Problem A. Accumulator Apex

Problem author and developer: Vitaly Aksenov

Let us break each list of integers into sublists greedily: 1) we go over elements maintaining their sum and the minimum sum seen so far; 2) when the sum becomes positive we split the list and store the sum and the minimum observed sum; and 3) we continue with the list from the step 1 until it becomes empty. Then, we populate the priority queue with the first lists of each list sorted by the minimum observed sum. Now, we should take the sublist with the biggest minimum sum — it reduces the chances to make x non-negative while increasing it, at the end. Thus, in each step, we take the sublist with the biggest minimum and apply it to x. Then, we withdraw this sublist from the corresponding list i and put the next sublist from the list i into the priority queue.

Problem B. Blueprint for Seating

Problem author: Artem Vasilyev; problem developer: Niyaz Nigmatullin

Consider any group not located next to a window. If the group consists of t seats, then the inconvenience part of the group is $s(t_1) + s(t_2)$, where t_1 and t_2 are chosen optimally such that $t_1 + t_2 = t$, and $s(x) = \frac{x(x-1)}{2}$.

For a group of t seats next to a window, the inconvenience part is $s(t)$.

So the inconvenience is the sum of some $s(r_1) + s(r_2) + \ldots + s(r_{2k})$. Each r_i is either one of the two sizes of groups next to a window, or one of the $2 \cdot (k-1)$ subgroups created by division of middle groups.

Consider we've chosen r, $a = \max r_i$ and $b = \min r_i$ such that $a - b \geq 2$, then $s(a-1)+s(b+1) < s(a)+s(b)$, because

$$\begin{aligned} 2 \cdot (s(a-1) + s(b+1)) &= (a-1) \cdot (a-2) + (b+1) \cdot b \\ &= (a-1) \cdot a - 2 \cdot (a-1) + (b-1) \cdot b + 2b \\ &= 2 \cdot (s(a) + s(b)) + 2 \cdot (b - a + 1) \\ &< 2 \cdot (s(a) + s(b)) \end{aligned}$$

That's because $(b - a + 1)$ is negative.

So the only way to divide is almost equally. The multiset of r: $n \bmod (2k)$ times $\left\lceil \frac{n}{2k} \right\rceil$ and $2k-(n \bmod (2k))$ times $\left\lfloor \frac{n}{2k} \right\rfloor$.

We can iterate over four ways to choose the window groups, not considering, when the window group is of size of 0. Then we need to calculate how to arrange middle groups.

We have $(k-1)$ middle groups, and x subgroups of size $\left\lfloor \frac{n}{2k} \right\rfloor = t$ and y subgroups of size $t+1$.

Each group can be of size either $2t$, or $2t+1$, or $2t+2$. If we iterate with number of groups of size $2t+1$ say g, then $\frac{x-g}{2} = f$ groups will be of size $2t$ and $\frac{y-g}{2} = h$ groups of size $2t+2$. The number of arrangements with the fixed g is the product of some binomial coefficients, for example $\binom{f+g+h}{g} \cdot \binom{f+h}{f}$. We should only consider cases, when f and h are non-negative integers. And we should handle the case when $2t$ equals to zero, it means f needs to be zero as well.

The binomial coefficient can either be recalculated when iterating g, by several multiplications and divisions. Another way to do it is to pre-calculate all factorials and their inverse modulo the given prime number, and use them to calculate binomial coefficient in $\mathcal{O}(1)$. For a linear algorithm to pre-calculate all inverse values modulo prime p you can use the following recurrent formula:

$$x^{-1} \equiv -(p \bmod x)^{-1} \cdot \left\lfloor \frac{p}{x} \right\rfloor \pmod{p}$$

note that $(p \bmod x)^{-1}$ can be calculated before x^{-1}. The formula comes from relation $p \bmod x = p - \left\lfloor \frac{p}{x} \right\rfloor \cdot x$.

The total time complexity for a single test case could be $\mathcal{O}(k)$ or $\mathcal{O}(k \log k)$, depending on how you handle the binomial coefficients.

Problem C. Cactus Transformation

Problem author and developer: Levon Muradyan

Let's initially consider some edge cases:

- If both cactuses are the same, then we don't need to do anything;
- If cactuses consist of only cycles of length 3 and don't contain any bridges, then there is no transformation from the first cactus into the second one because if we remove any edge from the first cactus, we can not add the new one (we need to add the same deleted edge).

