

UCPC 2026 - I stage

Ukrainian Collegiate Programming Contest

October 3, 2026

Editorial

A Antivirus
B Collectible Pairs
C Dance of the Remainders
D Even and Odd
E Four Numbers
F Mixed-up Readings
G Number Game
H Pattern
I Round Table
J Scales
K Sum and Product
L Three Computers
M UCPC Garland
N Weather Diary

A Antivirus

Simulating each run separately costs up to $\Theta(nk)$, so we merge them. For a folder v let b_v be the largest number of levels that some run can still go deeper than v:

$$b_v = \max\{\, d_j - \operatorname{dist}(c_j, v) \,\},$$

where $\operatorname{dist}(c_j, v)$ is the number of levels from c_j down to v, and the maximum is taken over runs j such that v lies inside c_j (at any depth) or $v = c_j$ ($b_v = -1$ if there is none). Folder v is checked exactly when $b_v \geq 0$.

A run reaches $v \neq c_j$ only through a_v, where it has one level more left. Hence

$$b_v = \max\Big(\max_{j:\, c_j = v} d_j,\ b_{a_v} - 1\Big),$$

and for the root only the first term remains. Since $a_v < v$, computing b_v for $v = 1, 2, \ldots, n$ in this order always uses an already known b_{a_v}; no DFS or explicit tree is needed. The answer is the number of v with $b_v \geq 0$. Repeated runs on one folder must be combined by the maximum, not overwritten.

The complexity is $O(n + k)$.

B Collectible Pairs

For a huge m_i we cannot search; instead we build x with a tiny digit sum. Take $x = c \cdot 10^B$ where the digit sum of c is 8. Then x has digit sum 8 and $x+1$ (which just replaces the last zero by a one) has digit sum 9. If $B \geq 3$, then $8 \mid 1000 \mid 10^B$, so $8 \mid x$. A number with digit sum 8 is $\equiv 8 \pmod 9$, so $x+1 \equiv 0 \pmod 9$. Hence x and $x+1$ form a beautiful pair for every such c and every $B \geq 3$.

It remains to fit the pair into the bundle: $m_i \leq x$ and $x+1 \leq 2m_i$. Let m_i have $L \geq 5$ digits, let p be the number formed by its first two digits, and let $B = L-2 \geq 3$, so that $p \cdot 10^B \leq m_i < (p+1) \cdot 10^B$. It is enough to choose c with $p < c < 2p$: then $m_i < (p+1) \cdot 10^B \leq x$ and $x+1 < 2p \cdot 10^B \leq 2m_i$. Five values of c with digit sum 8 cover all $p \in [10, 99]$:

$$p \in [10, 16] \to 17, \quad [17, 25] \to 26, \quad [26, 34] \to 35, \quad [35, 61] \to 62, \quad [62, 99] \to 107,$$

since in each range $c > p$ and $c < 2 \cdot$ (smallest p). So for $m_i \geq 10^4$ the answer always exists, and we print x: the digits of c followed by $L-2$ zeros.

The construction needs $B \geq 3$, that is, at least five digits; with fewer digits $8 \nmid 10^B$ and it fails. All $m_i < 10^4$ are handled by precomputation: mark the beautiful numbers up to $2 \cdot 10^4$, compute for every m the smallest $x \geq m$ such that $(x, x+1)$ is a beautiful pair (a backward pass), and answer it if $x+1 \leq 2m$. If the smallest such x does not fit, no larger one does. The values of x with x, $x+1$ both beautiful start with $1, 2, \ldots, 9, 20, 80, 110, \ldots$, and this table shows that the answer is -1 exactly for $m_i = 10$ (the pair $20, 21$ sticks out of $[10, 20]$) and for $21 \leq m_i \leq 40$ (the next pair is $80, 81$, and $81 \leq 2m_i$ needs $m_i \geq 41$). For every $41 \leq m_i < 10^4$ a beautiful pair exists.

The complexity is $O(10^4)$ for the table plus $O(\text{total length})$ for the collectors.

