

UCPC 2026 - II stage

Ukrainian Collegiate Programming Contest

October 24, 2026

Editorial

A Archive Tree
B Bitwise Potholes
C Convenient Shipments
D Days of the Week
E Exam Session
F Four Steps
G Grouped Vertices
H Hall of Songs
I Inconvenient Swaps
J Jar of Colours
K Know This Problem?
L Lost List
M Mosaic Arch

A Archive Tree

Root the tree at s and write $f_v = k_v \cdot f_p$ for every folder $v \neq s$ with parent folder p. Then f_v is the product of the multipliers k on the path from s to v, and two subfolders of the same folder get different names exactly when their multipliers are different, because they share f_p. Multiplying all names in a subtree by the same number keeps the subtree valid, so the subtrees can be optimised independently. Let $h(u)$ be the smallest possible largest name in the subtree of u if $f_u = 1$; then $h(\text{leaf}) = 1$ and the answer is $h(s)$. If a subfolder c of u gets the multiplier k_c, the best its subtree can achieve is $k_c \cdot h(c)$.

Key claim. Let $g_1 \geq g_2 \geq \ldots \geq g_m$ be the values h of the $m \geq 1$ subfolders of u. Then

$$h(u) = \max_{1 \leq i \leq m} i \cdot g_i,$$

that is, the subfolder with the i-th largest value gets the multiplier i.

Proof. First, the multipliers $1, 2, \ldots, m$ suffice: the j-th smallest of m distinct positive integers is at least j, so replacing it by j keeps the multipliers distinct and increases no product. Second, the larger value should get the smaller multiplier. If $g \geq g'$ and $a < b$, then

$$\max(ag,\ bg') \leq bg \leq \max(bg,\ ag'),$$

so swapping the multipliers of two subfolders with $g \geq g'$ that received $b > a$ never increases the maximum (exchange argument). After all such swaps the multipliers $1, \ldots, m$ go to the subfolders in the order of decreasing h.

The values stay small: $i \cdot g_i \leq g_1 + \ldots + g_i$, so by induction $h(u)$ never exceeds the size of the subtree of u. The answer is at most n, although it is a product of many multipliers, and 32-bit integers suffice.

Algorithm. Run a BFS from s to find the parent folders, then process the folders in reverse BFS order: collect the values h of the subfolders, sort them in decreasing order and take $\max_i i \cdot g_i$. Every folder is sorted once as a subfolder, so the sorting costs $O(n \log n)$ in total.

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

B Bitwise Potholes

Every route is smooth iff all roads have the same number of potholes. If all numbers are equal, AND and OR of every route coincide. Conversely, take two roads e and f. Removing e splits the tree into two parts; let a be the endpoint of e in the part without f, and define b for f symmetrically. The route from a to b contains both e and f. AND equals OR on a route iff every bit is the same on all its roads, so the numbers of potholes on e and f are equal.

Bits are independent. An operation changes only bit i, so each bit is solved separately: choose the target value $c \in \{0, 1\}$ of bit i, and let S be the set of roads whose bit i differs from c. The routes used for bit i must flip exactly the roads of S, i.e. their symmetric difference must be S.

The cost of S is $\mathrm{odd}(S)/2$, where $\mathrm{odd}(S)$ is the number of cities with an odd number of roads of S (it is even by the handshake lemma). Lower bound: a route changes the degree parity only of its two endpoints, so every odd city of S is an endpoint of some route, and at least $\mathrm{odd}(S)/2$ routes are needed. Upper bound: pair the odd cities arbitrarily and take the routes between the pairs. Their symmetric difference P has the same odd cities as S. In a tree this determines the set: a road belongs to P (or to S) iff the part on one side of it contains an odd number of odd cities, because the degrees on that side sum to twice the inner roads plus one for the road itself if it belongs to the set. Hence $P = S$.

Counting. Let X_v be the XOR of the numbers of potholes on the roads at city v, and d_v its degree. For the target $c = 0$ the parity of v in S is bit i of X_v; for $c = 1$ it is this bit XOR $(d_v \bmod 2)$. Counting these parities over all cities gives $\mathrm{odd}_c(i) = \mathrm{odd}(S)$ for both targets, and the answer is

$$\sum_{i=0}^{29} \frac{\min(\mathrm{odd}_0(i),\ \mathrm{odd}_1(i))}{2}.$$

Higher bits are 0 on every road and cost nothing. The complexity is $O(30 \cdot n)$ per test case.