For all the other cases, the solution exists. To solve the problem, we will transform both cactuses into the same convenient cactus, let's call it as *star* cactus. If the cactus contains n vertices and c cycles, then the star cactus will have the following cycles: (1, 2, 3), (1, 4, 5), ..., (1, $2x$, $2x+1$), all the cycles contain the vertex 1. The bridges of the star cactus will be (1, $2x+2$), (1, $2x+3$), ..., (1, n), all the bridges will be incident to the vertex 1.

Operations for transforming the first cactus into the second one will be the concatenation of operations for transforming the first cactus into the star cactus and operations for transforming the second cactus into the star cactus but already in **reverse order**.

So, let's understand how we can transform any cactus into a star cactus. Before understanding the actions needed for an algorithm, let's do some denotations:

- Let's denote by deg_v — the degree of the vertex v ($1 \le v \le n$);
- Let's denote by $dist_v$ — the minimum distance from the vertex 1 to the vertex v ($1 \le v \le n$).

First action: let's take any cycle c_1, c_2, ..., c_k of length k (where $k > 3$), and do the following operation on it:

- Remove an edge (c_1, c_k) from the cactus and add an edge (c_1, c_3).

After applying this operation, the number of cycles of length greater than 3 will be decreased by 1. We will perform the following action while the cactus contains any cycle of length greater than 3.

Second action: let's take any bridge (v, u) from the cactus, such that $dist_v < dist_u$, $v \neq 1$, and $u \neq 1$. Remove an edge (v, u) from the cactus and add an edge (1, u). We will perform this action while the cactus contains any bridge that isn't incident to the vertex 1.

Third action: let's take any edge (1, v) that is incident to the vertex 1 and $deg_v > 2$. There are two cases:

Case 1: Edge (1, v) is bridge.

We can note that deg_v is odd because the number of bridges incident to any vertex is at most 1. Since $deg_v > 2$, we know that there is at least one cycle that contains the vertex v. Let's take any cycle that contains the vertex v and assume that the vertices of the cycle are v, u, and w. Let's do the following operations:

- Remove an edge (u, w) and add an edge (1, u);
- Remove an edge (v, w) and add an edge (1, w).

Let's note that after these operations, the number of cycles that contain the vertex 1 will be increased by 1, while the number of bridges incident to the vertex 1 will not be changed.

Case 2: Edge (1, v) isn't bridge.

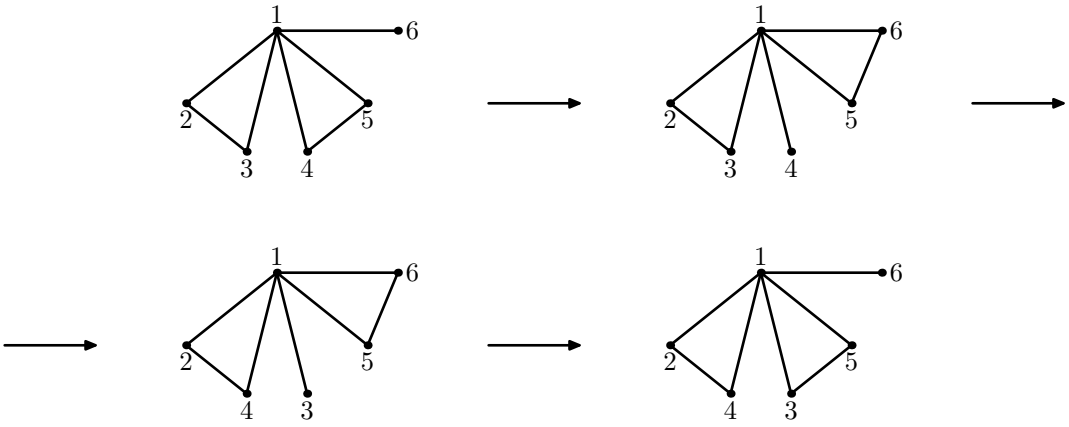
We can note that deg_v is even because all the bridges are incident to the vertex 1. We know that an edge (1, v) isn't a bridge, so let's assume the vertices of the cycle containing an edge (1, v) are 1, v and a. Since $deg_v > 2$, we know that there are at least two cycles that contain the vertex v. So, let's take any cycle that contains the vertex v and is different from the cycle (1, v, a). Let's assume that the vertices of the cycle are v, u, and w. Let's also understand that there is at least one bridge in the cactus because all the cases that don't contain a bridge were considered initially as edge cases. So, let's take some bridge (1, b) incident to the vertex 1. Let's do the following operations:

- Remove an edge (v, a) and add an edge (a, b);
- Remove an edge (u, w) and add an edge (1, u);
- Remove an edge (v, w) and add an edge (1, w).