C Dance of the Remainders

Let r be the common remainder. A remainder is smaller than the divisor, so $r = a \bmod b < b$, $r = b \bmod c < c$ and $r = c \bmod a < a$; thus r is smaller than each of a, b, c.

Suppose $a < b$. Then $a \bmod b = a$, so $r = a$, which contradicts $r < a$. Hence $a \geq b$. In the same way $b \geq c$ and $c \geq a$, so $a \geq b \geq c \geq a$, that is, $a = b = c$. Conversely, for every triple with $a = b = c$ the dance is harmonious: all three remainders are 0.

So the dance is harmonious exactly for the triples (x, x, x) with $1 \leq x \leq \min(n, m, k)$, and the answer is $\min(n, m, k)$. The complexity is $O(1)$ per test case.

D Even and Odd

Write $\Delta(A) = S_{\text{odd}} - P_{\text{odd}} - S_{\text{even}} + P_{\text{even}}$. A digit d at position i adds $+d$ or $-d$ through the S-terms ($+$ if d is odd) and $+d$ or $-d$ through the P-terms ($+$ if i is even). So it contributes $2d$ if d is odd and i is even, $-2d$ if d is even and i is odd, and 0 otherwise: A is balanced if and only if the sum of odd digits at even positions equals the sum of even digits at odd positions.

Let M be N followed by a block of k digits. The block occupies positions $0, \dots, k-1$ and every digit of N moves up by k, so the contribution of N depends only on $k \bmod 2$. Compute in $O(|N|)$ the values D_0 and D_1 of $\Delta/2$ over the digits of N shifted by an even and by an odd amount. The block must then satisfy $x - y = T$ with $T = -D_{k \bmod 2}$, where x is the sum of the odd digits on its $e = \lceil k/2 \rceil$ even positions and y is the sum of the even digits on its $o = \lfloor k/2 \rfloor$ odd positions.

Feasibility in $O(1)$. The possible values of y are exactly the even numbers in $[0, 8o]$. A sum of j odd digits has the parity of j and takes every value of this parity in $[j, 9j]$. Taking the union over $j \le e$, the reachable x of parity p are all numbers of parity p in $[p, 9E_p]$, where E_p is the largest integer $\le e$ of parity p (nothing is reachable if $p = 1$ and $e = 0$). As $x \equiv T \pmod 2$, take $p = T \bmod 2$; a block exists if and only if $\max(T, p) \le \min(T + 8o,\ 9E_p)$. The parity is essential: for instance, two odd digits never sum to 11.

The smallest M. Since N has no leading zero, a shorter M is smaller, and blocks of equal length compare lexicographically. Find the minimal k by testing $k = 0, 1, 2, \dots$, then fill the block from position $k-1$ down to 0, each time with the smallest digit after which the remaining positions are still feasible. At an even position it suffices to try $0, 1, 3, 5, 7, 9$, at an odd one $0, 2, 4, 6, 8$: every other digit contributes nothing, just like 0.

Existence and length. An explicit block always exists. If $T \ge 0$, put j odd digits on the lowest even positions, where j is the smallest integer $\ge \lceil T/9 \rceil$ with $j \equiv T \pmod 2$; this needs $k \ge 2j - 1$. If $T < 0$, put a 1 at position 0 when T is odd, and write $y = |T|$ (or $|T| + 1$) as a sum of $\lceil y/8 \rceil$ even digits on the lowest odd positions; this needs $k \ge 2\lceil y/8 \rceil$. As $|D_0|, |D_1| \le 9\lceil |N|/2 \rceil$, the minimal k is at most $\frac{9}{8}|N| + O(1)$, so M can be more than twice as long as N, but the search and the construction stay linear.

The complexity is $O(|N| + |M|) = O(|N|)$ per query.

E Four Numbers

The sum does not depend on the order of the indices, so we choose a multiset of four elements of the array. Every such multiset splits into two pairs $\{i, j\}$ and $\{k, t\}$, and conversely any two pairs $x \le y$ and $z \le w$, even the same pair taken twice, give a valid quadruple.

