Problem A. Fight Against Numbers

Idea: Ivan Kazmenko
Development: Ivan Kazmenko

Note that, after reversing, the most significant digit in the binary notation, which is always one, becomes the last digit. After subtraction, this one is replaced by a zero. Therefore, each power move reduces the number of ones in the binary notation by exactly 1.

The answer is the number of ones in the binary notation of n.

Problem B. Time of the Magical Number

Idea: Nikolay Dubchuk
Development: Nikolay Dubchuk

The two main approaches are:

- one can perform a brute force check and verify each time for its relevance to 239;
- one can mathematically calculate how many successful combinations there are.

Problem C. Inflation

Idea: Nikolay Dubchuk
Development: Nikolay Dubchuk

One can solve it by the formula $n \cdot 5.5 \cdot p/100$. Or with a cycle: in the first month, the loss is $n \cdot p/1000$, in the second month, the loss is $2 \cdot n \cdot p/1000$, ..., in the tenth month, the loss is $10 \cdot n \cdot p/1000$.

Problem D. Nine Out Of Ten

Idea: Anton Maidel
Development: Anton Maidel

Let's consider the subproblem of maximizing the number of successful experiments. The set of successful experiments consists of two non-overlapping subsets: the mad scientist correctly identified a successful experiment and the mad scientist incorrectly identified an experiment as unsuccessful. Since these subsets do not overlap, we can maximize their sizes independently of each other. According to the problem statement, the size of the set of correctly identified experimental results is $\frac{n}{10}$, and according to the condition, the size of the set of experiments that the scientist identified as successful is x. To maximize the number of correctly identified successful experiments, we use the fact that the experiments are independent and indistinguishable from each other, leading us to the result of $\min(\frac{n}{10}, x)$. The formula for the second subset is accordingly: $\min(\frac{9n}{10}, n - x)$. By summing these two minima, we obtain the formula for our subproblem: $\min(\frac{n}{10}, x) + \min(\frac{9n}{10}, n - x)$. The second subproblem of minimizing the number of successful experiments is equivalent to the problem of maximizing the number of unsuccessful experiments, which can be solved similarly to the first subproblem, and the final formula for the minimization subproblem looks like this: $n - \min(\frac{n}{10}, 10 - x) - \min(\frac{9n}{10}, x)$.

Problem E. Garden Bed

Idea: Nikolay Dubchuk
Development: Nikolay Dubchuk

For the minimum area, multiply the smallest board size by the sum of all the others. For the maximum area, solve the knapsack problem (selecting boards so that the dimensions are closer to a square). It can be solved by brute force.

Problem F. Largest Area

Idea: Mikhail Ivanov
Development: Mikhail Ivanov

The largest ellipse that can fit inside a square is a circle. To obtain the answer for an arbitrary rectangle, one must apply an affine transformation to convert the square into a rectangle, which will also transform the circle into an ellipse. Affine transformations preserve area ratios, which implies that the resulting ellipse will have the largest possible area.

From this reasoning, we can conclude that the ratio of the area of the largest ellipse to the area of the rectangle containing it is constant. For a square, this ratio is $\frac{\pi}{4}$; thus, the solution to the problem is to calculate the area of the rectangle (for example, using the pseudo-scalar product of the vectors of two of its adjacent sides) and multiply it by $\frac{\pi}{4}$.

Problem G. Out of Bounds Piano

Idea: Mikhail Ivanov
Development: Mikhail Ivanov

Imagine that the keyboard is two-sided infinite, and we enumerate its keys with integers. Note that each next key can be uniquely determined by its letter and the previous key. Let us construct the route on this infinite keyboard (starting with any number with proper residue mod 7) and store the reached minimum and maximum. Then, with jumps of seven, let us try to put these minimum and maximum into the segment $[0; 51]$: the answer is "`YES`" whenever it is possible. To check that, add the minimum number of sevens (possibly negative) such that the minimum is (still) non-negative, and check that the maximum is at most 51.

