

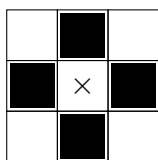
Greece and Cyprus Collegiate Programming Contest 2026

Editorial

24 October 2026

A. Abstract Paintings

First consider a single column. There cannot be three consecutive white cells, by the second condition. There cannot be just *one* white cell between two consecutive black cells either. Indeed, suppose cells (i, j) and $(i + 2, j)$ are black. In each adjacent column $j - 1$ and $j + 1$, cells in rows i and $i + 2$ must be white, since they are next to black cells. The condition on three consecutive rows therefore forces $(i + 1, j - 1)$ and $(i + 1, j + 1)$ to be black. Now the white cell $(i + 1, j)$ has four black neighbours, contradicting the connectivity of the white cells.



The centre white cell
has no white neighbour.

Thus there are *exactly two* white cells between consecutive black cells in every column. In particular n must be a multiple of 3; otherwise the answer is zero. Write $k = n/3$. Every column is determined by a residue $r_j \in \{0, 1, 2\}$: its black cells occur at rows congruent to $r_j \pmod{3}$.

Neighbouring columns cannot have the same residue, since this would give horizontally adjacent black cells. Conversely, different residues cause no such adjacency. Therefore, from column j to column $j + 1$, the residue changes by $d_j = +1$ or -1 modulo 3. This includes the transition from the last column back to the first. Put

$$D = d_0 + d_1 + \cdots + d_{m-1}.$$

To close the pattern around the torus, we need $D \equiv 0 \pmod{3}$.

It remains to check *white connectivity*. Imagine cutting the torus along a vertical seam and following the diagonal chains of black cells. Between these chains are k strips of white cells. Crossing the seam identifies a strip with the strip whose index differs by $D/3$ modulo k : after going around the m columns, the diagonal chains have moved by D rows, or $D/3$ positions in the list of black cells. Consequently, the white strips are joined into

$$\gcd\left(k, \frac{D}{3}\right)$$

connected components. (For $D = 0$, this gives k separate strips; for a shift coprime to k , repeated circuits visit every strip.) All white cells are connected exactly when this gcd is 1.

For a fixed sequence (d_j) there are three choices of r_0 . If exactly t of the m steps are $+1$, then $D = t - (m - t) = 2t - m$, and there are $\binom{m}{t}$ such sequences. Hence the answer is

$$3 \sum_{\substack{0 \leq t \leq m \\ 3 \mid (2t-m) \\ \gcd((2t-m)/3, n/3)=1}} \binom{m}{t} \pmod{998244353}.$$

Precompute factorials and inverse factorials up to m to evaluate the binomial coefficients. The overall complexity is $\mathcal{O}(m \log n)$ time, allowing for the gcd computations, and $\mathcal{O}(m)$ memory.

B. Bus Problem

If some lane has no cars, the answer is **INF**: the bus can use that lane regardless of its length. Otherwise, we can binary-search the integer bus length b . Feasibility is monotone: if a bus of length b can pass, a shorter bus can follow the same route with no more difficulty. Since all car endpoints lie between 0 and 10^9 , it suffices to search $b \in [1, 10^9]$ (and return 0 if even $b = 1$ fails).

Here is a feasibility test for one fixed b .

Free segments and vertices. Split every lane into maximal intervals containing no car interior: the gaps before, between, and after its cars. A bus occupying $[x, x + b]$ fits in a free segment $[L, R]$ exactly when

$$L \leq x \leq R - b.$$

The two unbounded end gaps are handled in the obvious way. Discard finite gaps shorter than b . Create one graph vertex for each remaining gap, and associate with it the interval $I_v = [L_v, U_v]$ of possible *left-end positions* of the bus.

Two vertices on adjacent lanes are joined when $I_u \cap I_v \neq \emptyset$: this intersection is precisely where the bus can change lanes. The number of such edges is $\mathcal{O}(n + m)$. To see this, note that the intervals on any one lane are disjoint and ordered, so the intersections of two neighbouring lanes can be enumerated by merging their sorted interval lists, with two pointers. Over all lane pairs, each list is visited at most twice.

Earliest reachable position. For each vertex v , let e_v be the smallest reachable left-end position of the bus in that gap. Initially, $e_v = -\infty$ for the leftmost gap of every lane, since the bus can start arbitrarily far to the left; all other values are $+\infty$.

