

## Problem A. Adjacent Decrements

**Problem.** An operation on an array picks an adjacent pair: if the two values are equal, both are decreased by 1; otherwise the smaller one is decreased by 1. Values must stay non-negative, so an operation is allowed only if it does not create a negative entry. An array is *good* if some sequence of operations turns it into all zeroes. Given  $n$  and  $k$  ( $1 \leq n, k \leq 250\,000$ ), count the good arrays of length  $n$  with entries in  $\{0, 1, \dots, k\}$ , modulo 998 244 353.

**Solution.** Call a maximal constant run  $a_l = a_{l+1} = \dots = a_r = v > 0$  a *summit* if both its neighbours are strictly smaller than  $v$  (an array border also counts as smaller). The key structural fact is:

*An array is good if and only if every summit has even length.*

*Necessity.* We show that any single operation applied to an array containing an odd summit leaves an array that still contains an odd summit; since the all-zero array has no summits, such an array can never be destroyed. Fix an odd summit  $[l, r]$  of value  $v$ , length  $\ell$ . Elements of the summit can never be decreased by a pair crossing its border, because the outside neighbour is strictly smaller (and outside values only decrease, so this stays true). An operation touching only the neighbours keeps them below  $v$ , so the summit survives unchanged. The only interesting operation is the equal-pair move at some  $(i, i+1)$  inside the run: it splits the run into pieces  $[l, i-1]$  and  $[i+2, r]$  still at value  $v$ , of total length  $\ell-2$ , which is odd, so one of the two pieces has odd (hence positive) length; its outer neighbour is still  $< v$  and its inner neighbour now equals  $v-1 < v$ , so it is again an odd summit.

*Sufficiency.* By induction on the sum it is enough to find, in any nonzero array all of whose summits are even, one operation preserving this invariant. Let  $v$  be the maximum value; every run of  $v$ 's is a summit, hence even. Pick one such run  $[l, r]$  and apply the equal-pair move at  $(l, l+1)$ . If  $\ell \geq 4$ , this is always safe:  $[l+2, r]$  stays an even summit, and the new pair of  $(v-1)$ 's (possibly merged with a run of  $(v-1)$ 's on the left) has right neighbour  $v$ , so it is not a summit. If  $\ell = 2$  the two new  $(v-1)$ 's may merge with runs of  $(v-1)$ 's of lengths  $p$  (left) and  $q$  (right) into one run of length  $p+q+2$ ; the move is unsafe only when this run is a summit of odd length, i.e.  $p+q$  is odd (this forces  $v \geq 2$ : for  $v = 1$  the new values are zeroes and zeroes are never summits). Say  $p$  is odd, so  $p \geq 1$  and  $a_{l-1} = v-1 \geq 1$ . Then instead decrease  $a_{l-1}$  using its larger neighbour  $a_l = v$ : the left run of  $(v-1)$ 's loses its last element and becomes an even summit of length  $p-1$  (its new right neighbour is  $v-2$ ), the new value  $v-2$  at position  $l-1$  has right neighbour  $v$ , so no odd summit appears. The case  $q$  odd is symmetric.

*Counting.* Let  $F_k(x) = \sum_{m \geq 0} f_m x^m$  where  $f_m$  counts good arrays of length  $m$  over  $\{0, \dots, k\}$ , and let  $G_k(x)$  count good *zero-free* arrays over  $\{1, \dots, k\}$  (both include the empty array). Two identities follow from the characterization. First, writing an array as blocks of positive values separated by single zeroes, and noting that summits never involve zero values, goodness is exactly “every block is good”, hence  $F_k = G_k \cdot \frac{1}{1-xG_k} = \frac{G_k}{1-xG_k}$ . Second, subtracting 1 from every entry is a bijection from zero-free arrays over  $\{1, \dots, k\}$  to arrays over  $\{0, \dots, k-1\}$  that preserves all runs and comparisons; the only discrepancy is at value 1: a run of 1's can be a summit of  $b$  only if  $b = (1, \dots, 1)$  entirely, and it maps to the (always good) zero array. So  $b$  is good iff  $b-1$  is good, except that  $b = 1^\ell$  with  $\ell$  odd is bad. Therefore  $G_k = F_{k-1} - \frac{x}{1-x^2}$ .

Combining the two identities gives the step  $G_k = \frac{G_{k-1}-x}{(1-x^2)(1-xG_{k-1})}$  with  $G_0 = 1$  — a fractional linear map with fixed polynomial coefficients. Represent  $G_k = S_k/D_k$  as a ratio of formal power series; then the step becomes linear:  $S_{k+1} = S_k - xD_k$ ,  $D_{k+1} = (1-x^2)(D_k - xS_k)$ , starting from  $S_0 = D_0 = 1$ . In matrix form  $\begin{pmatrix} S_{k+1} \\ D_{k+1} \end{pmatrix} = M \begin{pmatrix} S_k \\ D_k \end{pmatrix}$  where  $M = \begin{pmatrix} 1 & -x \\ -x+x^3 & 1-x-x^2+x^3 \end{pmatrix}$ , so  $\begin{pmatrix} S_k \\ D_k \end{pmatrix} = M^k \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  can be computed by binary exponentiation of a  $2 \times 2$  matrix of polynomials, truncating everything modulo  $x^{n+1}$  and multiplying with NTT. Finally  $F_k = \frac{S_k}{D_k - xS_k}$ ; since  $D_k$  has constant term 1, the denominator is invertible as a power series, and the answer is  $[x^n]F_k$ , obtained with one Newton inversion and one multiplication. As a sanity check,  $F_1 = \frac{1}{1-x-x^2}$  counts binary strings whose runs of 1's all have even length (Fibonacci numbers), and  $[x^5]F_2 = 36$ ,  $[x^7]F_3 = 855$  match the samples.

**Complexity:**  $O(n \log n \log k)$  time and  $O(n)$  memory.

## Problem B. Binary Rotations

**Problem.** All full binary bracketings with  $\ell$  leaves ( $1 \leq \ell \leq 14$ ) are listed in lexicographic order as  $S_1, \dots, S_n$ , where  $n = C_{\ell-1} \leq C_{13} = 742\,900$ . The allowed operation is the right rotation  $((AB)C) \rightarrow (A(BC))$  applied to any subexpression. Given a permutation  $p$  of  $1..n$ , count the pairs  $i < j$  such that  $S_{p_i}$  is reachable from  $S_{p_j}$  by zero or more rotations. Zero or more right rotations define a partial order on bracketings, so we must count the pairs  $(i, j)$ ,  $i < j$ , with  $S_{p_j} \leq S_{p_i}$  in this order.

**Solution.** Number the leaves  $1, \dots, \ell$  from left to right and call the position between leaves  $g$  and  $g+1$  *gap*  $g$  ( $1 \leq g \leq \ell-1$ ). Every internal node splits its leaf interval between its two children; this gives a bijection between the  $\ell-1$  internal nodes and the  $\ell-1$  gaps (the node assigned to gap  $g$  is the LCA of leaves  $g$  and  $g+1$ ). Define the *bracketing vector*  $r(T)$  by:  $r_g$  — the index of the rightmost leaf of the internal node whose split is at gap  $g$ . Always  $g+1 \leq r_g \leq \ell$  and  $r_{\ell-1} = \ell$ , so only  $\ell-2$  coordinates carry information. The vector determines the tree: the root's gap is the smallest  $g$  with  $r_g = \ell$ , and the two halves are reconstructed recursively.

*Lemma 1.* One rotation changes exactly one coordinate, and increases it. Indeed, rotating  $((AB)C) \rightarrow (A(BC))$  affects only two nodes; the node at the gap  $g$  between  $A$  and  $B$  changes its right end from the last leaf of  $B$  to the last leaf of  $C$ , while the node at the gap between  $B$  and  $C$  keeps the last leaf of  $C$  as its right end. Hence reachability implies  $r(T) \leq r(T')$  componentwise.

*Lemma 2.* Conversely, if  $r(T) \leq r(T')$  componentwise then  $T'$  is reachable from  $T$  — so rotation-reachability coincides with componentwise dominance of the vectors. Sketch: let  $g$  be the smallest gap with  $r_g(T) < r_g(T')$  and let  $u$  be the node of  $T$  at gap  $g$ , with right end  $m = r_g(T)$ . First,  $u$  is a left child: if  $u$  were the right child of a node  $q$ , then  $q$  has the same right end  $m$  and a smaller gap  $h < g$ , so  $r_h(T) = m$ , and by minimality  $r_h(T') = m$ ; but in  $T'$  the node at gap  $g$  contains leaves  $h+1, \dots$  hence contains the node at gap  $h$  in its span, forcing  $r_g(T') \leq m$  — contradiction. So the rotation at  $u$ 's parent applies; it raises  $r_g$  from  $m$  to  $r_m(T)$  (the parent's right end — the parent is exactly the node at gap  $m$ ). Finally  $r_m(T) \leq r_m(T') \leq r_g(T')$ : the second inequality holds because in  $T'$  the node at gap  $g$  spans past leaf  $m+1$  (as  $r_g(T') > m$ ), hence contains the node at gap  $m$ . So the rotation keeps  $r(T) \leq r(T')$  while strictly increasing  $\sum_g r_g(T)$ ; induction finishes the proof.