Problem H. Competition Results

Idea: Anastasia Grigorieva
Development: Vladislav Makarov

This problem has many solutions. Let's analyze one of them. Which place did the runner 1 take? All runners that they overtook have a higher number (because 1 is the smallest possible number). Therefore, they took the place $n - a_1$. Let's remember this fact and imagine that runner 1 did not participate in the competition (but the numbers of all other runners remain the same). The numbers a_j for $j \geqslant 2$ will not change: since 1 is the smallest possible number, runner 1 was not counted in any of the quantities a_j regardless of their performance. In the competition without runner 1, the smallest number belongs to runner 2. Therefore, runner 2 took the place $n - a_2$ in the competition without runner 1. We can continue applying this argument for all runners in the increasing order of their numbers.

How do we solve the problem now? Let's create an array of all possible places in the increasing order (initially, these are the numbers from 1 to n in the increasing order). Suppose that we consider runner i at the moment. They took the place located at the position $n - a_i$ in this array. Let's remove this place from the array. In other words, we can look at the above argument in the following way: when we pretend that runner i never participated in the competition, we allow them to "take away" their place with them "while leaving". For example, runner 2 took the place at the position $n - a_2$ in the array $(1, 2, \ldots, n - a_1 - 2, n - a_1 - 1, n - a_1 + 1, n - a_1 + 2, \ldots, n)$. In the end, we obtain an array that maps each runner's number to their place. Let's take the "converse" array that maps the runners' numbers to their places. This array is the answer to the problem.

This solution works in $O(n^2)$ time. Indeed, we have to do the following operation n times in total: remove an element from somewhere in the middle of the array. Removing an element from the middle of an array requires shifting all subsequent elements one position to the left. Hence, each iteration takes $O(n)$ time. There are also faster solutions, and there are many of them. If you are interested in this topic, we recommend searching for the keyword "inversion table" (in the context of permutations).

Problem I. Who Am I?

Idea: Nikita Gaevoy
Development: Nikita Gaevoy

Let's consider an arbitrary player. In a random test, this player will guess their card with a probability

of exactly $1/n$, where n is the number of players at the table. We need at least one player to guess the answer with a probability of 1, therefore, in the correct strategy, the answer will be guessed by exactly one player. Thus, the players' strategies must have some parameter that is different for all players and takes exactly n possible values, from which the player can recover their card. The sum of the numbers on all the cards modulo n will serve as such a parameter.

Thus, in the first run, we will choose the player whose number matches the sum of the numbers on all the cards, and in the second run, using this number as the player's number, we will recover the answer.

Problem J. Nature Reserve

Idea: Anastasia Grigorieva
Development: Vladislav Makarov

The problem can be solved in the following way. Firstly, find two points within our set that are as distant as possible. Secondly, draw a line through each of them that is perpendicular to the segment between them. It is clear that the distance between such lines is exactly equal to the distance between the chosen points.

So why, with such a choice of the strip, will all points fall either inside it or on its boundary? Let's understand this. Let our pair of points be A and B (if there are many pairs that achieve the maximum distance, we may choose any of them). Let ℓ_A be the line that passes through A and is perpendicular to AB. Similarly, let ℓ_B be the line that passes through A and is perpendicular to AB. Suppose that one of our points (let's call it C) does not lie between ℓ_A and ℓ_B. Without loss of generality, C and B lie strictly on the opposite sides of ℓ_A. Thus, the segment CB intersects the line ℓ_A. Let's call the intersection point P. Then, $|BC| > |BP| \geqslant |BA|$. The first inequality holds because P lies strictly on the segment BC. The second inequality holds because the segment BA is perpendicular to ℓ_A and thus represents the shortest distance from B to a point on ℓ_A. Therefore, points B and C are farther apart than A and B. Contradiction.