Suppose e_u is known. For an adjacent vertex v , the earliest possible lane change from u to v is at

$$x = \max(e_u, L_v).$$

This change is possible if $x \leq \min(U_u, U_v)$, in which case we relax $e_v \leftarrow \min(e_v, x)$. The bus may drive right inside u before the change; it never needs to drive left.

Run a Dijkstra-like computation with a priority queue ordered by e_v . The relaxations never decrease the current position, so once the smallest tentative position is extracted, it is final, just as in Dijkstra's algorithm. The bus can finish if any rightmost unbounded gap is reachable. Conversely, every physical route passes through these gaps and lane changes, and is represented by graph relaxations; thus the test is exact.

There are $\mathcal{O}(n + m)$ vertices and edges, so one test takes $\mathcal{O}((n + m) \log(n + m))$ time after sorting the cars. Including binary search, the complexity is

$$\mathcal{O}((n + m) \log(n + m) \log 10^9).$$

C. Coprime Pairs

Let μ be the Möbius function: $\mu(1) = 1$, $\mu(d) = 0$ if d is divisible by a squared prime, and otherwise $\mu(d) = (-1)^r$ if d has r distinct prime factors. Its useful property is

$$[\gcd(a, b) = 1] = \sum_{d|a, d|b} \mu(d).$$

Indeed, when $g = \gcd(a, b)$, the right-hand side is $\sum_{d|g} \mu(d)$, equal to 1 for $g = 1$ and 0 otherwise.

Let $c_d(x)$ denote the number of vertices in the subtree of x whose values are divisible by d . Interchanging the order of summation gives

$$\text{ans}(x) = \sum_{d=1}^{10^5} \mu(d) \binom{c_d(x)}{2}.$$

We cannot evaluate this formula independently for every vertex and every d . Instead, solve the following subproblem for each d with $\mu(d) \neq 0$.

Subproblem. Mark the k vertices whose values are divisible by d . We need to count, for every subtree, the unordered pairs of marked vertices contained in it, in roughly $\mathcal{O}(k)$ time.

Rather than adding a pair to every ancestor of its two endpoints, assign the pair only to their lowest common ancestor (LCA). Let $f_d(v)$ be the number of marked pairs with LCA exactly v . Then the number of marked pairs in the subtree of x is simply $\sum_{v \in \text{subtree}(x)} f_d(v)$.

Virtual tree. Preprocess the original rooted tree for LCA queries and Euler-tour entry times. List the k marked vertices in DFS order. Build their *virtual tree*: it contains the marked vertices and the LCAs of consecutive marked vertices in DFS order, connected according to ancestry. It has at most $2k - 1$ vertices. A stack constructs it in linear time from the sorted list, using LCA queries; each vertex enters and leaves the stack once. With an Euler-tour RMQ structure, each LCA query can be answered in $\mathcal{O}(1)$ time after preprocessing.

For a virtual-tree vertex v , compute bottom-up

$$c(v) = [v \text{ is marked}] + \sum_{u \text{ virtual child of } v} c(u).$$

Every pair of marked vertices in the virtual subtree of v either has LCA v or lies entirely below one virtual child. Therefore

$$f_d(v) = \binom{c(v)}{2} - \sum_{u \text{ virtual child of } v} \binom{c(u)}{2}.$$

For vertices absent from the virtual tree, $f_d(v) = 0$: an LCA of two marked vertices is necessarily present. Add $\mu(d)f_d(v)$ to a global array $w[v]$. Once all values of d have been processed, a single bottom-up traversal of the *original* tree computes

$$\text{ans}(x) = w[x] + \sum_{u \text{ child of } x} \text{ans}(u).$$

This is exactly the required Möbius sum.

Keeping the total work small. Enumerate the divisors of each a_v , visiting the vertices in DFS order and appending v to the list for every divisor of a_v . Thus every marked-vertex list is already sorted; it does not need to be sorted separately. All virtual trees together then require roughly

$$\mathcal{O}\left(n + \sum_{v=1}^n \tau(a_v)\right)$$

LCA and stack operations, where $\tau(a)$ is the number of divisors of a (plus LCA preprocessing). For $a \leq 10^5$, $\tau(a) \leq 128$; the sum of $\tau(a)$ for $1 \leq a \leq 10^5$ is only about $1.17 \cdot 10^6$. Even in the worst case of repeated vertex values, there are at most $1.28 \cdot 10^7$ marked occurrences. An $\mathcal{O}(1)$ -query LCA structure keeps this bound linear in the number of marked occurrences.