Build all $s = n(n+1)/2$ pair sums $a_x + a_y$ with $x \le y$ and sort them: $p_1 \le p_2 \le ... \le p_s$. The answer is the maximum of $p_l + p_r \le m$ over $l \le r$; note that $l = r$ is allowed (it is needed, e.g., for $n = 1$). Find it with two pointers: start with $l = 1$, $r = s$ and, while $l \le r$, if $p_l + p_r \le m$, update the answer and increase l (for this l no larger partner remains), otherwise decrease r (it exceeds m even with the smallest remaining p_l). If no sum fits, which happens exactly when $4 \min a_i > m$, print -1. The answer can be negative, zero or -1, so keep a separate flag that some sum fits instead of a sentinel value. All sums are at most $4 \cdot 10^{18}$ in absolute value and fit into signed 64-bit integers.

The two-pointer pass takes $O(n^2)$ time; the only non-linear step is sorting about $4.5 \cdot 10^6$ numbers, a single sequential sort with a small constant. Formally the complexity is $O(n^2 \log n)$ time and $O(n^2)$ memory; replacing the sort with radix sort makes it $O(n^2)$. The time limit is tight: $O(n^2 \log n)$ solutions with a large constant, such as a binary search in the sorted sums for each of the n^2 pairs (random memory access) or a balanced search tree of pair sums, may not fit, and $O(n^3)$ solutions do not.

F Mixed-up Readings

Sort the unknown array: $a_1 \le a_2 \le \ldots \le a_n$. We recover it element by element, always taking the smallest reading that is not yet explained.

Key observation. Every reading of a group T is at least $\max_{i \in T} a_i$: a sum of positive numbers is at least each of its terms, and an OR has every bit of each operand, so it is at least each operand. Suppose $a_1, \ldots, a_k$ are already known, and remove from the log the $2 \cdot (2^k - 1)$ readings of all non-empty groups $S \subseteq \{1, \ldots, k\}$. The remaining readings belong to groups T that contain some index $j > k$, so each of them is at least $a_j \ge a_{k+1}$. The group $\{k+1\}$ itself gives exactly a_{k+1}. Hence the minimum of the remaining multiset is a_{k+1}. By induction this recovers the whole array, and it also shows that the answer is unique up to order.

Removing the readings. When a new element $x = a_{k+1}$ is found, the new groups are $S \cup \{k+1\}$ for all $S \subseteq \{1, \ldots, k\}$, including $S = \varnothing$. Their readings are $\mathrm{sum}(S) + x$ and $\mathrm{or}(S) \mid x$. Keep two arrays with $\mathrm{sum}(S)$ and $\mathrm{or}(S)$ for all 2^k subsets of the found elements (starting from the empty set with 0 and 0); each new element doubles them by appending $\mathrm{sum}(S) + x$ and $\mathrm{or}(S) \mid x$.

Now sort b and scan it in increasing order, keeping a min-heap of readings that are explained but not yet matched. If the current b_p equals the top of the heap, it is explained: pop the heap and move on. Otherwise b_p is the smallest unexplained reading, i.e. the next element x; push the $2 \cdot 2^k$ new readings. The pair for $S = \varnothing$ equals x twice and matches b_p itself and its twin copy. All pushed values are at least $x = b_p$, so none of them is already behind the scan, and the scan computes exactly the multiset difference. Stop after the n-th element. Note that equal readings are frequent (e.g. $\mathrm{sum}(S) = \mathrm{or}(S)$ for bit-disjoint elements), so the explained values must be counted with multiplicity, not stored as a set.

The complexity is $O(2^n \cdot n)$ per test case (sorting and the heap).

G Number Game

In the first step of a move Marta may erase and append zeros anywhere, so a number with c ones can be turned into any number with exactly c ones (its leading digit is still a one). Hence only c matters: one move turns the number on the board into $x+1$ for an arbitrary x with $\text{popcount}(x) = c$.