Thus the answer is the number of pairs  $i < j$  with  $r(S_{p_i}) \geq r(S_{p_j})$  componentwise — a dominance-counting problem in  $d = \ell-2 \leq 12$  dimensions over  $n \leq 742\,900$  points. Classical  $O(n \log^{d-1} n)$  tools are hopeless here; instead we meet in the middle over the *coordinates*. Split the  $d$  coordinates into two halves  $H_1, H_2$  of size at most 6 and project every vector onto each half. The projections are heavily repeated: for  $\ell = 14$  with the split  $H_1 = \{1, \dots, 6\}$ ,  $H_2 = \{0, 7, \dots, 11\}$  (0-indexed; chosen to balance the sides) there are only  $P = 9996$  distinct  $H_1$ -classes and  $R = 1221$  distinct  $H_2$ -classes. Precompute, by brute force in  $O((P^2 + R^2)d)$  byte comparisons, for every  $H_1$ -class  $a$  the list  $\text{lower}(a)$  of  $H_1$ -classes dominated by  $a$ , and for every  $H_2$ -class  $b$  the list  $\text{upper}(b)$  of  $H_2$ -classes dominating  $b$ .

Now sweep the permutation left to right, maintaining a table  $\text{cnt}[b][a]$  indexed by ( $H_2$ -class,  $H_1$ -class). For the current item  $y$  first *query*: add  $\sum_{s \in \text{upper}(b_y)} \text{cnt}[s][a_y]$  to the answer; then *insert*: increment  $\text{cnt}[b_x][q]$  for all  $q \in \text{lower}(a_x)$  where  $x = y$ . An earlier item  $x$  is counted at  $y$  iff  $b_x \geq b_y$  (row  $b_x$  appears in  $\text{upper}(b_y)$ ) and  $a_x \geq a_y$  (column  $a_y$  was incremented, being in  $\text{lower}(a_x)$ ), i.e. exactly when  $r(x) \geq r(y)$ . For  $\ell = 14$  this is about  $2.2 \cdot 10^8$  increments and  $1.1 \cdot 10^8$  additions on a flat array — about 1.5 seconds.

**Implementation notes.** Generate all trees recursively together with their vectors (for a root splitting  $\ell$  leaves as  $k + (\ell - k)$ : copy the left child's vector, put  $\ell$  at gap  $k$ , append the right child's vector shifted by  $k$ ), pair each with its string, and sort the strings; note that in a valid expression the first mismatching characters can only be  $($  versus  $x$  (after any common prefix the next character is either forced to be  $)$  or is the start of a subexpression), so plain ASCII comparison already realizes the required order  $( < x < )$ . Store  $r_g - (g+1) \in [0, 12]$ , so each 6-coordinate projection packs into 4 bits per coordinate; dominance is non-strict coordinatewise (equal coordinates are allowed).

**Complexity:**  $O(n\ell + (P^2 + R^2)\ell + n(\bar{\lambda} + \bar{\mu}))$  time, where  $P, R$  are the numbers of distinct half-projections and  $\bar{\lambda}, \bar{\mu}$  the average sizes of the dominated/dominating class lists (about 450 combined per item for  $\ell = 14$ ), and  $O(P \cdot R)$  memory — roughly  $1.2 \cdot 10^7$  counters ( $\approx 50$  MB).

## Problem C. Circle Construction

Output

$$P_1 = (-r, 0), P_2 = (0, -r),$$

$$P_3 = (r, 0), P_4 = (0, r),$$

$$P_5 = (r + 1, r + 2), P_6 = (r + 2, r + 1).$$

The first four points lie on  $x^2 + y^2 = r^2$ .

Consider two points  $(a, b)$  and  $(b, a)$ . Every circle through both of them has its center on the line  $x = y$ . Consequently, for any two such pairs with different values of  $a + b$ , there is a circle passing through all four points: choose the appropriate center on  $x = y$  so that its distances to the two pairs are equal.

Both  $(P_3, P_4)$  and  $(P_5, P_6)$  are swapped-coordinate pairs, and their coordinate sums are  $r$  and  $2r + 3$ . Therefore, they lie on one circle. Similarly,  $(P_1, P_2)$  and  $(P_5, P_6)$  have coordinate sums  $-r$  and  $2r + 3$ , so they also lie on one circle.

The point  $P_5$  does not lie on the first circle, so each of the two auxiliary circles is different from the first one. If the two auxiliary circles were equal, that circle would contain  $P_1, P_2, P_3$ , and  $P_4$ , and hence it would be the first circle. This is a contradiction, so all three circles are pairwise distinct. The six points are pairwise distinct as well, and their coordinates satisfy the required bounds.

The construction takes  $O(1)$  time per test case.

## Problem D. Dynamic Adjustment

**Problem.** For an integer array  $a_1, a_2, \dots, a_m$  let  $f(a)$  be the minimum of  $\sum_{i=1}^m |a_i - b_i|$  over all integer arrays  $b$  with  $b_1 \leq b_2 \leq \dots \leq b_m$ . You are given  $x_1, x_2, \dots, x_n$  ( $1 \leq n \leq 200\,000$ ,  $|x_i| \leq 10^9$ ) and a permutation  $p_1, p_2, \dots, p_n$ . For every  $i$  from 1 to  $n$  you must output  $f(a)$ , where  $a$  consists of the elements of  $x$  at positions  $p_1, \dots, p_i$ , taken in increasing order of position. So positions are *activated* one by one, and after each activation we must know the cost of making the active subsequence non-decreasing.

**Solution.** *Step 1: a combinatorial dual of  $f$ .* Claim:  $f(a_1, \dots, a_m)$  equals the maximum, over all families of pairwise disjoint index pairs  $(i, j)$  with  $i < j$ , of  $\sum (a_i - a_j)$ . One direction is elementary: if  $b$  is non-decreasing and  $i < j$ , then  $|a_i - b_i| + |a_j - b_j| \geq (a_i - b_i) + (b_j - a_j) \geq a_i - a_j$  because  $b_i \leq b_j$ ; summing over disjoint pairs bounds the cost of every feasible  $b$  from below by the weight of every family. For the other direction write the LP dual: with multipliers  $\lambda_i \geq 0$  for the constraints  $b_i \leq b_{i+1}$ , minimizing  $|a_i - b_i| + (\lambda_i - \lambda_{i-1})b_i$  over  $b_i$  is finite iff  $c_i = \lambda_i - \lambda_{i-1} \in [-1, 1]$ , giving the dual  $\max \sum c_i a_i$  subject to  $c_i \in [-1, 1]$ , all prefix sums of  $c$  non-negative and the total sum zero. The constraint matrix is a network matrix, so an integral optimum  $c_i \in \{-1, 0, +1\}$  exists, and by Hall's theorem such labelings are exactly the pairings of each  $-1$  with a distinct earlier  $+1$ ; a pair  $(+1 \text{ at } i, -1 \text{ at } j)$  contributes  $a_i - a_j$ . (This is the same duality that makes the classic slope-trick for static isotonic  $L_1$  regression —“push  $a_i$  twice, pop the maximum once” — correct.)

*Step 2: reduction to unit-job deadline scheduling.* Map position  $pos$  to the deadline  $d = n + 1 - pos$ , so that suffixes of positions become prefixes of deadlines. Consider  $n$  unit time slots and unit jobs, where a job with deadline  $d$  may occupy any slot  $\leq d$ ; a set of jobs is feasible iff for every  $d$  the number of chosen jobs with deadline  $\leq d$  is at most  $d$ . Feasible sets are the independent sets of a transversal matroid. After step  $t$  the ground set is: two copies of a job with deadline  $n + 1 - pos$  and profit  $x_{pos}$  for every active position  $pos$ , and one *placeholder* job with deadline  $n + 1 - pos$  and profit  $H = 2 \cdot 10^9$  for every inactive  $pos$ . Picking one job per deadline shows the rank is exactly  $n$ . Since  $H$  exceeds every  $|x_i|$ , every maximum-weight basis contains all  $n - t$  placeholders (otherwise a basis exchange inserting a missing placeholder would gain at least  $H - 10^9 > 0$ ). Hence a maximum-weight basis selects exactly  $t$  real copies, giving each active task a

multiplicity  $c \in \{0, 1, 2\}$  with  $\sum c = t$ . Cancelling the placeholders in Hall's inequalities leaves, for every position  $p$ :  $\sum_{\text{active } q \geq p} c_q \leq \#\{\text{active } q \geq p\}$ , i.e. the doubled tasks ( $c = 2$ , label  $+1$ ) and the dropped tasks ( $c = 0$ , label  $-1$ ) satisfy exactly the labeling condition of Step 1 restricted to the active subsequence. The basis weight is  $(n - t)H + \sum_{\text{active}} x + \sum (c_i - 1)x_i$ , so if  $W_t$  denotes the maximum basis weight, the answer after step  $t$  is  $W_t - (n - t)H - \sum_{\text{active}} x$ . This explains the two oddities of the construction: each task is inserted *twice* because multiplicity 2 encodes the label  $+1$ , and the placeholders keep the slot structure static, so activating position  $pos$  is just “delete the placeholder with deadline  $n + 1 - pos$ , insert the new job twice”.

*Step 3: maintaining the maximum-weight basis.* Keep the current basis  $S$  and the pool of unselected jobs, and define  $slack(d) = d - \#\{\text{selected jobs with deadline} \leq d\} \geq 0$ ; a deadline is *saturated* if its slack is 0. Two standard matroid exchange lemmas drive the updates (for distinct weights they follow by comparing the greedy runs before and after the change; ties are harmless since any extremal candidate gives the same weight).

*Insertion* of a job  $e$  with deadline  $d_e$  and profit  $w$ : if no saturated deadline  $d \geq d_e$  exists,  $S + e$  is independent and becomes the new basis. Otherwise let  $d^*$  be the *first* saturated deadline  $\geq d_e$ . The circuit of  $e$  consists precisely of the selected jobs with deadline  $\leq d^*$ : evicting any of them repairs every saturated constraint  $d \geq d^*$ , there are no saturated deadlines in  $[d_e, d^*)$ , and evicting a job with deadline  $> d^*$  leaves  $d^*$  violated. So find the minimum-profit selected job with deadline  $\leq d^*$ ; if  $w$  is larger, swap it out into the pool and select  $e$ , otherwise put  $e$  into the pool.