C Convenient Shipments

Key observation. Fix a left end i and extend the group to the right. The values $\gcd(a_i, \dots, a_j)$ form a divisor chain, and each time the value changes it becomes a proper divisor of the previous one, so it at least halves. Since $a_i \le 10^9 < 2^{30}$, there are at most 31 distinct values. So the prefix gcds of any segment split into $O(\log A)$ blocks of consecutive prefixes with equal gcd, and the same holds for suffixes.

Segment tree. For each node store: cnt, the number of convenient groups inside it; tot, the gcd of the whole node; pre, the list of distinct prefix gcds in order of growing prefix, each with the number of prefixes having it; and suf, the same for suffixes. A leaf with value x has $\text{cnt} = [x > 1]$ and $\text{pre} = \text{suf} = ((x, 1))$.

To merge a left child L with a right child R: $\text{tot} = \gcd(L.\text{tot}, R.\text{tot})$; pre is L.pre followed by the blocks $(\gcd(L.\text{tot}, g), c)$ for (g, c) in R.pre, with equal neighbours joined, and suf symmetrically. Finally, $\text{cnt} = L.\text{cnt} + R.\text{cnt} + \text{cross}$, where cross counts the convenient groups that cross the middle, i.e. pairs (suffix of L, prefix of R) with $\gcd(s, g) > 1$, weighted by the multiplicities.

Counting crossing pairs. If $\gcd(s, g) > 1$ for a suffix gcd s and a prefix gcd g, then the same holds for every shorter suffix and every shorter prefix, since their gcds are multiples of s and g. So when the suffix grows, the set of prefixes that pair well with it is a prefix of R.pre that can only shrink. Go through L.suf in order of growing length and move a pointer backwards through R.pre while the current pair has gcd 1, keeping the total multiplicity of the remaining prefix blocks. This takes $O(\log A)$ gcd computations per merge.

Operations. An update changes a leaf and re-merges its $O(\log n)$ ancestors. A query $[l, r]$ folds the $O(\log n)$ canonical nodes from left to right with the same merge; the accumulator needs only cnt, tot and suf. The answer can reach $n(n+1)/2 \approx 4.5 \cdot 10^{10}$, so it needs 64-bit integers.

The complexity is $O((n + q) \log n \log A)$ gcd computations, where $A = 10^9$. Memory is $O(n)$, since a node of length k has lists of length at most $\min(k, 31)$.

D Days of the Week

Number the days of the week from 0 (`monday`) to 6 (`sunday`), and let d_1, d_2, d_3 be the days of the week of the 1/8 stage, the 1/4 stage and SEERC. If one stage is held on weekday a and the next one on weekday b, the number of days between them is positive and congruent to $b - a$ modulo 7. So it is at least $g(a, b)$, where $g(a, b) = (b - a) \bmod 7$ taken in $\{0, \dots, 6\}$, except that 0 becomes 7: two stages on the same weekday are on different calendar days, hence at least a week apart.

The number of days from the 1/8 stage to SEERC inclusive is the sum of the two gaps plus one, so it is at least $g(d_1, d_2) + g(d_2, d_3) + 1$. This value is reached by holding every stage on the earliest suitable day after the previous one. The answer is therefore $g(d_1, d_2) + g(d_2, d_3) + 1$, which always lies between 3 and 15.

In C++ and Java, $(b - a) \,\%\, 7$ is negative when $b < a$; use $((b - a) \,\%\, 7 + 7) \,\%\, 7$.

The complexity is $O(1)$ per test case.

E Exam Session

Go through the days in increasing order and decide, for each day, which exam to take on it. **Greedy (earliest deadline first):** among the exams that are not taken yet and whose window has already opened ($l_i \le d$, where d is the current day), take the one with the smallest r_i. If this $r_i < d$, the answer is `NO`; if every exam gets a day, the answer is `YES`.