Let's note that after these operations, the number of cycles that contain the vertex 1 will be increased by 1, while the number of bridges incident to the vertex 1 will not be changed.

We will perform this action with two cases while the cactus contains any vertex $v \neq 1$, such that $deg_v > 2$.

After all of these three actions, we will get the structure of the star cactus, and it remains to fix the labels in the vertices. In the picture below, you can see the operations that are needed for swapping two labels (in the example below, we swapped the labels 3 and 4).



In practice, the number of operations for this solution is approximately 5000, but for solid proof, you can see that it can't be greater than 15000.

Problem D. Divisibility Test

Problem author and developer: Roman Elizarov

It can be shown, that this particular kinds of divisibility tests can be checked by looking at the remainder of b^k modulo n:

- **Kind 1** — $b^k \equiv 0 \pmod{n}$.
- **Kind 2** — $b^k \equiv 1 \pmod{n}$.
- **Kind 3** — $b^k \equiv -1 \pmod{n}$.

So, all it takes to solve this problem is to compute b^k modulo n for increasing k and stop at the first one that reaches remainder of 0, 1, or -1. Report that there is no answer if the computation loops without reaching any of the required remainders.

Problem E. Evaluate It and Back Again

Problem author and developer: Artem Vasilyev

Let's pick a positive integer X without leading or trailing zeros. Consider an expression `X+X-0`. Its value is $2X$, but the reversed expression evaluates to `0-rev(X)+rev(X)=0`. Let's call this expression `f(X)`. The result of evaluating `f(X)` is $2X$ and the result of evaluating `rev(f(X))` is 0. Thus, when both p and q are even, `f(`$\frac{p}{2}$`)+rev(f(`$\frac{q}{2}$`))` is a possible answer. For negative X, define `f(X)` to be `0-|X|-|X|`.

When p or q is odd, let's make both of them even. First, if they have different parities, add `12+` in the beginning. This would add 12 to the value of p and 21 to the value of q, so now they have the same parity. If they are both odd, append `+1`.

The remaining issue to be solved is that p and q might have trailing zeros. This can be solved by adding `+x` instead of `+1` in the last step (such `x` that makes p and q both even, but not ending with 0, always exists). The maximum length of such an expression is $4 + 2 \cdot (4 + 2 \cdot \texttt{len}(\frac{p}{2})) \le 84$.

The larger limit on the length of the answer allowed for many other approaches. For example, you can use the fact that `12-4*4+5` is a solution for $p = 1$, $q = 0$, multiply it by any number, and add it with its reverse. Another approach (yielding significantly longer expressions) is to use numbers like `xx..xx` (or palindromes in general), `xx..xy..yy`, `9*9*...*9*x` to make the input numbers (either p and q independently, or, their difference and an offset). Depending on the specific construction, this type of solution also has a good chance to pass the tests.

Problem F. Fugitive Frenzy

Problem author and developer: Mikhail Ivanov

Let us call a state of the game *initial* if, from the police officer's perspective, it is effectively same as the beginning of the game in the vertex she is currently in (that is, the police officer does not have any sort of information about the fugitive's position except that he is not sharing a vertex with her). The state before the first move of the game is indeed initial, but also such is every moment when the police officer comes into a leaf (because at that moment all the other vertices form a connected component). Our aim is to prove that, being in the initial state, the police officer should choose at random one of the leaves she is not currently in and walk into it until she gets there and comes into another initial state. As for the fugitive, after an initial state his optimal strategy is to choose at random a leaf unoccupied by the police officer, move there and wait there till the next initial state. Moreover, the probabilistic distribution on these leaves only depends on the accommodation vertex of the police officer in the initial state.

We will prove this in several steps. Firstly, we will assume that at the beginning each of the players generates an infinite random binary string. Each time they need to make a random decision, instead of accessing a random number generator they can just take another one bit (or several ones) from their string. Therefore, we may assume that after the initial string generation their strategies are deterministic. Such pair of strings, generated by the players, will be called an *elementary event*.

We may assume that the fugitive only hides in the leaves of the tree. Indeed, if he choses a non-leaf vertex v, he can as well hide in any leaf reachable from v. It will not change the set of reachable vertices no matter where the police officer moves, but might help him avoid the capture in some elementary events.