*Deletion* of a selected job  $x$  (applied only to placeholders, which are always selected): remove  $x$ , then let  $L$  be the *last* saturated deadline of  $S - x$  (0 if none). A pool job  $u$  satisfies  $S - x + u$  independent iff its deadline exceeds  $L$ . The new basis is  $S - x$  plus the maximum-profit such  $u$ , if one exists; profit ties may be broken by the largest deadline (any addable extremal choice works).

*Step 4: data structures.* A segment tree over deadlines  $1..n$  stores *slack* with subtree minima and lazy addition: (un)selecting a job with deadline  $d$  adds  $\mp 1$  on the suffix  $[d, n]$ ; the first saturated deadline  $\geq q$  and the last saturated deadline are found by descending into the leftmost (rightmost) child whose minimum is 0, all in  $O(\log n)$ . Profits (all  $x_i$  together with  $H$ ) are compressed to ranks  $1..K$ ; for each rank an ordered multiset keeps the deadlines of its selected and unselected jobs. Two segment trees over ranks store per rank the *minimum selected* deadline (or  $\infty$ ) and the *maximum unselected* deadline (or 0). Descending them answers “smallest rank owning a selected job with deadline  $\leq d^*$ ” (the eviction candidate) and “largest rank owning an unselected job with deadline  $> L$ ” (the re-selection candidate) in  $O(\log n)$ ; the multiset then produces the concrete deadline (its minimum, respectively maximum, which is exactly the value that certified the query). A running sum of selected profits is maintained.

*Step 5: the main loop.* Initially the  $n$  placeholders (deadlines  $1..n$ ) are all selected and every slack is 0. At step  $t$  with  $pos = p_t$  and  $d = n + 1 - pos$ : delete the placeholder with deadline  $d$ , insert the job  $(d, x_{pos})$  twice, add  $x_{pos}$  to `activeSum`, and print `selectedSum - (n - t) · H - activeSum`. Since the three updates all use the same deadline  $d$ , the rank returns to  $n$  after each step and exactly  $t$  real copies stay selected, as required by Step 2. Answers fit in 64-bit integers ( $|f| \leq n \cdot 10^9$ ).

**Complexity:**  $O(n \log n)$  time and  $O(n)$  memory.

## Problem E. Equal Totients

**Problem.** Given  $t \leq 20$  queries, each with two integers  $l, r$  ( $1 \leq l \leq r \leq 10^{17}$ , and it is guaranteed that  $r \geq \frac{5l}{3}$ ), find two distinct integers  $x, y \in [l, r]$  with  $\phi(x) = \phi(y)$ , where  $\phi$  is Euler's totient function, or report  $-1 -1$  if no such pair exists.

**Solution.** The numbers are far too large to factor everything in the range, so we should look for a *constructive* pair of equal totients whose ratio is small enough to fit inside  $[l, r]$ .

*Key observation.*  $\phi$  is multiplicative:  $\phi(ab) = \phi(a)\phi(b)$  whenever  $\gcd(a, b) = 1$ . Take any  $k$  with  $\gcd(k, 6) = 1$  and consider the pair  $x = 3k, y = 4k$ . Since  $3 \nmid k$  and  $2 \nmid k$ , we get  $\phi(3k) = \phi(3)\phi(k) = 2\phi(k)$  and  $\phi(4k) = \phi(4)\phi(k) = 2\phi(k)$ , so  $\phi(x) = \phi(y)$ , and  $y/x = 4/3$ . (The full coprimality condition is essential: for  $k = 2$  we get  $\phi(6) = 2 \neq 4 = \phi(8)$ , and for  $k = 3$  we get  $\phi(9) = 6 \neq 4 = \phi(12)$ .) Compare this with

the classic pair  $(n, 2n)$  for odd  $n$ : its ratio 2 is too large for our guarantee, while  $4/3 < 5/3$  leaves slack.

*Lemma (the pair fits).* Let  $A = \lceil l/3 \rceil$  and let  $k^*$  be the smallest  $k \geq A$  with  $\gcd(k, 6) = 1$ . Then  $k^* \leq A + 3$ , because the residues coprime to 6 are  $\{1, 5\} \pmod{6}$  and the largest gap in this set is 4 (from  $6m + 1$  to  $6m + 5$ ), so any window of 4 consecutive integers contains one. Consequently  $x = 3k^* \geq l$  and  $y = 4k^* \leq 4(\frac{l+2}{3} + 3) = \frac{4l+44}{3}$ , which is at most  $\frac{5l}{3} \leq r$  whenever  $l \geq 44$ . So for all  $l \geq 44$  the guarantee  $r \geq \frac{5l}{3}$  ensures the constructed pair lies in  $[l, r]$ . For small  $l$  the bound  $y \leq \frac{4l+44}{3} \leq 68$  holds, so the construction also succeeds whenever  $r \geq 68$ ; only genuinely tiny ranges remain.

*Algorithm.* If  $r \leq 100$ , brute force: compute  $\phi(x)$  for every  $x \in [l, r]$  (trial division, or even the naive  $\sum_{y \leq x} [\gcd(x, y) = 1]$ , since  $r \leq 100$ ), store the first number attaining each totient value in a map, and output the first collision, or  $-1 -1$  if none. Otherwise  $r > 100$ : scan  $k = \lceil l/3 \rceil, \lceil l/3 \rceil + 1, \dots$  until  $\gcd(k, 6) = 1$  (at most 4 steps) and output  $3k$  and  $4k$ . Equivalently, scan  $x = l, l+1, \dots$  for the first  $x \equiv 3$  or  $15 \pmod{18}$  (i.e.  $x = 3k$  with  $\gcd(k, 6) = 1$ ); the gaps in this progression are 6 and 12, so  $x \leq l + 11$  and the same bound  $y = \frac{4x}{3} \leq \frac{4l+44}{3}$  applies. By the lemma and the case analysis above ( $l \geq 44$  uses the guarantee;  $l \leq 43$  gives  $y \leq 68 < r$ ), the pair always fits, so in this branch the answer is never  $-1 -1$ .

*Remark.* In fact, under the guarantee  $r \geq \frac{5l}{3}$  the answer  $-1 -1$  is never printed at all: for  $r > 100$  this follows from the construction, and for  $r \leq 100$  one can check exhaustively that every valid  $(l, r)$  contains a pair (e.g.  $l = 1, r = 2$  already gives  $\phi(1) = \phi(2) = 1$ ). Still, handling the  $-1 -1$  branch costs nothing and keeps the brute force honest. A final note on judging rather than solving: since a contestant may output *any* valid pair of numbers up to  $10^{17}$ , the checker must compute  $\phi$  of arbitrary 17-digit numbers, which requires Miller–Rabin primality testing and Pollard’s rho factorization with 128-bit intermediate arithmetic — but the solution itself needs nothing beyond gcd.

**Complexity:**  $O(1)$  per test case for the constructive branch (at most 4 candidates for  $k$ ); the brute-force branch runs only for  $r \leq 100$  and costs  $O(r\sqrt{r})$  (or  $O(r^2 \log r)$  with the naive  $\phi$ ), a negligible constant. Total:  $O(t)$  time,  $O(1)$  memory.

## Problem F. Flipping Lamps

**Problem.** There are  $n$  lamps ( $1 \leq n \leq 10\,000$ ), all initially off. On day  $i$  ( $1 \leq i \leq m \leq 200\,000$ ) one must choose exactly  $a_i$  distinct lamps ( $0 \leq a_i \leq n$ ) and toggle each of them. A plan is the sequence of the  $m$  chosen sets; two plans differ if the chosen sets differ on some day. Count the plans after which exactly  $r$  lamps ( $0 \leq r \leq n$ ) are on, modulo 998 244 353.

**Solution.** Encode the state of the lamps as a vector  $y \in \mathbb{F}_2^n$ . Toggling a set  $S$  of size  $a$  means adding (XOR-ing) the indicator vector of  $S$ , so a plan is a walk on the hypercube in which the  $i$ -th step adds an arbitrary vector of Hamming weight  $a_i$ . Let  $f_a : \mathbb{F}_2^n \rightarrow \mathbb{Z}$  be the indicator of the sphere  $\{s : |s| = a\}$ . Then the number of plans ending in state  $y$  is exactly the XOR-convolution  $(f_{a_1} * f_{a_2} * \dots * f_{a_m})(y)$ , because a convolution term corresponds to an ordered choice of one weight- $a_i$  vector per day summing to  $y$ .

XOR-convolutions are diagonalized by the Walsh–Hadamard transform  $\widehat{f}(z) = \sum_s (-1)^{\langle z, s \rangle} f(s)$ : it maps convolution to pointwise product, and inversion is  $f(y) = 2^{-n} \sum_z (-1)^{\langle y, z \rangle} \widehat{f}(z)$ . The key point is that the transform of a sphere is a coefficient of an explicit product. For  $0 \leq x \leq n$  define

$$P_x(t) = (1+t)^{n-x} (1-t)^x, \quad w_d(x) = [t^d] P_x(t).$$

*Lemma 1.* For every  $z$  with  $|z| = j$  we have  $\widehat{f}_a(z) = w_a(j)$ . *Proof:*  $\sum_s (-1)^{\langle z, s \rangle} t^{|s|}$  factors over the coordinates: a coordinate inside the support of  $z$  contributes  $1 - t$  (including it in  $s$  costs a sign), and any other coordinate contributes  $1 + t$ ; the whole sum is therefore  $(1+t)^{n-j} (1-t)^j = P_j(t)$ , and extracting the coefficient of  $t^a$  keeps exactly the vectors  $s$  of weight  $a$ .