Why it is correct. Suppose some valid choice of days agrees with the greedy on all days before d, and let e be the exam picked on day d. In that choice $x_e \ge d$: every earlier day holds another exam or was skipped when every remaining l_i was larger. Hence $r_e \ge x_e \ge d$, and the greedy does not answer `NO` on day d. If day d is free in that choice, move e to day d. If day d belongs to another exam f, swap them: e goes to day d, and f goes to day x_e, which fits because $l_f \le d < x_e \le r_e \le r_f$ (the greedy chose e, so $r_e \le r_f$). Either way the choice stays valid and now agrees with the greedy on day d too. By induction, whenever a valid choice exists, the greedy never answers `NO`; and if it answers `YES`, its own days are a valid choice.

Implementation. Sort the exams by l_i and keep the opened ones in a min-heap by r_i. Days go up to 10^9, so we do not walk over them one by one: when the heap is empty, jump the current day straight to the next l_i. Each step of the loop takes one exam out of the heap, so there are exactly n steps.

The same criterion can be stated via Hall's theorem: the answer is `YES` if and only if every segment of days $[a, b]$ contains at most $b - a + 1$ whole windows $[l_i, r_i]$. The greedy checks this implicitly.

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

F Four Steps

For a string of digits u, let c_0, c_{01}, c_{012} and c_{0123} be the numbers of its subsequences equal to `0`, `01`, `012` and `0123`; the number of climbs is c_{0123}. Appending one digit changes exactly one of the counters, and only by adding another one:

$$\texttt{0}\colon\ c_0 \mathrel{+}= 1, \qquad \texttt{1}\colon\ c_{01} \mathrel{+}= c_0, \qquad \texttt{2}\colon\ c_{012} \mathrel{+}= c_{01}, \qquad \texttt{3}\colon\ c_{0123} \mathrel{+}= c_{012}.$$

Indeed, a new occurrence of `01` must end at the new `1`, and its `0` is any of the c_0 earlier zeros; the other digits are similar.

Key observation. We only need $c_{0123} \bmod 4$, and since the updates use nothing but additions, all four counters can be kept modulo 4 from the start: reducing modulo 4 commutes with addition, so the reduced counters after every prefix are exactly the true counters reduced modulo 4. The true counts are huge (the number of climbs can exceed 2^{64}), but they are never needed.

So the whole string is processed by an automaton with $4^4 = 256$ states $(c_0, c_{01}, c_{012}, c_{0123}) \bmod 4$, starting in $(0,0,0,0)$. Let $\mathrm{dp}[q]$ be the number of ways to fill the `?` of the current prefix so that it ends in state q, modulo $998\,244\,353$. A fixed digit moves each $\mathrm{dp}[q]$ to the state given by its update; a `?` adds $\mathrm{dp}[q]$ to each of the four successors of q. The answer is the sum of $\mathrm{dp}[q]$ over the 64 states with $c_{0123} \equiv 0 \pmod 4$. Only the current layer of dp has to be stored; precomputing the successor of every state for every digit keeps the inner loop simple.

All 256 states are needed: for instance, keeping c_0 only modulo 2 is wrong, because c_0 is later multiplied by the number of `123` after it, which may be odd.

The complexity is $O(256 \cdot 4 \cdot |s|)$ per test case, with $O(256)$ memory.

G Grouped Vertices

For a vertex c let $B(c, r)$ be the set of vertices at distance at most r from c, and for an edge uv let $B(uv, r) = B(u, r) \cup B(v, r)$. Both sets are connected (each vertex is joined to u or v by a path inside the set), and their diameters are at most $2r$ and $2r + 1$ respectively.

Key observation. The answer for x is the size of the largest ball of the matching parity:

$$\max_c |B(c, x/2)| \text{ for even } x, \qquad \max_{uv} |B(uv, (x-1)/2)| \text{ for odd } x$$

(and 1 for odd x if $n = 1$).