The answers may be as large as $\binom{n}{2}$, so use 64-bit integers.

D. Distinct Orders

For each k , there are 2^k possible orders of the first k dishes, one for each choice of versions. Consequently, no solution can have more than

$$\sum_{k=1}^n 2^k = 2^{n+1} - 2$$

days. We will attain this bound.

Let X be a sequence for $n - 1$ dishes that uses all possible orders exactly once and, after its last day, returns every dish to its initial version. Let $X + 1$ mean adding 1 to every number in X . Construct the sequence for n dishes as

$$(X + 1), 1, (X + 1), 1.$$

The base case is $X = (1, 1)$ for one dish. For example, for two dishes this produces $(2, 2, 1, 2, 2, 1)$.

Why are the orders distinct? Describe each dish by a bit: the version that would be served if the dish were ordered next. Ordering the first k dishes prints their k current bits and then flips those bits.

Each day of $X + 1$ orders dish 1 and flips its bit, while the orders on dishes $2, \dots, n$ are exactly those produced by X . The length of X is even (it contains $\sum_{k=1}^{n-1} 2^k$ orders), so dish 1 returns to its starting bit after a whole copy of $X + 1$. The other dishes also return to their starting bits by the induction hypothesis.

Thus the first and second copies of $X + 1$ begin with *identical states of dishes* $2, \dots, n$, but with *opposite states of dish* 1: the intervening order of length 1 flips precisely that dish. At corresponding days of the two copies, the orders have the same suffix x (versions of dishes $2, \dots, k$), and their first bits are opposite. Together they therefore produce both $0x$ and $1x$ exactly once, for every possible suffix x and every $k \geq 2$.

The two singleton orders also have opposite bits, because the first singleton flips dish 1 and the intervening copy has even length. After the final singleton, dish 1 and all other dishes are back in their initial states. This completes the induction.

The constructed length satisfies $L_1 = 2$ and $L_n = 2L_{n-1} + 2 = 2^{n+1} - 2$, so it is optimal. The construction takes $\mathcal{O}(2^n)$ time, proportional to its output size.

E. Eurovision Lights

Set $a_0 = 0$. Switching on one lamp changes the number of lit segments by $+1$, 0 , or -1 : respectively, it creates a separate segment, extends an existing one, or joins two segments. There must always be at least one lit segment after the first step, and after the last step the whole row is lit. Therefore the necessary conditions are

$$a_n = 1, \quad a_i \geq 1 \ (1 \leq i \leq n), \quad |a_i - a_{i-1}| \leq 1 \ (1 \leq i \leq n).$$

They are also sufficient. We can construct the *actual row* directly, maintaining one simple invariant: **every gap between consecutive lit segments consists of exactly one unlit lamp**. We build the row from left to right, leaving some positions unlit for later.

At step i , let $d = a_i - a_{i-1}$:

- If $d = +1$, append an unlit lamp as a separator (except at step 1), then append a lit lamp, switched on at step i . This creates a new segment.
- If $d = 0$, append a lit lamp, switched on at step i . It extends the rightmost segment.
- If $d = -1$, switch on one of the previously left unlit separators. This merges the two adjacent segments. Keep the separator positions in a stack to find one immediately.

For example, the required counts $(1, 2, 1, 2, 1)$ lead to positions switched on in the order $1, 3, 2, 5, 4$.

There is always a separator to fill when $d = -1$: just before the step there are $a_{i-1} - 1 \geq 1$ gaps. When $d = +1$, the extra separator and lit lamp also fit within the eventual length n . Indeed, after step i the construction has exactly i lit lamps and $a_i - 1$ unlit separators, hence uses

$$i + a_i - 1$$

positions. Since the segment count can decrease by at most one per remaining step and must end at $a_n = 1$, necessarily $a_i \leq n - i + 1$. Thus $i + a_i - 1 \leq n$ at every step.

At the end, $a_n = 1$, so no separators remain and exactly n positions have been created. Record the time each position is switched on, then invert this array to obtain the required permutation. The algorithm is $\boxed{\mathcal{O}(n)}$ time and memory.

F. Flamingo Lake

Write 1 for a cell with a flamingo and 0 for an empty cell. When a column is added to the east, a new cell $(x, y + 1)$ receives a flamingo precisely when all existing cells among $(x - 1, y), (x, y), (x + 1, y)$ are empty. The corresponding rule for adding a southern row is identical after exchanging rows and columns.