*Lemma 2.* For a fixed  $y$  with  $|y| = r$ ,  $\sum_{z: |z|=j} (-1)^{\langle y, z \rangle} = w_j(r)$ . This is the same computation as Lemma 1 with the roles of the two vectors exchanged.

By symmetry of the whole process under permutations of the lamps, the count  $N(y)$  of plans ending at  $y$  depends only on  $|y|$ , so the answer is  $\binom{n}{r} N(y_0)$  for any fixed  $y_0$  of weight  $r$ . Applying the inverse transform to the pointwise product and grouping the  $2^n$  frequencies  $z$  by their weight  $j$  (using both lemmas) gives

the closed form

$$\text{Answer} = \binom{n}{r} \cdot \frac{1}{2^n} \sum_{j=0}^n w_j(r) \prod_{d=0}^n w_d(j)^{c_d},$$

where  $c_d$  is the number of days with  $a_i = d$ . The necessary parity condition ( $r \equiv \sum a_i \pmod{2}$ ), since each toggle changes the number of lit lamps by  $\pm 1$  is automatically encoded in this sum: from  $P_{n-j}(t) = P_j(-t)$  we get  $w_d(n-j) = (-1)^d w_d(j)$ , so when the parity fails, the terms for  $j$  and  $n-j$  cancel.

To evaluate the formula we never expand the products. Differentiating  $P_x(t)$  gives

$$(1-t^2) P'_x(t) = ((n-2x) - nt) P_x(t),$$

and comparing the coefficients of  $t^d$  on both sides yields the three-term recurrence

$$(d+1) w_{d+1}(x) = (n-2x) w_d(x) - (n-d+1) w_{d-1}(x), \quad w_0(x) = 1, \quad w_1(x) = n-2x.$$

For each fixed  $j$  this produces the entire column  $w_0(j), w_1(j), \dots, w_n(j)$  in  $O(n)$  modular operations (division by  $d+1$  uses precomputed inverses of  $1, \dots, n+1$ ). The outer coefficients  $w_j(r)$ ,  $j = 0, \dots, n$ , come from one more run of the same recurrence with  $x = r$  (there the degree index varies while  $x$  is fixed).

Putting everything together: read the  $a_i$  only into the histogram  $c_0, \dots, c_n$ ; precompute factorials for  $\binom{n}{r}$  and the inverse table; compute the array  $w_j(r)$  by the recurrence; then for each  $j = 0, \dots, n$  run the recurrence in  $d$ , and whenever  $c_d > 0$  multiply the running product by  $w_d(j)^{c_d}$  via fast exponentiation. Accumulate  $w_j(r) \cdot \prod_d w_d(j)^{c_d}$ , and finally multiply by  $\binom{n}{r}$  and by the modular inverse of  $2^n$  (all arithmetic modulo the prime 998 244 353; compute  $2^n \bmod p$  by fast exponentiation first).

**Complexity:**  $O(n^2 + n \cdot t \log m + m)$  time, where  $t \leq \min(n+1, m)$  is the number of distinct values among the  $a_i$ ; since  $\sum_d \log c_d \leq t \log(m/t) + t$ , this is at most a few times  $10^8$  simple operations for the worst constraints. Memory is  $O(n+m)$  (in fact  $O(n)$  beyond reading the input).

## Problem G. Game

Let  $M = 2^{m+1} - 1$ . We claim that the Grundy value of a single pile of  $a$  stones equals  $a \bmod M$ ; then, by the Sprague-Grundy theorem, the first player wins if and only if  $(a_1 \bmod M) \oplus \dots \oplus (a_n \bmod M) \neq 0$ . For  $m = 1$  this is the classic game “subtract a power of two”, whose losing positions are the multiples of 3.

**Lemma 1.** Every  $k$  with  $1 \leq k \leq M-1$  is an allowed amount.

Indeed,  $k < 2^{m+1}$  has at most  $m+1$  binary digits, and the only number below  $2^{m+1}$  with  $m+1$  ones is  $M$  itself.

**Lemma 2.** No allowed amount is a multiple of  $M$ : every positive multiple of  $2^t - 1$  has at least  $t$  ones in binary.

Proof: induction on the multiple  $V$ . If  $V < 2^t$ , then  $V = 2^t - 1$ , which has exactly  $t$  ones. Otherwise let  $2^a$  ( $a \geq t$ ) be the highest set bit of  $V$  and consider  $V' = V - 2^a + 2^{a-t}$ . Then  $V' \equiv V \pmod{2^t - 1}$ ,  $0 < V' < V$ , and the popcount does not increase (adding  $2^{a-t}$  can only merge ones via carries). By induction  $V'$  has at least  $t$  ones, hence so does  $V$ .  $\square$

Now fix a pile of  $a$  stones and let  $r = a \bmod M$ . By Lemma 1, for every residue  $s \neq r$  modulo  $M$  there is an allowed  $k \leq a$  with  $a - k \equiv s$ : take  $k \equiv r - s \pmod{M}$  with  $1 \leq k \leq M-1$  (possible since  $k \leq M-1 \leq a$  whenever  $a \geq M$ ; and if  $a < M$ , all  $k \in [1, a]$  are allowed and the options are exactly  $a-1, \dots, 0$ ). By Lemma 2, no allowed move preserves the residue. So, by induction on  $a$ , the set of Grundy values of the options is exactly  $\{0, 1, \dots, M-1\} \setminus \{r\}$  (truncated to  $\{0, \dots, a-1\}$  when  $a < M$ ), whose mex is  $r$ . Hence  $G(a) = a \bmod M$ .

## Problem H. Hamiltonian LCM

**Problem.** Given an array  $a$  of  $n$  integers with  $1 \leq a_i \leq 7$ , rearrange its elements into a sequence  $b_1, \dots, b_n$  maximizing  $\text{lcm}(b_1, b_2) + \text{lcm}(b_2, b_3) + \dots + \text{lcm}(b_{n-1}, b_n)$ , and output this maximum value.

**Solution.** Only the counts  $c_1, \dots, c_7$  of the values matter. Encode an arrangement by the  $7 \times 7$  matrix  $y$  in which  $y_{ij}$  is the number of positions where value  $j$  immediately follows value  $i$  (diagonal entries are allowed and contribute  $\text{lcm}(i, i) = i$  per unit); the value of the arrangement equals  $\sum_{i,j} y_{ij} \text{lcm}(i, j)$ . If the sequence starts with  $s$  and ends with  $t$ , then for every value  $x$  the out-degree is  $\sum_j y_{xj} = c_x - [x = t]$  and the in-degree is  $\sum_i y_{ix} = c_x - [x = s]$ . Let  $P$  be the set of *present* values ( $c_x > 0$ ) and  $k = |P|$ . For  $n = 1$  the answer is 0; assume  $n \geq 2$ .

*Lemma 1.* A nonnegative integer matrix  $y$  satisfying the degree equations for some  $s, t \in P$  is realized by an arrangement if and only if the support of  $y$ , viewed as an undirected graph, connects all values of  $P$ . *Proof.* Necessity is clear: all elements of a walk lie in one connected piece. For sufficiency, add an auxiliary arc  $t \rightarrow s$ ; the multigraph becomes balanced (in-degree equals out-degree at every vertex), and every weakly connected balanced multigraph is strongly connected and admits an Euler circuit (Hierholzer's algorithm). Removing the auxiliary arc from the circuit leaves an Euler path from  $s$  to  $t$ , i.e. an arrangement with matrix exactly  $y$ . Note that for  $n \geq 2$  every present value has positive degree, so connectivity is over all of  $P$ .  $\square$

*Lemma 2 (tree decomposition).* The support of any feasible  $y$  contains a spanning tree  $T$  of  $P$ , and every edge  $\{i, j\}$  of  $T$  satisfies  $y_{ij} + y_{ji} \geq 1$ , so each tree edge can be given an orientation along which  $y$  has at least one unit. Call the pair of vectors  $(o, \iota)$ , where  $o_x$  and  $\iota_x$  are the out- and in-degrees of  $x$  in the oriented tree, the *profile* of the oriented tree, and  $w = \sum_{\{i,j\} \in T} \text{lcm}(i, j)$  its *forced weight* (independent of the orientation).  $\square$

Conversely, fix any spanning tree of  $P$  with an orientation, with profile  $(o, \iota)$  and forced weight  $w$ , and let  $z$  be *any* nonnegative integer matrix with row sums at most  $c_x - o_x$ , column sums at most  $c_x - \iota_x$ , and total sum exactly  $n - k$ . Then  $y = (\text{oriented tree}) + z$  is feasible: its row sums are at most  $c_x$  and add up to  $n - 1 = \sum_x c_x - 1$ , hence exactly one unit of row slack remains, at some present value — this value is the endpoint  $t$ ; symmetrically the unique unit of column slack determines  $s$ ; the degree equations for this pair  $(s, t)$  hold, and the support contains the tree, hence is connected, so Lemma 1 applies. The endpoints need not be enumerated: the linear program chooses them through the slack.