By the bounds above, these balls are valid sets. Conversely, let S be a connected set with diameter $D \le x$. Since S is connected, distances inside S equal distances in the tree. Take a path $a \dots b$ of length D in S and its middle: a vertex c if D is even, an edge uv if D is odd. Every $w \in S$ is within $\lfloor D/2 \rfloor$ of the middle. Indeed, let p be the vertex of the path $a \dots b$ closest to w, say on the half towards a; then the route from w to b passes through p and then through the middle, so if w were farther than $\lfloor D/2 \rfloor$ from the middle, we would get $d(w, b) > D$. Now fit S into a ball of the right parity:

- x even, D even: $S \subseteq B(c, D/2) \subseteq B(c, x/2)$;
- x even, D odd: $S \subseteq B(uv, (D-1)/2) \subseteq B(u, (D+1)/2) \subseteq B(u, x/2)$;
- x odd, D odd: $S \subseteq B(uv, (D-1)/2) \subseteq B(uv, (x-1)/2)$;
- x odd, D even: $S \subseteq B(c, D/2) \subseteq B(cv, (x-1)/2)$ for any neighbour v of c (it exists unless $n = 1$).

Algorithm. Run a BFS from every vertex c and count the vertices at each distance; the prefix sums of these counts are $|B(c, r)|$ for all r at once (for r beyond the largest distance the ball is the whole tree). Keep the maximum for every r. Do the same with a two-source BFS from both ends of every edge. This is $2n - 1$ BFS runs of $O(n)$ each, so the complexity is $O(n^2)$ time and $O(n)$ memory per test case.

H Hall of Songs

By linearity of expectation, the answer is the sum over the n initial songs of the probability that the song is still known after k evenings. All songs are symmetric, so the answer is $n \cdot s$, where s is the probability that one fixed song survives.

Follow that song and let c be the number of musicians who know it; initially $c = 1$. An evening changes c only if exactly one of a and b knows the song. If b knows it and a does not, c grows by one; if a knows it and b does not, c drops by one. Each of the two events has probability

$$r_c = \frac{c\,(n-c)}{n\,(n-1)},$$

and with probability $1 - 2r_c$ nothing changes. The next value of c depends only on the current one, so c is a Markov chain on the states $0, 1, \dots, n$, and the song is lost for good exactly when c reaches 0 (then $r_0 = 0$).

Let T be the $(n+1) \times (n+1)$ transition matrix of this chain: $T_{c,c-1} = T_{c,c+1} = r_c$ and $T_{c,c} = 1 - 2r_c$. The distribution of c after k evenings is the row 1 of T^k, hence

$$\text{answer} = n \cdot \bigl(1 - (T^k)_{1,0}\bigr).$$

All computations can be done directly modulo $M = 998\,244\,353$: the probabilities r_c become $c(n-c) \cdot (n(n-1))^{-1} \bmod M$, and the matrix power uses only additions and multiplications, so the result is the required residue $p \cdot q^{-1} \bmod M$. Compute T^k by binary exponentiation: at most $2\log_2 k \approx 120$ multiplications of matrices of size at most 101×101. Note that k needs a 64-bit type. It must not be reduced modulo $M - 1$: T is singular, so Fermat's little theorem does not apply to it.

The complexity is $O(n^3 \log k)$.

Remark. Looking backwards in time, the ancestral lines of the n musicians merge like in Kingman's coalescent, which leads to a closed form computable in $O(n \log k)$; it is not needed here.

I Inconvenient Swaps

Two elements can be swapped exactly when their values have the same remainder modulo 3, where the remainder is taken in $\{0, 1, 2\}$. Split the positions into three classes by $a_i \bmod 3$. The values never change, and a swap involves two positions of one class, so every class keeps its own multiset of values on its own positions.

Conversely, any two elements of one class may be swapped directly, and every permutation is a product of transpositions. Hence the values of a class can be arranged on the positions of that class in any order, independently of the other classes. The reachable arrays are exactly the arrays in which every class holds its own values in some order.

Let b be the array in which every class receives its values sorted in increasing order, placed on its positions from left to right. It is the lexicographically smallest reachable array: take any other reachable array c and the first position i where $b_i \neq c_i$, and let r be the class of i. The positions of class r to the left of i hold the same values in b and in c, so c_i is one of the remaining values of class r, and b_i is the smallest of them. Thus $b_i < c_i$.

So we collect the positions and the values of every class, sort the values and write them back in order. The only trap is the remainder of a negative number: in C++ and Java $-1 \,\%\, 3 = -1$, although $2 - (-1) = 3$ is divisible by 3, so use $((a_i \,\%\, 3) + 3) \,\%\, 3$.