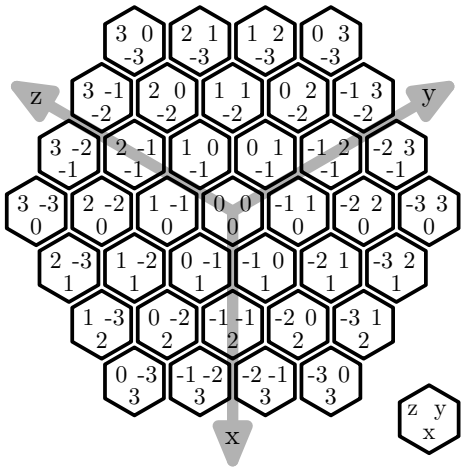
On the other hand, it is impossible to obtain a better answer. Indeed, suppose that we have two points C and D from our set. Moreover, suppose that two parallel lines ℓ_C and ℓ_D pass through C and D, respectively. Then, the distance between ℓ_C and ℓ_D does not exceed $|CD|$, since the segment CD connects the parallel lines in some way and, therefore, is not longer than the distance between ℓ_C and ℓ_D along the perpendicular. And CD is some segment between the two given points. Therefore, it is not longer than the segment AB. Here, it does not even matter that all our points must lie between ℓ_C and ℓ_D (and, in general, the set of correct answers to the problem will not change if this requirement is removed).

Problem K. Hex Operations

Idea: Ivan Kazmenko
Development: Ivan Kazmenko

First, let us learn to work with a hex with some convenience. Here is one way to do it.

Introduce homogeneous coordinates (x, y, z) along three axes with angles of 120 degrees between them. For each cell, we will have $x + y + z = 0$. If we agree that the center of the hex is at cell $(0, 0, 0)$, then the cells of the hex are (x, y, z) for which $|x|, |y|, |z| < n$.



For example, here is a pseudocode which reads the initial hex:

```
m := n - 1
create table a[m + n][m + n]
for x := -m, ..., m:
    for y := -m, ..., m:
        z := -x - y
        if -m <= z and z <= m:
            read a[m + x][m + y]
```

Now, the hex from the example is stored in the array `a` as follows:

```
0 0 4 1 8
0 3 5 1 7
2 1 6 1 8
1 7 1 9 0
8 9 9 0 0
```

Zeroes correspond to unused cells of the array.

Every other operation with a hex looks similarly in code. The code for output is almost the same. Here is another example for the "`R`" operation, a 60-degree clockwise rotation:

```
create table b[m + n][m + n]
for x := -m, ..., m:
    for y := -m, ..., m:
        z := -x - y
        if -m <= z and z <= m:
            b[m + x][m + y] := a[m - y][m - z]
```

Operations "`L`" and "`T`" are implemented similarly.

There is the second part of the problem: how to perform 250 000 operations with a hex of size 500.

Note that the available operations form a group of transformations of a hex which has only 12 elements. In other words, the hex has just 12 different states: 6 possible rotations and another 6 reflected cases.

Let us learn to calculate the operation in this group without transforming the hex itself. For example, we can store the state of the hex as two integers: the number of rotations $rotate \in \{0, 1, 2, 3, 4, 5\}$ and the reflection $flip \in \{-1, +1\}$. Then the reflection operation changes the value of *flip*, and a rotation operation adds to *rotate* (or subtracts) the value of *flip* (modulo 6).

In the end, first, apply all rotations (there will be from 0 to 5), and then perform or not perform a reflection.

Problem L. Collatz Hypothesis and Random Increases

Idea: Mikhail Ivanov
Developer: Mikhail Ivanov

There were plenty of ways to solve the problem. One series of solutions is as follows: fix two parameters $w \in \mathbb{Z}_{>0}$ and $k \in [0; 1]$. Let us try to apply the Collatz function to the current number w times. If 1 is reached within these steps, then we apply the Collatz function. If at most kw times we applied the Collatz function to an odd number, then we also apply the Collatz function (intuitively, the less we have to apply the Collatz function to an odd number, the less we multiply it by three and the more we divide it by two, thus on average we better decrease the number shown on the screen). Otherwise, apply the random function.