Next, we will note that the fugitive may be assumed to not change the shelter after the police officer's step towards him. To prove that, we will change his strategy as follows. Imagine that police officer is in an initial state. Let us fix an elementary event and perform a simulation during which each time the police officer moves, she moves towards the fugitive. At some moment she inevitably catches the fugitive in some leaf ℓ. Note that ℓ can be found deterministically because all randomness exploited by the fugitive is contained within an already generated random string. Then, according to the new strategy, the fugitive prematurely jumps into ℓ and waits till either he is caught or the police officer takes a step away from him. It can be shown that, if the new strategy lead to a capture at some point, the previous strategy also led to a capture no later than the new one elementary event-wise.

In a very similar fashion it can be shown that the police officer should never move along the same edge twice in a row (unless the first of these moves led him into a leaf) — that is, the optimal strategy of the police officer in an initial state consists of choosing some leaf and walking there. Also, with a similar

reasoning, it can be shown that the fugitive only needs to chosse his shelter at the beginning of the game and after the police officer getting into a leaf (and thus triggering the initial state).

Therefore, the decisions in this game are only made in initial states. Thus, we can condense the game to its initial states only and assume that the game now is as follows. The police officer appears in vertex s. Then the players take turns, starting with the fugitive. At his turn, the fugitive, who knows the current position of the police officer, chooses a leaf f, different from the police officer's position, and sits there. At the police officer's turn, she chooses a leaf p' and walks there from her current vertex p, spending $\operatorname{dist}(p, p')$ minutes. Then, if $p' \neq f$, the game continues, otherwise, it ends.

In this setting it is easy to see that one move takes at most $n-1$ minutes, and the police officer can guess the correct leaf and finish the game with probability at least $\frac{1}{n-1}$, therefore, the mathematical expectation of the duration of the game, assuming the optimal play by the police officer, is no more than $(n-1)^2$, in particular, it is finite.

Due to Nash, there is a pair of optimal strategies called the *Nash equilibrium*, in which the probabilistic distribution only depends on the position of the police officer. Therefore, we have completed the proof of the fact in question.

Let us fix the optimal strategy. Denote by $p_{u,v}$ the probability that, if the police officer is initially in u, she chooses to walk to v. Denote by $q_{u,v}$ the probability that, if the police officer is initially in u, the fugitive chooses to hide in v. Denote by x_u the mathematical expectation of the duration of the game if the police officer is initially in u. According to the previous reasoning, $p_{u,v} = q_{u,v} = 0$ unless v is a leaf different from u.

We will prove the converse: both $p_{u,v}$ and $q_{u,v}$ are positive if v is a leaf different from u. To see why, firstly we will assume that $q_{u,v} = 0$ — that is, when the police officer is in u the fugitive never choses to hide in v. Then the police officer, if $p_{u,v} > 0$, can modify her strategy so that she does not visit v, e.g. by stopping right before visiting v and choosing the next leaf right there. This circumstance would violate the Nash equilibrium. Therefore, $q_{u,v} = 0$ implies $p_{u,v} = 0$. Similarly, one can deduce that if $p_{u,v} = 0$ for a leaf $v \neq u$ but $p_{u,v'} > 0$ for some other leaf v' then $q_{u,v'} = 0$: otherwise, the fugitive could modify his strategy and hide in v instead of v', ensuring he doesn't get caught and prolonging the chase. But, as we showed earlier, $q_{u,v'} = 0$ implies $p_{u,v'} = 0$, and that leads to a contradiction. So $p_{u,v}$ cannot equal zero for a leaf $v \neq u$, and so cannot $q_{u,v}$.

So, each of $p_{u,v}$ and $q_{u,v}$ is non-zero if and only if v is a leaf different from u. Our last goal is to calculate the values of $p_{u,v}$, $q_{u,v}$ and x_u. To do that, we will write down several equations on these numbers. Denote by L the set of leaves of the tree. Firstly, x_u, as the mathematical expectation of the duration of the chase, satisfies:

$$x_u = \sum_{v \in L \setminus \{u\}} F(u, v) \qquad \text{where } F(u, v) = p_{u,v}(\operatorname{dist}(u, v) + (1 - q_{u,v})x_v).$$

Also, as a solution to an optimization problem, $p_{u,v}$ and $q_{u,v}$ satisfy *complementary slackness conditions* (as a part of Karush–Kuhn–Tucker conditions). Namely, note that if, for a fixed u, $P_u(v) = \frac{\partial}{\partial p_{u,v}} F(u, v) = (\operatorname{dist}(u, v) + (1 - q_{u,v})x_v)$ differs for two different vertices $v \in L \setminus \{u\}$, then it would be profitable for the police officer to always walk into the vertex with the lower value of $P_u(v)$, and that would violate the Nash equilibrium. Similarly, for a fixed u, $Q_u(v) = \frac{\partial}{\partial q_{u,v}} F(u, v) = -p_{u,v}x_v$ should be equal among all $v \in L \setminus \{u\}$, otherwise the fugitive would have the incentive to only hide in the vertices v with the maximum $Q_u(v)$ and violate the Nash equilibrium from his side.