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

J Jar of Colours

A stroke only copies colours to the right. So in every obtainable colouring b each plank i has a source $p(i) \le i$ with $b_i = c_{p(i)}$ (initially $p(i) = i$). A stroke (l, r) sets $p(l), \dots, p(r)$ to $p(l)$, which keeps p non-decreasing: $p(j) \le p(l)$ for $j < l$ and $p(j) \ge p(r) \ge p(l)$ for $j > r$.

Claim. b is obtainable iff there is a non-decreasing p with $p(i) \le i$ and $b_i = c_{p(i)}$ for all i.

Necessity is shown above. Conversely, each used source k serves a segment $[s_k, e_k]$ with $k \le s_k$. Make the strokes (k, e_k) for the used sources from right to left. Plank k is still untouched at that moment, since the earlier strokes start to the right of k. The stroke also paints $[k, s_k - 1]$, but these planks belong to smaller sources and are repainted later, while all later strokes end before s_k.

Counting each colouring once. Split b into maximal runs of equal colour; run j starts at plank s_j and has colour v_j. Replacing all sources inside a run by the smallest of them keeps p valid, so run j may use a single source k_j, where $1 = k_1 < k_2 < \dots$ (neighbouring runs have different colours) and $k_j \le s_j$. Take the sources greedily: k_j is the first plank after k_{j-1} with colour v_j. By induction, the greedy k_j is never to the right of the j-th source of any valid p, so b is obtainable iff its greedy chain exists and $k_j \le s_j$ for all j.

Let prv_k be the previous plank with colour c_k (0 if there is none). A greedy chain can go from k to k' iff $\mathrm{prv}_{k'} < k < k'$; this also rules out $c_{k'} = c_k$. Hence the obtainable colourings are in bijection with pairs (greedy chain, run starts $1 = s_1 < s_2 < \dots$) with $k_j \le s_j$.

Let $f_i[k]$ be the number of such prefixes of length i whose last run has source k. Then $f_1[1] = 1$ and, for $k' \le i + 1$,

$$f_{i+1}[k'] = f_i[k'] + \sum_{k=\mathrm{prv}_{k'}+1}^{k'-1} f_i[k].$$

The first term extends the last run; the sum starts a new run at plank $i+1$, which is allowed exactly because $k' \le i + 1$. The answer is $\sum_k f_n[k]$. (For pairwise distinct colours it is the Catalan number C_n.)

With prefix sums of f_i, each step takes $O(n)$; prv is found by sorting the pairs (c_i, i). The complexity is $O(n^2)$ time and $O(n)$ memory per test case.

K Know This Problem?

Call the non-decreasing subsequence U and the non-increasing one D, and process the elements from left to right. Whether the rest of the array can still be placed depends only on the last element of U and the last element of D. A new element x can join U if it is at least the last element of U, and join D if it is at most the last element of D, so a smaller last element of U and a larger last element of D are never worse. An empty U acts as $-\infty$, and an empty D as $+\infty$.

Two numbers are enough. After processing h_i, one of the two last elements is h_i itself. So we keep only

- u: the largest possible last element of D over all valid splits of $h_1, \dots, h_i$ with $h_i \in U$ ($+\infty$ if D is empty, "impossible" if there is no such split);
- d: the smallest possible last element of U over all valid splits with $h_i \in D$ ($-\infty$ if U is empty, "impossible" if there is none).

Initially $u = +\infty$ and $d = -\infty$. Let $p = h_i$ and $x = h_{i+1}$.

- From $h_i \in U$ (last of U is p, last of D is u): if $p \le x$, then x may join U, a candidate u for the new u; if $u \ge x$, then x may join D, a candidate p for the new d.
- From $h_i \in D$ (last of D is p, last of U is d): if $p \ge x$, a candidate d for the new d; if $d \le x$, a candidate p for the new u.

The new u is the maximum of its candidates and the new d is the minimum of its candidates ("impossible" if there are none). The answer is `YES` if and only if u or d is possible at the end. This is $O(n)$ time, and the array does not even have to be stored.

All comparisons are non-strict, because equal values are allowed in both subsequences: a solution for strictly monotone subsequences fails. Keep "impossible" separate from "empty" and from real values: 10^9 is a valid value, so it cannot serve as $+\infty$.