Periodicity away from the growing corner. Fix a row x and a cell (x, y) on the right boundary, with at least two rows below it. In all subsequent eastern expansions, the three rows $x - 1, x, x + 1$ already exist. Their cells do not change, and southern expansions do not affect the calculation of newly added cells in row x . We claim that, regardless of how eastern and southern expansions are interleaved,

$$\boxed{(x, y + 3) = (x, y + 1), \quad (x, y + 4) = (x, y + 2), \quad \dots} \quad (2)$$

where equality means equal occupancy, and the coordinates on the right are considered after their columns have appeared.

First observe that a flamingo at (x, y) forces $(x, y + 1)$ to be empty and $(x, y + 2)$ to contain a flamingo: all three cells $(x - 1, y + 1), (x, y + 1), (x + 1, y + 1)$ must be empty. This repeats, so a flamingo reappears every two columns. The same argument applies to a flamingo at $(x, y + 1)$.

The only remaining possibility is that $(x, y), (x, y + 1)$, and $(x, y + 2)$ are all empty. For $(x, y + 1)$ to be empty, a flamingo must occur at either $(x - 1, y)$ or $(x + 1, y)$. Suppose it is at $(x - 1, y)$. Then $(x - 1, y + 1)$ and $(x, y + 1)$ are empty. Since $(x, y + 2)$ is also empty, its new-column test requires a flamingo at $(x + 1, y + 1)$. These two neighbouring flamingos lie on *opposite* sides of row x . Both reappear every two columns, at alternating horizontal positions, so row x remains empty forever. The other case is symmetric. This proves (2). At the top edge, the all-zero case cannot arise, and the same periodicity holds using the existing neighbours.

Representing a boundary. Let C be the current rightmost column, read from top to bottom. Define two *predicted columns* U, V : U is the column that would be created by the next eastern expansion, and V is the column that would be created by a second eastern expansion immediately afterwards, with no southern expansion between them. For any row index i ,

$$U_i = 1 \iff C_{i-1} = C_i = C_{i+1} = 0, \quad V_i = 1 \iff U_{i-1} = U_i = U_{i+1} = 0,$$

where positions outside the current column are treated as zero. At the top or bottom endpoint, a run of three empty cells in the same row across consecutive columns is impossible: emptiness in the second column requires a flamingo in its only neighbouring row in the first column, which forces a flamingo in the third. Thus the same periodicity holds at both endpoints for a fixed-height column. The next predicted columns after U, V are again U, V , and an eastern expansion changes the triple

$$\boxed{(C, U, V) \mapsto (U, V, U)}. \quad (3)$$

The number of flamingos arriving in that expansion is the number of ones in U .

A southern expansion adds exactly one bit b to the *bottom* of the rightmost column. Its predictions can be updated locally. If the old length of C is h , appending b changes U only at indices $h - 1$ and h , because no other three-cell neighbourhood changes. Consequently, V changes only at indices $h - 2, h - 1, h$. All other predicted bits remain unchanged. We can recompute these few bits from a fixed-length suffix of C, U, V .

Thus for each boundary it suffices to store its length, the last six bits of C, U, V , and the total numbers of ones in U and V . Under (3), we simply rotate the stored suffixes and counts. On appending a bit,

recompute the last few entries of U, V and update their counts by subtracting the old entries and adding the new ones. Both operations take $\mathcal{O}(1)$ time.

Processing all events. Maintain such a summary for the rightmost column R and for the bottom row B , read from left to right. These boundaries share their last cell, the south-east corner. Initially both boundaries are the one-bit string 1, with predictions $U = 0, V = 1$; the answer starts at 1.

For an **E** event, rotate the right-boundary triple as in (3) and add the number of ones in its new actual column to the answer. Its bottommost bit is the new corner, so append this bit to the bottom-row summary. An **S** event does the same with R and B interchanged. Every appended corner bit is incorporated into the predictions of that boundary, so successive expansions of either orientation use the current shape of the lake.

There are n constant-time updates. The complexity is $\boxed{\mathcal{O}(n)}$ time and $\boxed{\mathcal{O}(1)}$ extra memory, and the answer fits in a 64-bit integer.