*Theorem.* Let  $w^*(o, \iota)$  be the maximum forced weight over all spanning trees of  $P$  with an orientation having profile  $(o, \iota)$ , and let  $T(o, \iota)$  be the optimum of the *transportation problem*: maximize  $\sum_{i,j} z_{ij} \text{lcm}(i, j)$  over nonnegative integer  $z$  with row sums at most  $c_x - o_x$ , column sums at most  $c_x - \iota_x$ , and total exactly  $n - k$  (profiles with some  $c_x - o_x < 0$  or  $c_x - \iota_x < 0$  are discarded). Then the answer equals  $\max_{(o, \iota)} (w^*(o, \iota) + T(o, \iota))$ . *Proof.* “ $\geq$ ”: by the previous paragraph every profile together with every feasible  $z$  yields a valid arrangement of that value. “ $\leq$ ”: take an optimal arrangement, its matrix  $y$  and endpoints  $(s, t)$ ; by Lemma 2 choose a spanning tree in the support and orientations carried by  $y$ , obtaining some profile  $(o, \iota)$  with forced weight  $w \leq w^*(o, \iota)$ ; then  $z = y - (\text{oriented tree})$  is feasible for the transportation problem of this profile (row sums  $c_x - [x = t] - o_x \leq c_x - o_x$ , etc.), so the value of  $y$  is at most  $w^*(o, \iota) + T(o, \iota)$ .  $\square$

Since  $\text{lcm}$  is symmetric, transposition maps optimal transportation plans of the profile  $(o, \iota)$  to those of  $(\iota, o)$  and preserves forced weights, so the two profiles may be merged. The table of profiles with their maximal forced weights does not depend on the input: for each of the  $2^7 - 1$  subsets  $P$  it is obtained by enumerating all  $k^{k-2}$  labeled spanning trees (Cayley's formula, via Prüfer sequences) and all  $2^{k-1}$  orientations, in total  $\sum_{k=1}^7 \binom{7}{k} k^{k-2} 2^{k-1} < 1.5 \cdot 10^6$  oriented trees, once before reading the tests. After merging, the number of distinct profiles is 32 613 for  $k = 7$  and 67 158 summed over all subsets.

The transportation problems are solved exactly by min-cost flow on a fixed network with 16 vertices (a source, 7 row vertices, 7 column vertices, a sink); the transportation polytope is integral, so the flow optimum is an integer matrix  $z$ . First the *base* problem — caps  $c_x$  on both sides, total  $n - 1$  — is solved by successive shortest paths with potentials; each augmentation ships at least one unit, so it finishes after at most  $n - 1$  shortest-path computations. Every profile problem is then solved as a *bounded correction* of the base optimum: truncate the base flow to the profile's caps and to the total  $n - k$ ; since a profile subtracts only  $k - 1$  units from each side, this creates at most  $(k - 1)$  units of excess at the column side and  $(k - 1)$  units of total-flow excess, while the row side only acquires deficits — at most  $2(k - 1) \leq 12$  units of positive imbalance in total. The base potentials remain dual-feasible, because the dual constraints

$\alpha_i + \beta_j + \gamma \geq \text{lcm}(i, j)$  do not involve the caps, and truncation never creates a residual arc that was absent at the base optimum. Successive shortest-path repair therefore restores feasibility while maintaining reduced-cost optimality, and each augmentation removes at least one unit of positive imbalance — at most 12 Dijkstra computations on the 16-vertex network per profile, each  $O(16^2)$  operations.

Three exact accelerations follow from the same analysis and do not affect the worst-case bound. First, if the support of the base optimum is already connected over  $P$ , its value  $R_0$  is the answer by Lemma 1 (and the answer never exceeds  $R_0$ , since appending the  $k - 1$  forced units to any residual plan gives a feasible base plan). Second, scanning profiles in decreasing order of  $w^*$ , one may stop as soon as  $w^* + R_0 - (k - 1) \leq \text{best}$ : every unit shipped has value at least 1, so  $T(o, \iota) \leq R_0 - (k - 1)$  for every profile. Third, any dual-feasible triple  $(\alpha, \beta, \gamma)$  read off the base potentials yields, by weak duality, the upper bound  $w^* + \sum_x \alpha_x(c_x - o_x) + \sum_x \beta_x(c_x - \iota_x) + (n - k)\gamma$  for a profile at the cost of  $O(7)$  operations, and profiles whose bound does not beat the current best are skipped without a flow computation.

**Complexity:** with alphabet size  $m = 7$ , a one-time precomputation of  $\sum_{k \leq m} \binom{m}{k} k^{k-2} 2^{k-1} < 1.5 \cdot 10^6$  oriented trees builds the profile tables; each test case takes  $O(n)$  for reading plus at most 32613 transportation corrections of at most 12 augmentations each, i.e. at most  $32613 \cdot 12$  Dijkstra computations on a 16-vertex network ( $\approx 10^8$  elementary operations, in practice tens of milliseconds), while the base solves contribute at most  $\sum n$  augmentations over the whole input. In total  $O(\sum n + t \cdot m^{m-2} 2^{m-1})$  with all constants explicit and proven.

## Problem I. Interval MST

**Problem.** Given a sequence  $x_1, x_2, \dots, x_n$  ( $1 \leq n \leq 2 \cdot 10^5$ ,  $|x_i| \leq 10^9$ ), consider the complete graph on vertices  $1, \dots, n$  in which the edge  $\{u, v\}$  with  $u < v$  has weight  $x_v - x_u$  (possibly negative). For each of  $q \leq 2 \cdot 10^5$  queries  $(l, r)$  compute  $f(l, r)$  — the minimum weight of a spanning tree of the subgraph induced by vertices  $l, l + 1, \dots, r$  (with  $f(l, l) = 0$ ).

**Solution.** Fix a query with  $l < r$  and call a vertex  $i$  with  $l < i < r$  *internal*. For an internal  $i$  let  $P_i = \max_{l \leq j < i} x_j$ ,  $S_i = \min_{i < j \leq r} x_j$ , and  $c_i = \min(0, x_i - P_i, S_i - x_i)$ . Note that  $\min(x_i - P_i, S_i - x_i)$  is exactly the weight of the cheapest edge incident to  $i$ .

*Lemma.*  $f(l, r) = x_r - x_l + \sum_{i=l+1}^{r-1} c_i$ .

*Lower bound.* Take any spanning tree  $T$  and root it at  $r$ ; every vertex except  $r$  owns its parent edge, so every edge is owned exactly once. Let  $l = q_0, q_1, \dots, q_s = r$  be the tree path from  $l$  to  $r$ ; the path edges are owned by  $q_0, \dots, q_{s-1}$ , and every off-path vertex  $i$  is internal and its owned edge is incident to  $i$ , hence weighs at least  $c_i$ . Split the path into maximal *descending runs*, i.e. maximal stretches  $q_j, \dots, q_t$  with strictly decreasing indices; inside a run the weights telescope, so the total path weight equals  $(x_r - x_l) + 2 \sum_{\text{runs}} (x_a - x_b)$ , where a run goes from position  $a$  down to position  $b < a$ . Both  $a$  and  $b$  are internal: a run cannot start at  $l$  (nothing is to the left of  $l$ ) nor at  $r$  ( $r$  is last on the path), and it cannot end at  $l$  or  $r$  (those are the first and last path vertices, and the last step into  $r$  goes rightward). Now  $x_a - x_b \geq x_a - P_a \geq c_a$  (since  $b < a$  implies  $x_b \leq P_a$ ) and  $x_a - x_b \geq S_b - x_b \geq c_b$  (since  $a > b$  implies  $x_a \geq S_b$ ), so each run contributes at least  $c_a + c_b$ . Runs are pairwise vertex-disjoint and disjoint from off-path vertices, hence  $w(T) \geq (x_r - x_l) + \sum_{i \in I} c_i$  for a set  $I$  of distinct internal vertices, and since all  $c_i \leq 0$  this is  $\geq (x_r - x_l) + \sum_i c_i$ .

*Upper bound.* Call an internal  $i$  a *pivot* if  $c_i = 0$ , i.e.  $x_i \geq P_i$  and  $x_i \leq S_i$ . Let  $l = m_0 < m_1 < \dots < m_k < m_{k+1} = r$ , where  $m_1, \dots, m_k$  are all pivots. It is enough to construct a tree independently on every interval  $[m_j, m_{j+1}]$  and then glue these trees at their common endpoints. The values  $P_i, S_i$ , and  $c_i$  inside such an interval do not change when they are computed locally: its left pivot dominates everything farther to the left, while its right pivot dominates everything farther to the right.

Thus, consider an interval  $[a, b]$  with no internal pivots. For every  $a < i < b$  we have  $c_i < 0$ , so there exists an edge incident to  $i$  whose weight is exactly  $c_i$ . Add the edge  $(a, b)$  and, for every internal vertex  $i$ , one such cheapest incident edge. We obtain a multigraph with exactly  $b - a$  edges and the required total weight  $x_b - x_a + \sum_{i=a+1}^{b-1} c_i$ .

Let  $A$  be the component containing  $a$  and  $b$ . If it contains  $h$  internal vertices, then it has  $h + 2$  vertices

and  $h + 1$  edges, so  $A$  is a tree. Every other component  $C$  consists only of internal vertices. If it contains  $h$  vertices, then those vertices contributed exactly  $h$  edges, so  $C$  is connected and contains exactly one cycle.

If there are no such components, the multigraph is already the required tree. Otherwise, let them be  $C_1, \dots, C_t$ . In the unique cycle of every  $C_j$ , choose an edge  $(p_j, q_j)$  with  $p_j < q_j$ , and renumber the components so that  $p_1 < p_2 < \dots < p_t$ . Delete  $(a, b), (p_1, q_1), \dots, (p_t, q_t)$  and add  $(a, q_1), (p_1, q_2), (p_2, q_3), \dots, (p_{t-1}, q_t), (p_t, b)$ .

All new edges are valid: in particular,  $p_j < p_{j+1} < q_{j+1}$ . Deleting  $(a, b)$  splits the tree  $A$  into two components, while deleting a cycle edge leaves every  $C_j$  connected. The new edges join all these parts into the chain  $A_{\text{left}} - C_1 - C_2 - \dots - C_t - A_{\text{right}}$ .