Remark: finding a split. Remember for every i which transition gave u and d, and walk back from $i = n$. An equivalent greedy: if h_i fits only one subsequence, put it there; if it fits both, look at the next value $w \neq h_i$ and put h_i into D if $w < h_i$, and into U otherwise. Looking only at h_{i+1} is not enough, because a long run of equal values may come before w.

L Lost List

A skier with limits (x, y) finds the slope type (i, j) suitable exactly when $x \geq i$ and $y \geq j$, so $a_{i,j}$ is the sum of $c_{x,y}$ over all $x \geq i$, $y \geq j$: a is the two-dimensional suffix sum of c. It consists of two one-dimensional suffix sums, one along the steepness levels and one along the length levels, and each of them is inverted uniquely: if $s_j = v_j + v_{j+1} + ... + v_m$, then $v_j = s_j - s_{j+1}$ (with $s_{m+1} = 0$). Hence the integers c with the required sums always exist and are unique:

$$c_{x,y} = a_{x,y} - a_{x+1,y} - a_{x,y+1} + a_{x+1,y+1},$$

where a is 0 outside the table (for $i = n + 1$ or $j = m + 1$). The survey is consistent if and only if all these $c_{x,y}$ are non-negative; otherwise the answer is -1. Check every pair (x, y), including $x = n$ and $y = m$, and do not reuse values from a previous test case as the zero border.

Every $c_{x,y}$ lies between $-2 \cdot 10^9$ and $2 \cdot 10^9$, so a 32-bit `int` suffices.

The complexity is $O(nm)$ per test case.

Mosaic Arch

Recolouring. Let $x_{i,j}$ be the colour of (i, j) if $i + j$ is even, and the opposite colour otherwise. For two side-adjacent cells the parity of $i + j$ differs, so their colours differ if and only if their values of x are equal. Consecutive cells of a Π-shape are side-adjacent, hence a Π-shape is good if and only if all its cells have the same x.

Let $d_{i,j}$ be the number of cells in the longest vertical segment of equal x that starts at (i, j) and goes down (computed from the bottom row up). Fix r, c_1, c_2. The cells $(r, c_1), \dots, (r, c_2)$ must lie inside one run of row r, a maximal segment of equal x; then the Π-shape with these r, c_1, c_2 and a given k is good if and only if $k \le d_{r,c_1}$ and $k \le d_{r,c_2}$. All $a_{i,j}$ are positive, so for fixed r, c_1, c_2 the best choice is $k = \min(d_{r,c_1}, d_{r,c_2})$, provided it is at least 2.

Grouping the pairs by k. A wide grid has too many pairs (c_1, c_2), so we group them by k. Let $P_j = a_{r,1} + \dots + a_{r,j}$ and $L_k(c) = a_{r+1,c} + \dots + a_{r+k-1,c}$ (the latter is $O(1)$ with prefix sums over columns). The Π-shape (r, c_1, c_2, k) has the weight

$$\big(L_k(c_1) - P_{c_1-1}\big) + \big(P_{c_2} + L_k(c_2)\big).$$

Fix a run and go through the distinct values $k \ge 2$ of $d_{r,c}$ in this run. For each k, call the columns of the run with $d_{r,c} \ge k$ active and scan them from left to right: at an active c_2, add its right bracket to the maximum of the left bracket over the active $c_1 \le c_2 - 2$, which is a prefix maximum maintained with a second pointer. Every value obtained is the weight of a good Π-shape (as $k \le \min(d_{r,c_1}, d_{r,c_2})$), and every pair of columns receives its best value in the scan with $k = \min(d_{r,c_1}, d_{r,c_2})$. So the answer is the maximum over all scans, or -1 if no scan finds a pair.

Complexity. A run of width w has at most $\min(w, n)$ distinct values of d, and each scan takes $O(w)$ time (keeping the active columns in a linked list and deleting those with $d_{r,c} = k$ after the scan makes it even faster). The runs of a row have total width at most m, so a row costs $O(m \cdot \min(n, m))$, and the complexity is $O(nm \cdot \min(n, m)) \subseteq O((nm)^{3/2})$ per test case, plus sorting the values of d within each run. The answer can exceed 10^{14}, so 64-bit integers are needed.