The aforementioned complementary slackness conditions can be reformulated in the following way: if in the function $F(u, v) = p_{u,v}(\operatorname{dist}(u, v) + (1 - q_{u,v})x_v)$ one replaces the array $p_{u,v}$ with any other array with a unit sum, which vanishes on $v \notin L \setminus \{u\}$, then the value of $F(u, v)$ does not change. Similarly, if one replaces the array $q_{u,v}$ with any other array with a unit sum, which vanishes on $v \notin L \setminus \{u\}$, then the value of $F(u, v)$ does not change. (But if one does the both modifications, then $F(u, v)$ *might* change.)

Now we are ready to write a system of equations on the numbers x_u. All numbers $p_{u,v}x_v$ are equal to each

other, so, for a fixed $v \in L \setminus \{u\}$,

$$1 = \sum_{w \in L \setminus \{u\}} p_{u,w} = \sum_{w \in L \setminus \{u\}} p_{u,w} x_w \cdot \frac{1}{x_w} = \sum_{w \in L \setminus \{u\}} p_{u,v} x_v \cdot \frac{1}{x_w} = p_{u,v} x_v \cdot \sum_{w \in L \setminus \{u\}} \frac{1}{x_w},$$

therefore,

$$p_{u,v} = \frac{1/x_v}{\sum_{w \in L \setminus \{u\}} 1/x_w}.$$

Let us substitute this expression in the formula for $F(u, v)$:

$$F(u, v) = \frac{1/x_v}{\sum_{w \in L \setminus \{u\}} 1/x_w} (\operatorname{dist}(u, v) + (1 - q_{u,v}) x_v).$$

It was said earlier that the array $q_{u,v}$ can be chosen arbitrarily if the unit sum is preserved. We will fix a vertex $t \in L \setminus \{u\}$ and take $q_{u,v} = [v = t]$ — that is, $q_{u,t} = 1$ and the rest of the $q_{u,v}$ are taken zero. After this substitution, $F(u, v)$ stays the same:

$$F(u, v) = \frac{1/x_v}{\sum_{w \in L \setminus \{u\}} 1/x_w} (\operatorname{dist}(u, v) + x_v - [v = t] x_t).$$

The formula for x_u takes the form:

$$x_u = \sum_{v \in L \setminus \{u\}} \frac{1/x_v}{\sum_{w \in L \setminus \{u\}} 1/x_w} (\operatorname{dist}(u, v) + x_v - [v = t] x_v).$$

The $x_v - [v = t] x_v$ part can be removed from under the summation sign:

$$x_u = \frac{|L \setminus \{u\}|}{\sum_{w \in L \setminus \{u\}} 1/x_w} - \frac{1}{\sum_{w \in L \setminus \{u\}} 1/x_w} + \sum_{v \in L \setminus \{u\}} \frac{1/x_v}{\sum_{w \in L \setminus \{u\}} 1/x_w} \operatorname{dist}(u, v).$$

This already yields a rather convenient expression:

$$x_u = \frac{|L| - 2 + [u \notin L] + \sum_{v \in L \setminus \{u\}} \operatorname{dist}(u, v)/x_v}{\sum_{v \in L \setminus \{u\}} 1/x_v}.$$

However, to get a bit more symmetric form, one can multiply it by the denominator:

$$x_u \cdot \sum_{v \in L \setminus \{u\}} 1/x_v = |L| - 2 + [u \notin L] + \sum_{v \in L \setminus \{u\}} \operatorname{dist}(u, v)/x_v.$$

Then, if $u \in L$, we can add one to both sides (or, equivalently, add $[u \in L]$ to both sides):

$$x_u \cdot \sum_{v \in L} 1/x_v = |L| - 1 + \sum_{v \in L} \operatorname{dist}(u, v)/x_v.$$

Finally, let us divide it back:

$$x_u = \frac{|L| - 1 + \sum_{v \in L} \operatorname{dist}(u, v)/x_v}{\sum_{v \in L} 1/x_v}.$$