G. Gas Stations

Make vertex 0 stand for the start, with $p_0 = a_0 = 0$. For two stations $j < i$ (and also for $j = 0$), the car can refuel at j and then reach and refuel at i if and only if

$$\frac{a_j}{2} + (p_i - p_j) + \frac{a_i}{2} \leq l.$$

The first term is the return trip from station j , and the last is the outward trip to station i . After multiplying by 2 this becomes

$$2p_i + a_i \leq 2p_j - a_j + 2l.$$

Introduce two coordinates:

$$\boxed{L_i = 2p_i + a_i, \quad R_j = 2p_j - a_j + 2l.}$$

Thus travelling directly from the previous refuelling j to the next one i is possible exactly when $R_j \geq L_i$.

Create a directed edge $j \rightarrow i$ whenever this holds. Traversing that edge adds cost a_i for the new station's detour. Thus the problem is a shortest-path problem: the sum of edge costs equals the sum of detour lengths, and the unavoidable main-road distance d is added at the end. All edge costs are nonnegative, so Dijkstra is applicable; moreover, edges only go from smaller to larger station indices, allowing an even simpler dynamic program.

Let dp_i be the minimum total detour length for a route whose last refuelling is at i , and set $dp_0 = 0$. Then

$$\boxed{dp_i = a_i + \min_{\substack{0 \leq j < i \\ R_j \geq L_i}} dp_j.}$$

An empty minimum means $dp_i = +\infty$. The formula suggests a geometric transformation: each already processed station is a point at coordinate R_j with weight dp_j , and station i asks for the smallest weight among points with coordinate at least L_i .

Coordinate-compress all R_j and L_i . Scan stations in their given increasing order. Use a segment tree that stores the smallest dp_j at each R_j , supporting a point minimum update and a suffix minimum query. Initially insert $R_0 = 2l$ with value 0. At station i , query the suffix $[L_i, \infty)$, calculate dp_i , and insert (R_i, dp_i) if it is reachable. This takes $\mathcal{O}(n \log n)$ time.

Finally, a route ending with station i can reach the destination if

$$\frac{a_i}{2} + d - p_i \leq l.$$

Minimize $d + dp_i$ among such stations, including $i = 0$ (which represents driving straight from the start). If none qualifies, output -1 . All calculations can use doubled integer distances, avoiding fractional arithmetic. Use 64-bit integers for the result.

H. Hacking the Traffic Lights

Only the starting-time residue $x \bmod c$ matters. Let $f_i(x)$ be the minimum shift needed for light i to be green when the car starts at residue x . Because the shifts of different lights are independent, the minimum total cost for a fixed x is simply

$$DP(x) = \sum_{i=1}^n f_i(x).$$

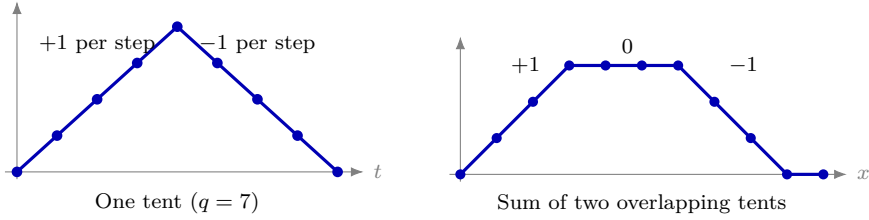
We need to minimize this sum over $0 \leq x < c$, without iterating over all c residues.

Put $L_i = (s_i - d_i) \bmod c$. Without a shift, the light is green exactly at the cyclic interval of starting residues $L_i, L_i + 1, \dots, L_i + g_i - 1$. Since we may shift its phase in either direction, $f_i(x)$ is the shortest *circular* distance from x to this green interval.

There are $q_i = c - g_i$ consecutive forbidden residues. If $q_i = 0$, then $f_i \equiv 0$. Otherwise, let $z_i = (L_i + g_i) \bmod c$ be the first forbidden residue. Number the forbidden residues as $z_i + t \bmod c$ for $0 \leq t < q_i$. Then

$$f_i(z_i + t) = \min(t + 1, q_i - t), \quad 0 \leq t < q_i,$$

and f_i is zero on all green residues. The cost rises by one for each step away from the first green endpoint, and falls by one on approaching the other. If q_i is even, its peak has two adjacent residues of equal height.



Each dot represents an integer starting residue; the straight segments are only a convenient way to show the *discrete slopes* between consecutive residues. Adding tent functions adds their slopes, so the sum is piecewise linear with only $\mathcal{O}(n)$ changes of slope.

