Problem A. An Experiment in Optics Lab

The Δy is represented as: $\tan\alpha \cdot \text{width}$. If it were sine, the problem would be easier. Instead we have:

$$\frac{\sin}{\sqrt{1-\sin^2}}$$

Taylor series:

$$\sum_{c=0}^{\infty} [\sin^c(\theta_i) \cdot f_c \cdot l_i]$$

Instead of ∞, take some not very big constant, and now it is maintainable with several segment trees.

Problem B. Bring Order, Stop Havoc!

Due to the condition about `D`-tasks and `T`-tasks, each programmer should finish firstly all their `D`-tasks and only then finish all start doing the `T`-tasks. Without loss of generality, suppose Petya finishes first (not later than Vasya). Then Petya needs to choose the next `T`-task he completes in a way that it finishes strictly after all `D`-tasks of Vasya get completed. Let us just iterate over all Vasya's `T`-tasks to choose the first one and check that it fits. After that, all other tasks' order does not matter: all `D`-tasks can be completed in any order (the moment of finishing the last of them does not depend on in), as well as all Vasya's `T`-tasks, starting with the first one, and all Petya's `T`-tasks, starting with the second one. So the number of schedules is a product of four factorials, and we need to multiply this product by the number of valid choices of Petya's first `T`-task to get the answer to the problem.

Problem C. Choosing Best Friend

Basic, but slow, solution is to find the maximum matching in bipartite graph between V and 2^V, where each vertex is connected with all subsets of its neighborhood, and we need to find the matching covering the first part minimizing the total weight of vertices of the second part covered by this matching.

Instead of 2^V, let us just for each vertex of the left part only keep n neighbors of minimum weight, it is enough.

Problem D. Definitely Not Prime

There are actually several constructions starting with some bigger length: (4|6|8|0)*. (6|9|0)*, digit 0* digit, (6|9)*49. For smaller length — bruteforce.

Problem E. Expose The Werewolf

If $A = B$, impossible. Assume that we don't ask questions; then everything we know is several pairs of numbers (a_i, b_i): we know that in one of the parts there are these a_i vertices, and in the other one there are b_i vertices.

No questions $\to$ knapsack. First statement: if the number of possible partitions > 4, then it is impossible to know (easy). If ≤ 4, then two questions are enough. So knapsack with counting the number of ways.

Issues:

1. Pairs are hard. Instead of pairs, we have knapsack with sizes $|a_i - b_i|$ and need to collect $A - B$ weight.
2. Basic solution is n^2. There is a FFT knapsack in $O(n \log^2 n)$. Also there is a trick in $O(n\sqrt{n})$: store the counting array, if there is an entry ≥ 3 (we have ≥ 3 elements of weight w), instead make it $w + 2w$. We did not lose reachability since we still have an object of weight w. This method loses the thing about «number of ways», so instead we store in this array number at least 5, so if it reached 7, we replace it with 5 plus 2 in the $2w$ cell; it works in $6 \cdot n \cdot \sqrt{n}$.

Problem F. Fix The Bad Ping

In two binary searches of full height we can find the x's of two cells, in two more of full width — the y's of two cells. The last query is to distinguish between $(x1, y2) + (x2, y1)$ and $(x1, y1) + (x2, y2)$.

Problem G. Good Coach

If $n > k$, then there are not enough digits. Even $n = k$ is impossible because 0 is not a valid bus number (it should be positive). Therefore, $n < k$. Let us look at the optimal answer. If there are two numbers of lengths that differ by at least two, we can move a digit from the longer number to the shorter one, and the maximum does not increase; so we may assume that all numbers have almost equal lengths. Moreover, we know exactly that the longest number (which will be the answer) will have length $l = \lceil \frac{k}{n} \rceil$, and there will be exactly $c = k - n(l-1)$ numbers of length l. To minimize the maximum of them, firstly we need to assign the first digits to them: $1, 2, \ldots, c$. Then the number starting with c will be our answer, and, to minimize it, we need to finish it as $c, 0, c+1, c+2, \ldots, c+l-2$.