The resulting graph is connected and still has  $b - a$  edges, hence it is a tree. Its weight is unchanged: the deleted and added edges have the same multisets of left endpoints and right endpoints. In both cases the left endpoints are  $\{a, p_1, \dots, p_t\}$ , and the right endpoints are  $\{q_1, \dots, q_t, b\}$ .

Therefore, the sum of  $x_v - x_u$  over all edges remains the same. Gluing the trees constructed between consecutive pivots gives a spanning tree of weight  $\sum_{j=0}^k (x_{m_{j+1}} - x_{m_j}) + \sum_{i: c_i < 0} c_i = x_r - x_l + \sum_{i=l+1}^{r-1} c_i$ .

This proves the upper bound and the Lemma. Evaluating the formula directly gives an easy  $O(r - l)$  solution per query, which is too slow.

*Answering queries.* Process everything offline on a segment tree over positions. Each query  $[l, r]$  is split into  $O(\log n)$  canonical blocks; when a block  $B$  is processed, the rest of the query acts on it only through two numbers:  $P$ , the maximum of  $x$  over the part of the query to the left of  $B$  ( $-\infty$  if empty), and  $S$ , the minimum over the part to the right ( $+\infty$  if empty). Inside  $B$  precompute for every position  $i$  the strict block-local prefix maximum  $p_i = \max_{j \in B, j < i} x_j$  and strict block-local suffix minimum  $s_i = \min_{j \in B, j > i} x_j$ , and let  $c_i^B = \min(0, x_i - p_i, s_i - x_i)$ ; then the true term of position  $i$  equals  $\min(0, x_i - \max(p_i, P), \min(s_i, S) - x_i) = \min(c_i^B, x_i - P, S - x_i)$ . Summing this over all  $i \in [l, r]$  counts the endpoints too: position  $l$  contributes  $h_L = \min(0, S_l - x_l)$  and position  $r$  contributes  $h_R = \min(0, x_r - P_r)$ , which do not belong to the formula, so initialize the answer with  $x_r - x_l - h_L - h_R$  (computed with two ordinary range max/min queries) and let the block machinery add the terms of all positions.

So, per block we must sum  $\min(c_i, x_i - P, S - x_i)$  (write  $c_i$  for  $c_i^B$ ) over its points for many pairs  $(P, S)$ . Let  $u_i = x_i - c_i$  and  $v_i = x_i + c_i$ . The minimum is attained by exactly one of three disjoint cases: it equals  $c_i$  iff  $u_i \geq P$  and  $v_i \leq S$ ; it equals  $x_i - P$  iff  $u_i < P$  and  $2x_i \leq P + S$ ; it equals  $S - x_i$  iff  $v_i > S$  and  $2x_i > P + S$ . In the region  $u_i < P$ ,  $v_i > S$ , the comparison between  $2x_i$  and  $P + S$  decides between the latter two cases; the remaining mixed regions are automatically covered by the same conditions. Requests with one-sided context ( $P = -\infty$  or  $S = +\infty$ ) degenerate to a single binary search over points sorted by  $v_i$  (resp.  $u_i$ ) with prefix sums of  $x_i$  and  $c_i$ . For two-sided requests, three sweeps over the block answer all of them: (1) requests by decreasing  $P$ , inserting points in order of decreasing  $u_i$  into a Fenwick tree indexed by  $v_i$  storing  $\sum c_i$ , query the prefix  $v_i \leq S$ ; (2) requests by increasing  $P$ , inserting points with  $u_i < P$  into a Fenwick tree indexed by  $2x_i$  storing count and  $\sum x_i$ , query the prefix  $2x_i \leq P + S$  to get  $\sum (x_i - P)$ ; (3) requests by decreasing  $S$ , inserting points with  $v_i > S$  into the same Fenwick tree, take the complement above  $P + S$  to get  $\sum (S - x_i)$ . All values are coordinate-compressed; requests are bucketed by their node, and only subtrees containing requests are visited.

Each query produces  $O(\log n)$  requests; a node with  $s$  points and  $t$  requests costs  $O((s + t) \log n)$  for sorting, Fenwick operations and binary searches, and the total size of all node point sets is  $O(n \log n)$ .

**Complexity:**  $O((n + q) \log^2 n)$  time and  $O(n + q \log n)$  memory.

## Problem J. Jump Cycle

**Problem.** Given an even integer  $n$  ( $2 \leq n \leq 200$ ), arrange all  $n^2$  cells of an  $n \times n$  grid into a cyclic sequence  $p_1, p_2, \dots, p_{n^2}$  (with  $p_{n^2+1} = p_1$ ) in which every cell appears exactly once, every two consecutive cells lie in the same row or in the same column, and the length of the  $i$ -th move equals  $|r_i - r_{i+1}| + |c_i - c_{i+1}| = ((i - 1) \bmod 4) + 1$ ; that is, the move lengths are 1, 2, 3, 4, 1, 2, 3, 4, ... cyclically,

including the closing move from  $p_{n^2}$  back to  $p_1$ . If such a tour exists, output for every cell the moment at which it is visited; otherwise report that no tour exists.

**Solution.** We prove that a tour exists if and only if  $n$  is divisible by 8, and for such  $n$  we give an explicit construction.

*Necessity:  $n$  must be divisible by 8.* Call the color of  $p_1$  *special*. The parities of the move lengths follow the pattern odd, even, odd, even,  $\dots$ , so the colors of the visited cells are  $S, \bar{S}, \bar{S}, S, S, \bar{S}, \bar{S}, S, \dots$ ; in other words, the cells of the special color are visited exactly at the positions  $i \equiv 0$  or  $i \equiv 1 \pmod{4}$ .

Now assign to every cell the pair of residues  $((r-1) \bmod 4, (c-1) \bmod 4)$ ; this splits the board into 16 residue classes, and all cells of one class share the same chessboard color (the parity of the sum of the two residues). A move of length 4 is horizontal or vertical and therefore preserves both residues. The length-4 moves are exactly the moves  $p_{4t} \rightarrow p_{4t+1}$ , so these moves form a bijection between the cells visited at positions  $i \equiv 0 \pmod{4}$  and the cells visited at positions  $i \equiv 1 \pmod{4}$ , and this bijection maps every residue class to itself. Since the cells of the special color are precisely those visited at positions  $i \equiv 0, 1 \pmod{4}$ , every residue class of the special color is split evenly between the two position types, and hence *has even size*.

It remains to count the class sizes. If  $n = 4k$ , every one of the 16 classes has exactly  $k^2$  cells; since the special-color classes must have even size,  $k^2$  is even, so  $k$  is even and  $8 \mid n$ . If  $n = 4k + 2$ , the row residues 0 and 1 occur  $k + 1$  times each and the residues 2 and 3 occur  $k$  times each (the same for columns). If  $k$  is even, the classes  $(0, 0)$  and  $(0, 1)$  both have odd size  $(k + 1)^2$ ; if  $k$  is odd, the classes  $(2, 2)$  and  $(2, 3)$  both have odd size  $k^2$ . In either case the two named classes have opposite chessboard colors, so one of them is of the special color and has odd size — a contradiction. Therefore  $n \equiv 2 \pmod{4}$  is impossible, and altogether  $8 \mid n$  is necessary.

*Construction for  $n = 8m$ : the base block.* The whole construction is assembled from one explicit closed tour of an  $8 \times 8$  block. In the matrix below, the number in each cell is the local visiting time of that cell:

$$B = \begin{pmatrix} 1 & 2 & 9 & 10 & 64 & 62 & 40 & 63 \\ 6 & 60 & 7 & 56 & 59 & 61 & 58 & 57 \\ 5 & 3 & 13 & 11 & 4 & 30 & 12 & 31 \\ 51 & 21 & 14 & 52 & 15 & 29 & 39 & 16 \\ 49 & 22 & 8 & 55 & 48 & 42 & 41 & 47 \\ 50 & 36 & 34 & 33 & 35 & 37 & 38 & 32 \\ 25 & 23 & 44 & 54 & 24 & 43 & 45 & 46 \\ 26 & 20 & 27 & 53 & 19 & 28 & 18 & 17 \end{pmatrix}.$$

Reading it as a cycle  $q_1, q_2, \dots, q_{64}, q_{65} = q_1$  (where  $q_t$  is the cell containing  $t$ ), one verifies directly that every consecutive pair lies in one row or one column and that the move lengths are  $1, 2, 3, 4, 1, 2, 3, 4, \dots$  around the whole cycle.

*Tiling and splicing.* Partition the  $n \times n$  board into  $m \times m$  blocks of size  $8 \times 8$  and place a translated copy of  $B$  into each block. This gives  $m^2$  disjoint 64-cycles, which we now merge into a single cycle. Every splice uses only moves of length 4, i.e. edges  $q_{4t} \rightarrow q_{4t+1}$ .

For a *horizontal splice* of two side-by-side blocks  $L$  (left) and  $R$  (right), note that in  $B$  we have  $q_{40} = (1, 7)$ ,  $q_{41} = (5, 7)$ ,  $q_8 = (5, 3)$ ,  $q_9 = (1, 3)$ , so  $q_{40} \rightarrow q_{41}$  and  $q_8 \rightarrow q_9$  are length-4 edges. Since  $R$  is shifted by 8 columns, we may delete these two edges (the first in  $L$ , the second in  $R$ ) and insert the cross edges  $q_{40}^{(L)} \rightarrow q_9^{(R)}$  and  $q_8^{(R)} \rightarrow q_{41}^{(L)}$ : the first goes from  $(1, 7)$  to  $(1, 11)$  and the second from  $(5, 11)$  to  $(5, 7)$  in the coordinates of  $L$ , so both are horizontal moves of length 4.