Wraparound at residue 0. Cut the cyclic sequence at 0 and regard it as a function on $0, 1, \dots, c$, with the value at c equal to that at 0. A tent crossing the cut merely becomes two pieces, one near 0 and one near c ; it still has only constantly many slope changes. For example, if $c = 10$ and the forbidden residues are $(8, 9, 0, 1)$, then the costs for residues $0, 1, \dots, 9$ are

$$(2, 1, 0, 0, 0, 0, 0, 0, 1, 2).$$

The endpoint value at 10 is again 2. Cutting at 0 therefore creates only a constant number of additional slope changes.

Sweep the slope changes. For each light, compute $f_i(0)$ and the discrete slope $f_i(x+1) - f_i(x)$ on its constant-slope pieces, splitting pieces at 0 and c if necessary. Record the (constantly many) positions where that slope changes. Sum all $f_i(0)$ to obtain $DP(0)$, and sum their first slopes to obtain the initial slope of DP . Sort the $\mathcal{O}(n)$ events. Between events the slope stays constant, so we can update the value by

$$DP(v) = DP(u) + (v - u) \cdot \text{current slope}$$

when moving from position u to the next event v . After processing an event, update the slope. On each straight piece the minimum occurs at an endpoint (or everywhere if the slope is zero), so checking the event positions and $0, c - 1$ suffices. The result is $\min_{0 \leq x < c} DP(x)$.

The complexity is $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ memory; use 64-bit integers for sums. For $c = 1$, every light is green and the answer is 0.

I. Improvised Handles

Consider two neighbouring boxes of *equal* height. If neither has a tab, neither can be the first of this pair to be removed: the other is still present and is not strictly shorter. Consequently, every adjacent equal-height pair must have at least one tab at an endpoint.

Partition the row into maximal segments of equal heights, with lengths $r_1, r_2, \dots$. Within a segment of length r , selecting tabbed boxes is exactly the task of covering every edge of a path of r vertices. Its minimum vertex cover has size $\lfloor r/2 \rfloor$: choose positions $2, 4, 6, \dots$, and for a lower bound consider the disjoint edges $(1, 2), (3, 4), \dots$.

These independent lower bounds are simultaneously achievable. Tab every second box in each maximal equal-height segment and remove all tabbed boxes first. Now no two remaining adjacent boxes have equal height. Remove all remaining boxes in *decreasing order of height*. When a box is to be removed, every taller neighbour has already gone, and every neighbour still present is strictly shorter. Hence it can be lifted without a tab.

The answer is therefore

$$\sum_{\text{maximal equal-height segments}} \left\lfloor \frac{r}{2} \right\rfloor.$$

A single scan computes it in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ extra memory.

J. Janitor's Cart

Think of the a_i statements at office i as separate objects arranged along the corridor. The greedy rule is: **start with the farthest uncollected object, and in the same trip collect up to $c - 1$ next-closest uncollected objects, moving towards the staircase.** Repeat until everything is collected. Equivalently, split all objects, listed from farthest to closest, into consecutive groups of at most c . Each group requires a trip whose farthest point is the distance of its first object.

Why is this optimal? Define the *suffix count*

$$S_i = \sum_{j=i}^n a_j = a_i + a_{i+1} + \dots + a_n.$$

This counts the objects in offices $i, i+1, \dots, n$, all of which must be carried across the interval between offices $i-1$ and i .

At least $\lceil S_i/c \rceil$ trips must reach office i , since each such trip can transport at most c of those S_i objects. Every trip reaching i traverses the one-meter interval between $i-1$ and i twice. Hence every solution has total distance at least

$$2 \sum_{i=1}^n \left\lceil \frac{S_i}{c} \right\rceil.$$

The greedy grouping attains this bound: exactly the first $\lceil S_i/c \rceil$ groups contain objects from office i or farther, so exactly that many trips reach i . Thus the greedy is optimal.

Skip repeated trips. There may be too many objects to simulate individually. Scan offices from n down to 1, maintaining **free**, the unused capacity in the last trip started at a farther office:

1. Use as much of **free** as possible to collect statements at the current office; no new trip is necessary for those objects.
2. If a statements remain at office i , start $q = \lceil a/c \rceil$ trips reaching i , and add $2iq$ to the answer. All but possibly the last are full; set **free** = $qc - a$.

If $a = 0$, do not change **free**: that space is still available for closer offices. Division groups potentially huge numbers of identical steps, so the algorithm is $\boxed{\mathcal{O}(n)}$ time and $\mathcal{O}(1)$ extra memory. Use 64-bit integers.

