find the K^{th} matrix, in lexicographic order, that follows certain properties
Observation 1: The properties mentioned are equivalent to the following: "any two squares that are at Manhattan distance at most 2 contain different values".
Observation 2: If we fix any 5 squares in the following configuration:

```
###.
##..
....
```

Then the entire matrix is fully determined. To see why, I will describe the order in which a few new squares can be deduced:

```
###6
##15
3247
```

Observation 3: If we ignore permutations of characters (eg. swapping `a`'s and `b`'s, etc.) there are only two possible matrices. These result from fixing the upper left corner of the matrix as follows:

```
abc.
de..
....
```

Or:

```
abc.
cd..
....
```

Observation 4: This implies that, overall, there are only 240 different matrices.
Observation 5: If we proceed with deducing the matrix as described before, we can notice that the matrix is periodic, with period 5, both line-wise and column-wise.
Observation 6: Thus, due to the periodicity, it is only necessary to built the matrix up to 5 lines and 5 columns.
So, to solve this problem for input N, M, K, it is sufficient to generate all possible matrices of size $min(N, 5)$ by $min(M, 5)$, to select the K^{th} one in lexicographic order, and then to extend it periodically to a matrix of size N by M.