Unfortunately, this system of equations is very unlikely to have a clear way to be solved analytically in general case. Even for small trees the exact formulae begin to look gargantuan, and there are really few classes of trees with neat formulae (e.g. star, bamboo, pair of equal stars connected by a bamboo). Instead, one can take some initial approximation, e.g. $x_u = 1$ or $x_u = (n - 1)^2$, and iteratively apply the formula above to get a more and more precise answer. This process seems to converge relatively fast, although jury does not have a proof of that. What is worth noting is that, at first, it is reasonable to only calculate x_u for $u \in L$ (since these are the only x_u which occur in the right hand side), and in the end, when the

leaf expectations are already calculated with satisfying precision, calculate the rest of the expectations with one iteration over all $u \in \overline{L}$.

One more possible speedup is to work with $y_u = 1/x_u$ instead of x_u, since this would lead to a much smaller number of divisions:

$$y_u = \frac{\sum_{v \in L} y_v}{|L| - 1 + \sum_{v \in L} \operatorname{dist}(u, v) y_v}.$$

This final approach works in $\mathcal{O}\big(n\ell + q\ell^2 + n\ell\big) = \mathcal{O}\big(n\ell + q\ell^2\big)$, where $n = |V|$, $\ell = |L|$ and q is the number of iterations: the first $n\ell$ summand stands for finding the pairwise distances between the vertices and the leaves, $q\ell^2$ stands for the iterative algorithm for finding the leaf values of x_u, and the last $n\ell$ summand stands for calculating the rest of the values of x_u. In practice, for $n \le 100$, $q = 1500$ or $q = 2000$ and a 64-bit floating point number type were enough to pass all tests, depending on the quality of the initial approximation. The time limit for this problem is pretty loose, so even Python solutions should pass with a reasonable implementation.

Problem G. Great City Saint Petersburg

Problem author and developer: Alisa Sayutina

Define sequences p and q such that $p_i = max(a_0, a_1, \ldots, a_i)$, and $q_i = max(a_i, a_{i+1}, \ldots, a_{n-1})$

Observe that the amount of accumulating rain is $\sum_{i=0}^{n-1} min(p_i, q_i) - a_i$. It's easy to update $\sum_i a_i$ based on the range increase operation, so let's focus on $\sum_{i=0}^{n-1} min(p_i, q_i)$.

A critical observation is that p_i is (non-strictly) increasing function, while q_i is (non-strictly) decreasing function. Another critical observation is that operation $a[l..r]$`+=1` only causes at most one range update for p: $p[l'..r']$`+=1` (if p would change at all, the first element to increase would be the leftmost element on segment $[l; r]$ larger than $max(a_0, \ldots, a_{l-1})$, and this increase will go until first sufficiently large element to the right of it. Both indices can be found with standard segment tree algorithms in $O(\log n)$ time). Similarly there at most one range update $q[l''..r']$`+=1`.

There are some different ways how to finalize calculation give those ideas. For instance, one can store a segment tree for $p_i - q_i$ (observe this is monotonically increasing function), and then at each `p[l..r] +=1` and `q[l..r]+=1` range operations caused by above, recompute how $\sum_{i=0}^{n-1} min(p_i, q_i)$ is changing. Specifically, $min(p_i, q_i)$ will increase during `p[l..r]+=1` operation for indices i such that $p_i < q_i$.

Problem H. Hypercatapult Commute

Problem author and developer: Pavel Mavrin

Let's build a directed graph with edges $a_i \to b_i$. Consider a (weakly) connected component in this graph. Let the size of this component be k.

For the acyclic component, we can deliver all the passengers in $k-1$ launches, making the chain of launches in the topological order.

If the component is not acyclic, we need to make at least one cycle of launches, so we need k launches. It can be shown that if it is possible to deliver all the passengers, it is possible to do it by making one big cycle of launches. Now let's look at the first launch, say $v \to u$. If we remove this launch, it will affect only passengers with $a_i = v$. So if we remove these edges, the remaining edges must form acyclic graph. We can iterate over all vertices v and check if removing this vertex makes the component acyclic, in $O(nm)$ time.

Problem I. Innovative Washing Machine

Problem author and developer: Ivan Safonov

Let our polygon be $A_1 A_2 \ldots A_n$.

Let us define the function $f(\varphi)$ as the value of pressure imbalance for the polygon rotated by angle φ.

As the answer to the problem we want to calculate $\frac{1}{2\pi}\int\limits_0^{2\pi} f(\varphi)d\varphi$.