For a *vertical splice* of an upper block  $U$  and a lower block  $D$ , we use  $q_{32} = (6, 8)$ ,  $q_{33} = (6, 4)$ ,  $q_{56} = (2, 4)$ ,  $q_{57} = (2, 8)$ , so  $q_{32} \rightarrow q_{33}$  and  $q_{56} \rightarrow q_{57}$  are again length-4 edges. With  $D$  shifted by 8 rows, replace them by  $q_{32}^{(U)} \rightarrow q_{57}^{(D)}$  (from  $(6, 8)$  to  $(10, 8)$ ) and  $q_{56}^{(D)} \rightarrow q_{33}^{(U)}$  (from  $(10, 4)$  to  $(6, 4)$ ): both are vertical moves of length 4.

In each splice we remove one edge from each of two disjoint cycles and reconnect the four loose ends crosswise, which merges the two cycles into one. We apply the splices along a spanning tree of the  $m \times m$

grid of blocks: inside every block-row we splice each pair of neighboring blocks horizontally, and inside the first block-column we splice each pair of neighboring blocks vertically. A block in the middle of a row loses its edge  $q_{40} \rightarrow q_{41}$  to the splice on its right and its edge  $q_8 \rightarrow q_9$  to the splice on its left, and a first-column block additionally loses  $q_{32} \rightarrow q_{33}$  and  $q_{56} \rightarrow q_{57}$  to the vertical splices; these four edges are pairwise distinct, so no edge is ever spliced twice. After the  $m^2 - 1$  splices all cells lie on one closed tour.

*Why the move pattern survives.* Inside every block, each edge goes from local label  $t$  to local label  $t + 1 \pmod{64}$ , so the label increases by 1 modulo 4 along every edge, and the length of the edge leaving label  $t$  is 1, 2, 3, 4 according to  $t \equiv 1, 2, 3, 0 \pmod{4}$ . Every inserted cross edge leaves a label divisible by 4 (namely 40, 8, 32, or 56), enters a label congruent to 1 modulo 4 (namely 9, 41, 57, or 33), and has length 4 — exactly the same behavior. Hence along the final global cycle the local label modulo 4 increases by 1 on every step, and every step's length is determined by that label in the required way. Starting the numbering of the global tour at any cell whose local label is congruent to 1 modulo 4, the global position modulo 4 coincides with the local label modulo 4 everywhere, so the move lengths are 1, 2, 3, 4, 1, 2, 3, 4, ... throughout, including the closing move (recall  $n^2 \equiv 0 \pmod{4}$ ). Finally, we output for every cell the moment at which the tour visits it.

**Complexity:**  $O(n^2)$ .

## Problem K. Keep Colors Apart

**Problem.** There are  $n$  events; event  $i$  has a deadline  $d_i$ , a color  $c_i$  and a weight  $w_i$  ( $1 \leq d_i, c_i, w_i \leq 10^9$ ), and every color occurs at most twice. Choose a subset of the events and assign the chosen events to pairwise distinct positive integer days so that an event assigned to day  $x$  has  $x \leq d_i$ , and whenever two consecutive days both carry events, these events have different colors. Maximize the total weight of the chosen events. The sum of  $n$  over all test cases does not exceed  $2 \cdot 10^5$ .

**Solution.** Cap all deadlines at  $n + 1$ : a schedule never needs a day beyond  $n + 1$ , because chosen events can always be shifted earlier day by day (moving an event earlier never violates its deadline, and a whole block can be shifted preserving adjacency). Process days  $1, 2, \dots, n + 1$  from left to right and commit each chosen event at the step equal to its capped deadline. After processing day  $i$  the schedule occupies a prefix of days  $1..i$  with some *slack*  $s = i - (\text{number of chosen events})$ .

*The state machine.* The following states describe the prefix after day  $i$ :

*free*[ $s$ ] for  $s \geq 1$  —  $s$  empty days, and the arrangement can be chosen so that no color constraint propagates into the future. *free*[0] — all  $i$  days are occupied, but the color of day  $i$  is not pinned (there are at least two choices for it). *force*[ $c$ ] — all days are occupied and the event on day  $i$  necessarily has color  $c$ ; since that event must have deadline exactly  $i$ , only colors of deadline- $i$  events appear here. *empty* — days  $1..i - 1$  are occupied and day  $i$  is empty; unlike *free*[1], the single empty slot is pinned at the frontier.

The key *untangling lemma*: if the slack is at least 1, a chosen multiset of events in which every color occurs at most twice can always be arranged without equal colors on adjacent days, and moreover the frontier can be kept unconstrained. Indeed, a clash is always an isolated same-colored pair (colors never occur three times), and one event of the pair can be shifted earlier into the empty slot, shifting the intermediate events one day earlier — deadlines are upper bounds, so shifting earlier is always legal. This is why *free* states may ignore colors entirely.

Transitions at day  $i$  with event group  $G$  (deadline  $i$ ), where  $\text{best}(x)$  is the sum of the  $x$  largest weights in  $G$ : from *free*[ $s$ ],  $s \geq 1$ , take any  $x \leq s + 1$  events, arriving at *free*[ $s + 1 - x$ ] with value  $+\text{best}(x)$ ; from *free*[0], either take nothing (going to *empty*) or take a single event of color  $c$  (going to *force*[ $c$ ]) — only day  $i$  is available, and the unpinned previous color never blocks a single event; from *force*[ $c$ ], take nothing ( $\rightarrow$  *empty*) or one event of a color  $c' \neq c$  ( $\rightarrow$  *force*[ $c'$ ]); from *empty*, take nothing ( $\rightarrow$  *free*[2]), one event ( $\rightarrow$  *free*[1] — the event may be placed on either of the two frontier days, so the slot becomes movable), or two events of *different* colors ( $\rightarrow$  *free*[0] — they occupy days  $i - 1, i$  in either order, so the final color has two distinct options). The answer is the maximum over all states after day  $n + 1$ . Implemented directly, this is  $O(n^2)$ .

*Speedup to  $O(n \log n)$ .* The function  $j \mapsto \text{free}[j]$  is concave (its marginal drops are the weights of the

marginal events, sorted), so represent it by the multiset  $\Gamma$  of its *gains*: after day  $i$ , the value at slack  $s$  equals the sum of the  $i - s$  largest elements of  $\Gamma$  (level 0 is the empty schedule of value 0). The bulk transition is a  $(\max, +)$  convolution of two concave sequences —  $free$  and  $x \mapsto \text{best}(x)$  — which is simply the merge of their difference multisets: insert all weights of  $G$  into  $\Gamma$ . The constraint  $x \leq s + 1$  turns into “use at most  $i$  levels”, and the exclusion of  $free[0]$  as a source is exactly the invariant  $|\Gamma| \leq i - 1$ , restored at the end of each day by deleting the smallest gains — the same exchange step as in the classic unit-job deadline-scheduling greedy, and it is safe by the same matroid-exchange argument. The tight-chain scalars  $free[0]$ ,  $empty$  and  $force[\cdot]$  are  $O(1)$  values per day ( $force$  ranges over the colors of the current group only, so their total number is  $O(n)$ ; the maximum over  $force[c']$  with  $c' \neq c$  is a top-two query). They exchange values with the multiset only at its deepest levels: the candidates from  $empty$  touch slack 2 and 1, so each day the two smallest used gains are recomputed as maxima of the convolution value and the scalar candidates and reinserted; concavity guarantees the representation stays exact. Every event is inserted once and deleted at most once, and each day does  $O(1)$  additional multiset work.

**Complexity:**  $O(n \log n)$  time,  $O(n)$  memory per test case.

## Problem L. Leaf Meeting

**Problem.** Given a tree with  $n$  vertices ( $2 \leq n \leq 2 \cdot 10^5$ , sum of  $n$  over all tests up to  $2 \cdot 10^5$ ), a token stands on every leaf. Each token independently chooses an infinite sequence of leaves  $a_1, a_2, \dots$  (consecutive ones distinct,  $a_1$  different from the starting leaf) and forever walks along the concatenation of shortest paths between consecutive leaves, traversing exactly one edge per move; no token may ever stand still. Decide whether the sequences can be chosen so that at some moment all tokens occupy the same leaf simultaneously.

**Solution.** First reformulate when a single token can be at a leaf. A leaf  $w$  has degree 1, so a shortest path between two other leaves never passes through  $w$  as an internal vertex. Hence a token starting at leaf  $v$  stands on leaf  $w$  at time  $t$  if and only if  $t$  is the total length of a *chain*  $v = a_0 \rightarrow a_1 \rightarrow \dots \rightarrow a_j = w$  of shortest leaf-to-leaf paths (consecutive leaves distinct),  $t = \sum_i \text{dist}(a_{i-1}, a_i)$ . So we must decide: is there a leaf  $w$  and a time  $T$  such that from every leaf  $v$  some chain to  $w$  has total length exactly  $T$ ?

Root the tree at an arbitrary leaf  $r$  and let  $\text{dep}(u)$  be the depth. Call a vertex *special* if its degree is not 2, i.e. it is a leaf or a branching vertex, and let  $d$  be the gcd of the depths of all special vertices. The key structural fact: for two distinct leaves  $a, b$ , their lca is either  $r$  itself (depth 0, when one of them is  $r$ ) or a vertex with at least two children, i.e. of degree at least 3 — in both cases a special vertex. Since leaves are special too, all three quantities  $\text{dep}(a)$ ,  $\text{dep}(b)$ ,  $\text{dep}(\text{lca}(a, b))$  are divisible by  $d$ , and therefore

$$\text{dist}(a, b) = \text{dep}(a) + \text{dep}(b) - 2 \text{dep}(\text{lca}(a, b)) \equiv \text{dep}(a) + \text{dep}(b) \pmod{2d}.$$

**Necessity.** Take any chain  $v = a_0 \rightarrow \dots \rightarrow a_j = w$ . Summing the congruence above over its segments, every intermediate leaf contributes its depth twice, so the total length is  $\equiv \text{dep}(v) + \text{dep}(w) \pmod{2d}$ . If all tokens meet at leaf  $w$  at time  $T$ , then  $T \equiv \text{dep}(v) + \text{dep}(w) \pmod{2d}$  for *every* leaf  $v$ , so all leaf depths are congruent modulo  $2d$ ; plugging in  $v = r$  with  $\text{dep}(r) = 0$  gives: *every leaf depth must be*  $\equiv 0 \pmod{2d}$  (equivalently,  $\text{dep}(v)/d$  is even for every leaf  $v$ ).