How does c change in one move? If x ends with exactly k ones ($0 \le k \le c$), then $+1$ turns these k ones into zeros and the zero before them into a one, so $x+1$ has $c-k+1$ ones. Every k is possible, so after a move the count can be any value in $[1, c+1]$: it can drop arbitrarily, but it grows by at most one.

The last move must increase $x = m-1$, so just before it the board must hold a number with exactly $p = \text{popcount}(m-1)$ ones; conversely, any such number can be turned into $m-1$. Let $c = \text{popcount}(n)$. The answer is:

- 0 if $n = m$ (check this first: it includes $n = m = 1$);
- -1 if $m = 1$, since the board would have to hold $x = 0$;
- 1 if $c = p$;
- 2 if $c > p$: one move to drop the count to $p \ge 1$, one more to reach m;
- $p - c + 1$ if $c < p$: the count needs at least $p - c$ moves to grow to p, and exactly $p - c$ suffice.

Subtracting 1 turns the trailing zeros of m into ones and its lowest one into a zero, so $p = \text{popcount}(m) - 1 + (\text{number of trailing zeros of } m)$. The numbers have up to 1000 binary digits, so everything is computed directly on the strings. The complexity is $O(|n| + |m|)$ per test case.

H Pattern

Let $u(n)$ be the answer. Write a stamp as the word $w = w_1 \dots w_n$ of its colors. A border of w is a word of length l, $1 \le l \le n-1$, that is both a prefix and a suffix of w; a stamp is successful iff it has no border. We count the unsuccessful stamps by their shortest border.

Lemma. The shortest border b of a stamp w with n cells is itself successful, and $2|b| \le n$.

Proof. A border c of b is a prefix and a suffix of w, i.e. a shorter border of w; so b is successful. Now suppose $l = |b| > n/2$ and let $s = n - l < l$. The two occurrences of b in w start at positions 1 and $s+1$ and share the piece $w_{s+1} \dots w_l$. It is a suffix of the first occurrence and a prefix of the second, so it is a border of b of length $l - s \ge 1$, a contradiction. □

Conversely, take any successful b with $2|b| \le n$ and any coloring x of $n - 2|b|$ cells. Then $w = bxb$ has shortest border exactly b: a shorter border of w would be a prefix and a suffix of b, i.e. a border of b. Hence the unsuccessful stamps with n cells are in bijection with such pairs (b, x), and

$$u(n) = k^n - \sum_{j=1}^{\lfloor n/2 \rfloor} k^{\,n-2j}\, u(j).$$

Evaluated directly, this is $O(n^2)$. Subtract k times the same identity for $n-1$: k^n cancels, and so does every term with $j \le \lfloor (n-1)/2 \rfloor$. For odd n nothing is left; for $n = 2m$ only the term $j = m$, i.e. $u(m)$, survives:

$$u(1) = k, \qquad u(2m+1) = k\,u(2m), \qquad u(2m) = k\,u(2m-1) - u(m).$$

For each query compute $u(1), \dots, u(n)$ modulo $10^9 + 7$ with 64-bit products, fixing the sign after the subtraction; one array serves all queries.

The complexity is $O(n)$ per query, $O(\sum n)$ in total.

I Round Table

Unroll the circle: write the students twice as positions $1, 2, \ldots, 2n$, where positions x and $x+n$ are the same student. A team (l_i, r_i) becomes the segment $[l_i, e_i]$ with $e_i = r_i$ if $l_i \le r_i$ and $e_i = r_i + n$ otherwise (its length is between 1 and n; for the whole circle $e_i = l_i + n - 1$), together with its copy $[l_i + n, e_i + n]$. A window $[p, p + n - 1]$ with $1 \le p \le n$ contains every student exactly once, so pairwise disjoint segments inside one window are pairwise disjoint teams.