Let us define the function $p_i(\varphi) = -x_i \cdot \sin(\varphi) + y_i \cdot \cos(\varphi)$. It is a coordinate of projection of vertex i into the orthogonal direction to the water level.

If vertices $u_1, u_2, \ldots, u_k$ are underwater, when

$$f(\varphi) = \frac{1}{k}\sum_{i=1}^{k}\left(\max_{j=1}^{k} d_{u_j} - d_{u_i}\right) = \frac{1}{k}\sum_{i=1}^{k}\left(p_{u_i}(\varphi) - \min_{j=1}^{k} p_{u_j}(\varphi)\right)$$

$$f(\varphi) = -\left(\frac{1}{k}\sum_{i=1}^{k} x_{u_i}\right)\sin(\varphi) + \left(\frac{1}{k}\sum_{i=1}^{k} y_{u_i}\right)\cos(\varphi) - \min_{j=1}^{k} p_{u_j}(\varphi)$$

Let's note, that $\min\limits_{j=1}^{k} p_{u_j}(\varphi) = \min\limits_{i=1}^{n} p_i(\varphi)$, because the global lowest vertex will be underwater. So:

$$f(\varphi) = -\left(\frac{1}{k}\sum_{i=1}^{k} x_{u_i}\right)\sin(\varphi) + \left(\frac{1}{k}\sum_{i=1}^{k} y_{u_i}\right)\cos(\varphi) - \min_{i=1}^{n} p_i(\varphi)$$

Let us find the integral of $\min\limits_{i=1}^{n} p_i(\varphi)$ and $-\left(\frac{1}{k}\sum\limits_{i=1}^{k} x_{u_i}\right)\sin(\varphi) + \left(\frac{1}{k}\sum\limits_{i=1}^{k} y_{u_i}\right)\cos(\varphi)$ separately.

First part.

We want to calculate $\int\limits_0^{2\pi} \min\limits_{i=1}^{n} p_i(\varphi)d\varphi$. Note, that the point A_i will be lowest for all $\varphi \in [\alpha(\overrightarrow{A_{i-1}A_i}), \alpha(\overrightarrow{A_iA_{i+1}})]$, where $\alpha(\overrightarrow{v})$ is a polar angle of vector $\overrightarrow{v}$.

So the integral is $\sum\limits_{i=1}^{n} \int\limits_{\alpha(\overrightarrow{A_{i-1}A_i})}^{\alpha(\overrightarrow{A_iA_{i+1}})} p_i(\varphi)d\varphi = \sum\limits_{i=1}^{n} y_i \int\limits_{\alpha(\overrightarrow{A_{i-1}A_i})}^{\alpha(\overrightarrow{A_iA_{i+1}})} \cos(\varphi)d\varphi - \sum\limits_{i=1}^{n} x_i \int\limits_{\alpha(\overrightarrow{A_{i-1}A_i})}^{\alpha(\overrightarrow{A_iA_{i+1}})} \sin(\varphi)d\varphi$.

Integration of sin and cos on segment is simple, this sum can be calculated.

Magically, expression can be simplified further, and the integral equals to minus perimeter of the polygon: $-\sum\limits_{i=1}^{n} |A_iA_{i+1}|$.

Second part.

We want to calculate $\int\limits_0^{2\pi}\left(-\left(\frac{1}{k}\sum\limits_{i=1}^{k} x_{u_i}\right)\sin(\varphi) + \left(\frac{1}{k}\sum\limits_{i=1}^{k} y_{u_i}\right)\cos(\varphi)\right)d\varphi$.

During the rotation of the polygon the set of underwater vertices $u_1, u_2, \ldots, u_k$ is changing. In which angles the set is changing? Note, that if the set of underwater vertices is changing in some angle φ, one of the vertices A_i lies on the water level in this moment. Vertex A_i can be left or right vertex of the water level segment.

For all vertices A_i let us find the angles $\varphi_{L,i}$, $\varphi_{R,i}$ in which A_i is left and right vertex of the water level segment, respectively. These angles can be found with two pointers method: let us iterate i in clockwise or counterclockwise order and maintain the set of currently underwater vertices (these vertices will be consecutive). So we maintain the pointer to vertex j, such that $area(A_iA_{i+1}\ldots A_j) \le s$ and $area(A_iA_{i+1}\ldots A_{j+1}) > s$ (for counterclockwise traversal). This index j can be simply maintained, because we can maintain the sum of pseudoscalar products $\overrightarrow{A_p} \times \overrightarrow{A_{p+1}}$ for $p \in [i, j)$ to maintain the

area. After that, to find the angle $\varphi_{L,i}$ we should look at the triangle $\triangle A_i A_j A_{j+1}$ and find a segment that divides the triangle into two pieces with given ratio of areas.