**Sufficiency.** Suppose all leaf depths are  $\equiv 0 \pmod{2d}$ ; we meet at  $w = r$ . Consider the set  $S$  of lengths of closed chains  $r \rightarrow \dots \rightarrow r$ ; it is closed under addition (concatenation is legal, since every chain leaving  $r$  starts with a leaf other than  $r$ ), and by the previous paragraph  $S \subseteq 2d\mathbb{Z}$ . We claim the subgroup of  $\mathbb{Z}$  generated by  $S$  is exactly  $2d\mathbb{Z}$ . Indeed, the two-segment chain  $r \rightarrow a \rightarrow r$  shows  $2 \text{dist}(r, a) = 2 \text{dep}(a) \in \langle S \rangle$  for every leaf  $a$ ; the triangle chain  $r \rightarrow a \rightarrow b \rightarrow r$  has length  $2 \text{dep}(a) + 2 \text{dep}(b) - 2 \text{dep}(\text{lca}(a, b))$ , so subtracting the two doubled depths shows  $2 \text{dep}(\text{lca}(a, b)) \in \langle S \rangle$  as well. Every special vertex is either  $r$ , or a leaf, or has at least two children and hence is the lca of two leaves from different child subtrees — so  $\langle S \rangle$  contains twice the depth of every special vertex, hence their gcd  $2d$ . Since  $S$  is an additively closed set of non-negative integers with  $\text{gcd}(S) = 2d$ , it contains *all sufficiently large multiples of*  $2d$  (take elements of  $S$  whose gcd is  $2d$ ; integer combinations with large enough positive coefficients realize every large multiple). Now fix for each leaf  $v$  any one chain  $v \rightarrow \dots \rightarrow r$

of some length  $L_v$ ; as shown,  $L_v \equiv \text{dep}(v) \equiv 0 \pmod{2d}$ . Choose a multiple  $T$  of  $2d$  large enough that every  $T - L_v$  is a sufficiently large multiple of  $2d$ ; the token from  $v$  walks its fixed chain and then appends a closed chain of length  $T - L_v \in S$ , while the token starting at  $r$  uses a closed chain of length  $T \in S$ . All tokens stand on  $r$  at time  $T$ .

So the answer is YES iff, after rooting at a leaf, every leaf depth is divisible by  $2d$  where  $d$  is the gcd of the depths of all vertices of degree  $\neq 2$ . (The criterion does not depend on which leaf is chosen as the root:  $\text{dist}(w, u)$  is a multiple of  $d$  for any leaf  $w$  and special  $u$  by the same lca argument, so  $d$  is root-independent, and  $\text{dep}_w(v) = \text{dist}(w, v) \equiv \text{dep}_r(w) + \text{dep}_r(v) \pmod{2d}$ .)

Implementation: one DFS from any leaf computes all depths; accumulate  $g = \text{gcd}$  of  $2 \text{dep}(u)$  over vertices with degree  $\neq 2$  (the root leaf contributes 0 and is harmless), then verify that all leaves have the same depth modulo  $g$  — since the root is a leaf of depth 0, this is exactly divisibility by  $g = 2d$ . The trivial path case is handled automatically ( $d$  equals the path length  $\ell$ , and the far leaf has depth  $\ell \not\equiv 0 \pmod{2\ell}$ , giving NO, which is correct since the two tokens forever occupy opposite ends).

**Complexity:**  $O(n)$  time and memory per test case (all gcd computations amortize to  $O(n + \log n)$  in total).

## Problem M. Matching Substrings

**Problem.** Given strings  $a$  and  $b$ , answer  $q$  queries. Each query is a string  $c$  of length  $m$ ; for every  $k = 1, \dots, m$  count the distinct strings  $x$  (the empty string is allowed) such that  $x$  is a substring of  $a$  and  $xc_1c_2\dots c_k$  is a substring of  $b$ . Strings  $x$  are compared as strings, not by occurrence. The sums of  $|a|$ , of  $|b|$ , and of all  $|c|$  are each at most 150 000 (over up to  $10^4$  tests).

**Solution.** Reverse everything. Let  $A = \text{rev}(a)$ ,  $B = \text{rev}(b)$ ,  $C = \text{rev}(c)$ ,  $y = \text{rev}(x)$ . Then  $x$  is a substring of  $a$  iff  $y$  is a substring of  $A$ , and  $xc_1\dots c_k$  is a substring of  $b$  iff  $w_k y$  is a substring of  $B$ , where  $w_k = c_k c_{k-1} \dots c_1$  is the suffix of  $C$  of length  $k$ . So for every string  $w$  occurring in  $B$  we want  $f(w)$  — the number of distinct *nonempty* substrings  $y$  of  $A$  with  $wy$  occurring in  $B$ ; the answer for prefix length  $k$  is  $f(w_k) + 1$  if  $w_k$  occurs in  $B$  (the +1 is the empty  $x$ ) and 0 otherwise; distinct strings of equal length are counted separately.

*Lemma 1.*  $f(w)$  depends only on  $\text{endpos}_B(w)$ , the set of ending positions of occurrences of  $w$  in  $B$ . Indeed,  $wy$  occurs in  $B$  iff some occurrence of  $w$  ends at a position  $p$  and  $y$  is a prefix of  $B[p+1\dots]$ . Hence  $f$  is a function of the states of the suffix automaton of  $B$ , and it suffices to compute one value  $f(v)$  per state  $v$ .

*Lemma 2.* Fix an end position  $p$  and let  $L_i$  be the matching statistic of  $B$  against  $A$ : the length of the longest prefix of  $B[i\dots]$  that occurs in  $A$ . The strings  $y$  contributed by  $p$  are exactly the nonempty prefixes of  $B[p+1\dots]$  of length at most  $L_{p+1}$ , because any prefix of a substring of  $A$  is again a substring of  $A$  (downward closedness). In the suffix tree of  $A$ , whose points (including implicit ones inside edges) are in bijection with the distinct nonempty substrings of  $A$ , this set is exactly the path from the root to the locus of  $B[p+1\dots p+L_{p+1}]$ . Therefore  $f(v)$  equals the number of points in the union of root-to-locus paths over all  $p \in \text{endpos}(v)$ .

*Preparation.* Build the suffix array with LCP and sparse table of  $A\#B$ ; for each suffix of  $B$  the maximal LCP with a suffix of  $A$  is attained at the nearest  $A$ -suffix in suffix-array order (prev/next scans), which yields  $L_i$  together with a witness position in  $A$ . Build the suffix tree of  $A$  from its suffix array and LCP (stack construction), locate each locus by binary lifting from the witness leaf to the shallowest node of string depth  $\geq L_i$ , and split edges so that every locus becomes an explicit node — at most  $|B|$  extra nodes. Precompute an Euler-tour LCA over the final tree.

*Union of vertical paths.* Maintain the current multiset of loci in an ordered set sorted by DFS entry time; with  $v_1, \dots, v_r$  the distinct active loci in this order, the union of root paths has size  $\sum_i \text{sdepth}(v_i) - \sum_{i < r} \text{sdepth}(\text{lca}(v_i, v_{i+1}))$ . Proof sketch: scanning loci in DFS order, the path to  $v_i$  adds exactly  $\text{sdepth}(v_i) - \text{sdepth}(\text{lca}(v_{i-1}, v_i))$  new points, since among the already inserted loci the DFS predecessor maximizes the depth of the LCA. Inserting or erasing a locus changes only the two neighbouring pairs, so each update costs  $O(\log |B|)$  (set operation plus  $O(1)$  LCA queries). Repeated loci are handled by per-node multiplicities.

*Aggregation over states.* In the suffix-link tree of the SAM of  $B$ ,  $\text{endpos}(v) = \{p : \text{the state of the prefix } B[0 \dots p] \text{ lies in the subtree of } v\}$ . Attach the locus of position  $p + 1$  (if  $L_{p+1} > 0$ ) to the prefix state of  $B[0 \dots p]$  and run DSU on tree over the suffix-link tree: process light children and discard, keep the heavy child, re-add light subtrees and the node's own attachments, and record  $f(v)$  as the current path-union size. Every attachment is inserted  $O(\log |B|)$  times, so this phase costs  $O(|B| \log^2 |B|)$ .

*Queries.* Feed  $C = \text{rev}(c)$  through the SAM of  $B$  with standard matching statistics (following suffix links on mismatch). The final matched length  $t$  is the longest suffix of  $C$  occurring in  $B$ , i.e. the largest  $k$  with  $c_1 \dots c_k$  a substring of  $b$ ; for  $k > t$  output 0. For  $k = t, t - 1, \dots, 1$  climb suffix links from the final state until the current state's length interval contains  $k$  (the state only moves up as  $k$  decreases, so this is amortized linear) and output  $f(\text{state}) + 1$ .

**Complexity:**  $O((|a| + |b|) \log(|a| + |b|) + |b| \log^2 |b| + |c|)$  time per test case and  $O((|a| + |b|) \log(|a| + |b|))$  memory for the sparse tables.