Problem H. Hero of Sushi

Firstly, instead of a suffix, let's work with the whole array. If $a_i < a_{i+1}$, we could transfer one coin into the future: we could make one update less on i and one update more on $i+1$.

So if we take our increasing array and just greedily do the maximum that we can, transferring some residue to the next day, it will be the correct answer, but not exactly increasing. The number of rerolls decreases by at most 1 each day: if we reached the maximum M, we will always have $\geq M - 1$, which is enough to get the optimal answer.

Let's learn how to solve this problem from right to left. How will the array look after this "rebalancing to the right"operation? Some Young diagram centered in the right-bottom corner. Stack of pairs $\langle x, \text{number of occurrences of } x \rangle$. Its top is at the left bound; it is increasing from shallow to depth.

When a new element comes to the beginning of the stack, we need a rebalancing. It is quite easy to recalculate the stack. But we cannot maintain the answer directly.

Let us store two more numbers:

- **answer** — the answer on the suffix if there is literally nothing more except this suffix, and the game starts at the beginning of this suffix;
- **surcharge** — how much money we need to add to increase the answer by 1 (never able to increase by 2: if by ≥ 2, then actually the stack wasn't rebalanced).

So we're adding a new element to the stack; during the rebalancing we have some residue — let's compare it with the surcharge. Possibly it increases the answer by 1, and now we need to recalculate the surcharge.

Problem I. ICJ

The graph in the statement is too big, instead of making all bad edges directly, let us create a new vertex for each airport which says "we will use a bus transfer in this airport" and connect it with bidirectional infinite cost edges with all vertices of this airport; this way the number of edges is linear, and Dijkstra will work fast enough.

Problem J. Just A Friendly Trick

Firstly, only the whole array. Let dp[i][j] denote: the first i elements, j cycles. Either a new cycle, or an existing cycle (the number of ways is equal to the number of previously seen elements of the same color).

The generating function satisfies:

$$p_{i+1}(x) = p_i(x)\big(c_i + x\big).$$

To calculate only one coefficient, in the end we multiply by:

$$\sum x^{n-\text{favorite number}}.$$

This is $O(n \log^2 n)$.

What about all prefixes? To run a divide-and-conquer, we first calculate the answer for the left half. Then we notice that we don't need all the coefficients, but only the last $N/2$: we can divide by $x^{N/2}$ and truncate all negative powers. Then in total it will also be $O(n \log^2 n)$.

Problem K. Keyword and Numeral

Read a string, output the block up to and including = as is, then extract two summands. According to the problem statement, there are only two possibilities: if the first character is a letter (then we also output it as is) or if the first character is a digit, in which case the summand is a number (meaning there is no need to check for the presence of letters inside). For the number, set the variable *cnt* to the number of digtis in it, and then print the numbed digit by digit, decreasing *cnt* by 1 after each digit. Every time *cnt* becomes a non-zero multiple of 3, print an apostrophe.

Problem L. Long and Random

Actually, for each k you need to delete around $\frac{k}{9} \pm c\sqrt{k}$ numbers. After that it is DP. In total $O(n\sqrt{n})$.

Problem M. Math, Nero and Seneca

With a trial division up to $B = 3\,999$ we can represent $n = p_1 p_2 \dots p_k q$ where $p_1 \le p_2 \le \dots \le p_k$ are primes less than or equal to B, and q is a positive integer all of whose prime divisors are greater than B. If $q > 1$, there is no representation at all because none of the Roman numerals is divisible by any prime divisor of q, thus the answer is 0. If $q = 1$ and $k \ge 2$ and $p_1 p_2 \le B$, then there are two different representations $n = p_1 p_2 \dots p_k = (p_1 p_2) \dots p_k$, thus the answer is 0 again. Otherwise the answer is 1: the only representation is $n = p_1 p_2 \dots p_k$ because in any other product there would be at least one composite number, but any feasible composite number is at least $p_1 p_2 > B$.