So, such event angles $\varphi_{L,i}$, $\varphi_{R,i}$ where a set of underwater vertices is changing can be found in $O(n)$. After that, we can sort events and calculate our integral, because on each segment between consecutive events, values $\frac{1}{k}\sum_{i=1}^{k} x_{u_i}$, $\frac{1}{k}\sum_{i=1}^{k} y_{u_i}$ are constant and we should just integrate sin and cos.

The total complexity of the solution is $O(n \log n)$.

Problem J. Joy of Pokémon Observation

Problem author and developer: Vitaly Goldshtein

For a single species ($s = 1$) in the habitat the answer is 1 if $t \mod l_1 = 0$ and 0 otherwise.

For two species ($s = 2$) in the habitat the problem is to count the number of combinations of $i_a \geq 0$ and $i_b \geq 0$ that $a \times i_a + b \times i_b = t$ (where $a = l_1$ and $b = l_2$). Let $i_a = k_a \times b + j_a$ where $0 \leq j_a < b$. Iterate over j_a and let $R = t - j_a * a$. In case $R \mod b = 0$, we should add to the answer number of combinations of k_a and i_b that $k_a \times a + i_b = \frac{R}{b}$. Or to find the number of k_a so that $k_a \times a \leq \frac{R}{b}$. I. e. we need to add to the result $\lfloor \frac{R/b}{a} \rfloor + 1$.

For three species ($s = 3$) in the habitat the problem is to count the number of combinations of $i_a \geq 0$, $i_b \geq 0$ and $i_c \geq 0$ that $a \times i_a + b \times i_b + c \times i_c = t$ (where $a = l_1$, $b = l_2$ and $c = l_3$). Let $i_a = k_a \times c + j_a$ and $i_b = k_b \times c + j_b$ where $0 \leq j_a < c$ and $0 \leq j_b < c$. Iterate over j_a and j_b and let $R = t - j_a * a - j_b * b$. In case $R \mod c = 0$, we should add to the answer number of combinations of k_a, k_b and i_c that $k_a \times a + k_b \times b + i_c = \frac{R}{c}$. Or to find the number of k_a and k_b so that $k_a \times a + k_b \times b \leq \frac{R}{c}$. Let $k_a = r_a \times b + e_a$ (where $0 \leq e_a < b$). Iterate over e_a and let $\hat{R} = \frac{R}{c} - e_a \times a$. So we need to add to the result the number of combinations of r_a and k_b, so that $r_a \times a + k_b \leq \lfloor \frac{\hat{R}}{b} \rfloor$. Let $\bar{R} = \lfloor \frac{\hat{R}}{b} \rfloor$. That can be computed using an arithmetic progression $\sum_{r_a=0}^{\lfloor \bar{R}/a \rfloor} \bar{R} - r_a \times a$.

Problem K. Kim's Quest

Problem author: Elena Kryuchkova; problem developer: Ilya Zban

This task is solvable with dynamic programming. We can process numbers one by one, and calculate the number of Harmonious Subsequences in the first i numbers of the initial sequence, such that the last number is $x \in \{0, 1\}$, and the number before the last is $y \in \{0, 1\}$ (or that there are less than 2 numbers). Knowing these values, it's easy to add $i + 1$-th number, you just need to make sure that $(x + y + a_{i+1}) \mod 2 \neq 1$.

This idea gives us a solution in $\mathcal{O}(n)$ memory and time.

Problem L. LOL Lovers

Problem author and developer: Mikhail Ivanov

It is easy to see that if the total number of loaves is not two and the first or the last letter is '`L`', you can cut this single letter from the rest of the string, and this division satisfies all requirements. The same goes for the case when the total number of onions is not two and the first or the last letter is '`O`'. The only cases not solvable by cutting the first or the last letters are: '`LO...OL`', '`OL...LO`', '`LOLO`', '`OLOL`', '`OOLL`' and '`LLOO`' but it is easy to see that only the last two of them are solvable at all. This gives the solution which works in $\mathcal{O}(n)$ time. However, since $n \leq 200$ in this problem, there are plenty of slower approaches which will still pass the tests, including iterating over all possible divisions and checking each in linear time.