Inside a fixed window we have the classic problem on a line, solved greedily: from the current position x, take the segment starting at x or later with the smallest end, and continue right after it (the first segment of any solution can be exchanged for the greedy one, which ends no later). So let

$$\mathrm{nxt}(x) = 1 + \min\{e : [s, e] \text{ is a segment or a copy, } s \ge x\},$$

computed for all $x \le 2n$ by suffix minima in $O(n + m)$; if there is no such segment, or the value exceeds $2n$, set $\mathrm{nxt}(x) = 2n + 1$, and let $2n + 1$ be a fixed point. The t-th team taken by the greedy from p ends at $\mathrm{nxt}^t(p) - 1$, so the greedy in the window of p forms $f(p)$ teams, where

$$f(p) = \max\{t : \mathrm{nxt}^t(p) \le p + n\}.$$

The answer is $\max_{1 \le p \le n} f(p)$. Indeed, each $f(p)$ is achievable. Conversely, take an optimal set and let p be the first student of one of its teams: no team of the set contains both student p and the student before p (it would share student p with that team), so in the window of p all of them are segments, and the line greedy finds at least as many. A single cut is not enough: in the second example the greedy from student 1 forms only 2 teams, while starting from student 3 gives 3.

To evaluate all $f(p)$ quickly, use binary lifting: $\mathrm{up}_0 = \mathrm{nxt}$ and $\mathrm{up}_j(x) = \mathrm{up}_{j-1}(\mathrm{up}_{j-1}(x))$, that is, $\mathrm{up}_j = \mathrm{nxt}^{2^j}$, for $2^j \le n$ (always $f(p) \le n$). Since $\mathrm{nxt}(x) > x$ for $x \le 2n$, the sequence $\mathrm{nxt}^t(p)$ is non-decreasing in t, so $f(p)$ is found like a binary search: start with $c = p$ and go through j from the largest down; whenever $\mathrm{up}_j(c) \le p + n$, set $c \leftarrow \mathrm{up}_j(c)$ and add 2^j to the count. This is $O(\log n)$ per start.

The complexity is $O(m + n \log n)$ time and $O(n \log n)$ memory.

J Scales

A comparison of two numbers is decided by the highest bit in which they differ. We find the bits of a and b from the top. Let A and B be the already known bits of a and b above bit j, and use only weighings (x, y) with $x = A \oplus s$, $y = B \oplus t$ for some $s, t < 2^{j+1}$. The known bits cancel, so such a weighing compares $a' \oplus s$ with $b' \oplus t$, where $a' = a \bmod 2^{j+1}$ and $b' = b \bmod 2^{j+1}$ are the unknown parts. We maintain c, the result of comparing a' with b'; initially $j = n - 1$, and the weighing $(0, 0)$ gives c.

Let α and β be bit j of a and b.

- If c is `=`, then $a' = b'$, so $\alpha = \beta$, and one weighing $(x, y) = (A \oplus 2^j, B)$ finds the bit: flipping bit j makes it the only difference, so the answer is `>` if and only if $\alpha = \beta = 0$; c stays `=`.
- Otherwise make the weighings $(A \oplus 2^j, B)$ and $(A, B \oplus 2^j)$; let r_1 and r_2 be the answers. If $\alpha = \beta$, the flipped bit is the highest difference, so r_1 and r_2 are strict and opposite (r_1 is `>` if and only if $\alpha = \beta = 0$). If $\alpha \neq \beta$, the flip makes bit j equal, and both answers are the comparison of the lower j bits. So $r_1 = r_2$ if and only if $\alpha \neq \beta$; then c tells which number has the one, and r_1 is the new c.

This is the classical solution with $2n + 1$ weighings. One observation brings it down to $2n$: at bit 0 with c not `=`, the unknown parts are single bits and they differ, so c tells which one is 1 and no weighing is needed. If c is never `=`, bits $n - 1, \ldots, 1$ cost two weighings each and bit 0 is free: $1 + 2(n - 1) = 2n - 1$ weighings. If c becomes `=` at bit $j \geq 1$, each of the j lower bits costs one weighing, $1 + 2(n - j) + j = 2n - j + 1 \leq 2n$ in total; if c is `=` from the start, $n + 1 \leq 2n$. (Even $2n - 1$ weighings suffice: if r_1 is `=`, it cannot come from equal bits, so $\alpha \neq \beta$, the lower parts are equal, and r_2 can be skipped.)