One reason why this solution might work poorly is that it may sometimes increase an even number. With a probability of 50% it turns odd, and the opportunity to halve it is missed. It's better to first halve the number, and only then to apply the random function. To address this issue, let us combine that (w, k)-strategy with $(1, k), \ldots, (w - 1, k)$-strategies: that is, if at least for one positive integer $v \le w$ in the nearest v iterations of the Collatz function we increase the number at most kw times, then we had better apply Collatz. This updated strategy is already accepted for $w = 11$, $k = 0.37$. To get a pair of well-performing parameters, one might brute-force through random pairs of parameters and check the performance of each one on many random inputs.

Another strategy is to try to construct an (almost) optimal solution. Let us choose a bound N and create a float array dp[1..N] denoting the mathematical expectation of the score of the optimal strategy starting with each number. Initialize it with the score reached by only applying the Collatz function. Then let us traverse the array s times, each time updating each value $dp[i]$ with the minimum of the score achieved by applying Collatz (and continuing optimally), or with the average over $a[3i + 1..6i]$, depending on what's smaller. To calculate the average over $a[3i + 1..6i]$, we can make use of the running sum (a.k.a. window sum) algorithm. This algorithm is accepted for $N = 10^7$, $s = 19$ (and certainly for many other pairs).

Problem M. Sums of Two

Idea: Ivan Kazmenko
Development: Ivan Kazmenko

This is a problem about constant optimizations.

To start, implement what the problem is asking. Maintain an array of boolean values v where v_k denotes whether an element k belongs to the set V. To count the number of pairs with sum k, consider all pairs of indices (i, j) where $i \le j$ and $i + j = k$. We will have to perform on the order of 10^{11} operations.

To make the operations more convenient, maintain another array w where the elements will be stored in reverse order: $w_k = v_{m-k}$ where $m = 999\,982$. Now the check $v_i \& v_{k-i}$ can be replaced by $v_i \& w_{m-k+i}$. The profitable part is that we can now store v_i and w_i in bits of 64-bit integers, and perform the & operation with 64 bits simultaneously, which is 64 times faster.

To speed the solution up a few times more, we can vectorize operations, but now with 64-bit numbers, with SSE or AVX. A careful implementation, or an even more careful use of `bitset` from the library, was enough to make the optimizing compiler do it by itself (in C++, for example, one should turn on the respective `pragma` instructions for that).

If this is not enough, code can be optimized further. For example, perform loop unrolling of the innermost loop by hand. Or store 64 `bitset` instances for shifts of $0, 1, \ldots, 63$ bits. The fastest jury solutions in several languages work 4–5 times faster than required in the problem.

Problem N. Wanted: Second Sock

Idea: Ivan Bochkov
Development: Ivan Bochkov

Let $f(p, m)$ be the answer to the problem. The problem can be viewed as follows: a random permutation

of the socks is fixed, and the expectation of the position of the first sock that appears twice is taken. Note that

1. $f(p,m) = \frac{f(p,m-1)\cdot(2p+m+1)}{(2p+m)}$. Indeed, let's select one unpaired sock and examine the order of the remaining socks. With the rest fixed, we can insert 1 sock in each gap with one probability. That is, moving to the expectation, the "length" of the gaps is multiplied by $\frac{(p+2m+1)}{(p+2m)}$.
2. $f(p,0) = f(p-1,1)$ if $p \geq 2$. Indeed, let us denote the last sock in the permutation as 1. Then we can just remove this sock and add the remaining sock of type 1 to the "unpaireds".

From the two statements, it follows that $f(p,m) = \frac{(2p+m+1)4^p}{(2p+1)C_{2p}^p}$, which, after calculating the factorials and their inverses, can be calculated in $O(1)$.