The jury may be adaptive, so every bit must be deduced for the worst case: a scheme that uses $2n + 1$ weighings on some pairs and "saves" one by guessing a bit is caught.

The complexity is $O(n)$.

K Sum and Product

One number works only for $s = 1$.

Two numbers. If $a_1 + a_2 = s$ and $a_1 a_2 = 1$, then a_1, a_2 are the roots of $t^2 - st + 1 = 0$, which are real if and only if $s^2 \geq 4$. So for $|s| \geq 2$ the answer is $n = 2$ (one number is impossible, since $s \neq 1$), with $a_{1,2} = (s \pm \sqrt{s^2 - 4})/2$.

Three numbers for all other s, that is, $|s| < 2$ and $s \neq 1$. One number fails because $s \neq 1$, and two fail because any two reals satisfy $a_1 a_2 \leq (a_1 + a_2)^2/4 = s^2/4 < 1$. Three numbers suffice for every s:

- take $a_1 = -1$; then $a_2 + a_3 = s + 1$ and $a_2 a_3 = -1$, so a_2, a_3 are the roots of $t^2 - (s+1)t - 1 = 0$, whose discriminant $(s+1)^2 + 4$ is positive;
- or, for $s < 0$, take s, $1/\sqrt{-s}$, $-1/\sqrt{-s}$; for $s \geq 0$, take $s + 2x$, $-x$, $-x$, where $(s + 2x)x^2 = 1$: the left side increases from 0 to $s + 2 > 1$ on $[0, 1]$, so x is found by binary search.

Precision. Since $|s|$ can be 10^9, for $n = 2$ one root is about $\pm 10^{-9}$, and $(|s| - \sqrt{s^2 - 4})/2$ loses all its digits to cancellation: compute the root of larger magnitude $y = \operatorname{sgn}(s)(|s| + \sqrt{s^2 - 4})/2$ and take the other one as $1/y$, decide the case split on the exact decimal s (for example, on the 64-bit integer $1000 \cdot s$) rather than on a rounded floating-point value, and print about 17 significant digits.

The complexity is $O(1)$ per value of s (with the binary search, about 100 iterations).

L Three Computers

Think of each team member as a walk along the time line: they either wait or solve a problem, which moves them from minute l_i to minute $r_i + 1$. This is a flow with three units.

Network. Compress all values l_i and $r_i + 1$ into points $x_0 < x_1 < \ldots < x_{m-1}$. Add chain edges $x_j \to x_{j+1}$ with capacity 3 and cost 0 (waiting), and for every problem an edge $l_i \to r_i + 1$ with capacity 1 and cost $-w_i$. Send a flow of value 3 from x_0 to x_{m-1}; the answer is $-$(minimum cost). The end point is $r_i + 1$, not r_i: problems $[1, 3]$ and $[4, 5]$ give edges $1 \to 4 \to 6$ and can follow each other, while $[1, 3]$ and $[3, 5]$ give $1 \to 4$ and $3 \to 6$, which cannot lie on one path.

Why a flow of value 3 is exactly a schedule. All edges go forward, so the network is a DAG, and an integral flow of value 3 splits into three paths from x_0 to x_{m-1} (there are no cycles). The problem edges on one path are pairwise disjoint in time, because a path that uses $l_i \to r_i + 1$ continues from $r_i + 1$; so each path is a valid schedule of one member, and capacity 1 means that no problem is used twice. Conversely, the schedule of each member is such a path (wait along the chain between problems), and three paths use every chain edge at most 3 times. The costs match, so the minimum cost is minus the best total. Min-cost flow with integer capacities has an integral optimum, and the algorithm below finds one.

Initial potentials. The costs are negative, so Dijkstra cannot be used directly. But the network is a DAG with nodes already in topological order, so one pass over $x_0, x_1, \ldots$ gives the shortest distances $\varphi(v)$ from x_0. Then every reduced cost $c(u, v) + \varphi(u) - \varphi(v)$ is nonnegative.

Three Dijkstra runs. Use successive shortest paths: run Dijkstra from x_0 on the reduced costs of the residual graph, add the distances to φ, and push flow along the shortest path to x_{m-1} (by its bottleneck, at most 3 minus the current flow). After an augmentation the reduced costs stay nonnegative (the standard potential argument), and every node stays reachable along the chain, whose residual capacity is at least $3 - \text{flow} \geq 1$, so φ is always finite. Each augmentation pushes at least one unit, so there are at most 3 Dijkstra runs on a graph with $O(n)$ nodes and edges.

For small n, Bellman-Ford or SPFA on every augmentation would do, but at these limits it is too slow: the residual graph has negative undo edges, and these algorithms can take $\Theta(n^2)$ time on it. The answer reaches $2 \cdot 10^{14}$, so use 64-bit integers.

The complexity is $O(n \log n)$ per test case.

M UCPC Garland

Say that lamp i plays role j ($1 \le j \le 4$) if some occurrence of `UCPC` has $i_j = i$, and call a lamp free if it plays no role. In a consistent coloring all occurrences have one coloring (c_1, c_2, c_3, c_4), so every lamp playing role j has color c_j, while free lamps are unrestricted. Conversely, any $c_1, \dots, c_4$ and any colors of the free lamps give a consistent coloring, as long as a lamp playing two roles does not get two different colors. The letters `U`, `C`, `P` are distinct, so only roles 2 and 4 can share a lamp. Hence, with f free lamps, the answer is 3^{f+3} if some lamp plays both roles 2 and 4, and 3^{f+4} otherwise; if `UCPC` does not occur at all, the answer is 3^n.

Let $l_1 < l_2 < l_3 < l_4$ be the occurrence found greedily from the left (each l_j is the first suitable letter after l_{j-1}), and $r_1 < r_2 < r_3 < r_4$ the one found greedily from the right. Since greedy matching is optimal, lamp i whose letter is the j-th letter of `UCPC` can be preceded by the first $j-1$ letters if and only if $l_j \le i$, and followed by the last $4-j$ letters if and only if $i \le r_j$. So lamp i plays role j if and only if it has the right letter and $l_j \le i \le r_j$. In particular, some lamp plays roles 2 and 4 if and only if $l_4 \le r_2$: such a lamp lies in $[l_4, r_2]$, and conversely, if $l_4 \le r_2$, then lamp l_4 itself plays both roles. Note that the inequality is not strict: in the first garland of the example $l_4 = r_2 = 4$.

Two greedy passes and one pass counting the free lamps take $O(n)$ time per garland, so the complexity is $O(\sum n)$ in total.

N Weather Diary

Write `S` for a sunny day and `R` for a rainy one, and split the sequence into maximal runs of days with the same weather. Every one of the $u+v$ weather changes happens between two neighboring runs, and the changes alternate: `SR`, `RS`, `SR`, and so on. Hence $|u-v| \le 1$ is necessary.

If $u+v \ge 1$, this is also sufficient. Take $u+v+1$ runs of length 1 with alternating weather, starting with a sunny day if $u \ge v$ and with a rainy day otherwise; they produce exactly u pairs `SR` and v pairs `RS`. Both kinds of runs are present, so we lengthen some sunny run by e days and some rainy run by g days: every added day creates one pair `SS` or `RR` and does not change the other counts. The total length is $u+v+1+e+g = n$.

If $u = v = 0$, the weather never changes, so all days are sunny or all are rainy. This is possible if and only if $e = 0$ or $g = 0$; this case is the one that is easy to forget.

So the answer is `YES` if and only if $|u-v| \le 1$ and not ($u = v = 0$, $e > 0$ and $g > 0$). The complexity is $O(1)$ per summary.