K. Kavala Olives

Let $k = \sum_{i=1}^n a_i$ be the number of olives. If $k = 0$, the answer is 0.

Preserving circular order. Imagine giving the olives distinct labels, arranged in clockwise order around the circle; olives on the same plate may be ordered arbitrarily. These labels need never overtake each other. Whenever we move an olive clockwise from a plate, choose the *last* olive in the group occupying that plate; whenever we move one counterclockwise, choose the *first*. This choice does not affect the visible moves, but preserves the circular ordering of the labels, even when several olives occupy one plate during the process.

Now imagine infinitely many copies of the circle laid out on an integer line, with corresponding plates n positions apart. List the initial positions of the k labeled olives in one turn:

$$p_1 \leq p_2 \leq \cdots \leq p_k, \quad 1 \leq p_i \leq n.$$

Follow each olive on the line, allowing its coordinate to become negative or exceed n when it crosses the boundary of the original circle. Write q_i for its final coordinate. Preserving the circular order means

$$\boxed{q_1 < q_2 < \cdots < q_k, \quad q_k - q_1 \leq n - 1.} \quad (4)$$

The strict inequalities hold because the final plates must be distinct. The span bound says that the k positions fit within one full turn; in particular, no two are equal modulo n .

Each olive makes at least $|p_i - q_i|$ moves, so any solution costs at least $\sum_i |p_i - q_i|$. Conversely, for any integer coordinates satisfying (4), we can simply move olive i from p_i to q_i along the line: temporary multiple occupancy is allowed, and the final residues modulo n are distinct. This costs exactly $\sum_i |p_i - q_i|$. Thus we only need to minimize this sum subject to (4).

Removing the gaps between destinations. Set

$$r_i = p_i - i, \quad x_i = q_i - i, \quad h = n - k.$$

Since the coordinates are integers, the constraints (4) become

$$\boxed{x_1 \leq x_2 \leq \cdots \leq x_k, \quad x_k - x_1 \leq h,} \quad (5)$$

and the cost becomes $\sum_i |r_i - x_i|$. Equivalently, the nondecreasing values x_i must all lie in *some* interval $[z, z + h]$ of length h . We will first ignore this interval, and then find its best position.

Nondecreasing values by merging median blocks. Consider the simpler problem

$$\min_{x_1 \leq \cdots \leq x_k} \sum_{i=1}^k |r_i - x_i|.$$

For a consecutive block B , if all its x_i equal a common number v , its contribution is

$$C_B(v) = \sum_{i \in B} |r_i - v|.$$

This is minimized at a median of the values in B . When there are two middle values, consistently choose the smaller one as the median m_B .

Start with each index as its own block. Whenever two consecutive blocks A, B have $m_A > m_B$, merge them and recompute the median of their union. Repeat until the block medians are nondecreasing. The resulting solution assigns to every position the median of its block.

Why is merging safe? We prove the following invariant: there exists an optimal nondecreasing solution which is constant within every current block. This is initially obvious. Suppose that in such a solution two neighbouring blocks A, B have values $u \leq v$, but their medians satisfy $m_A > m_B$. We may give both blocks the same value

$$w = \max(u, \min(m_A, v)).$$

Moving u up to w goes towards m_A (or leaves it unchanged), so C_A cannot increase. Moving v down to w likewise goes towards m_B (or leaves it unchanged), so C_B cannot increase. Moreover, $u \leq w \leq v$, so the surrounding blocks remain in order. Hence an optimum still exists after merging A and B , proving the invariant.

At termination, the block medians are nondecreasing, so assigning each block its median is feasible. It minimizes the cost of *every block separately*; by the invariant, some global optimum is constant within these blocks. Therefore our assignment is globally optimal. Denote it by x_i^* .

The algorithm is implemented with a stack of blocks: after adding a new singleton, repeatedly merge the last two blocks while their medians are decreasing. Maintain each block's values in an ordered multiset, together with its median. When merging, insert the elements of the smaller multiset into the larger one. Every element is transferred $\mathcal{O}(\log k)$ times, and every insertion takes $\mathcal{O}(\log k)$ time, giving $\mathcal{O}(k \log^2 k)$ total time.

Restricting to an interval. Fix an integer z and require all $x_i \in [z, z+h]$. The optimum is obtained simply by *clipping* the unrestricted solution:

$$x_i(z) = \max(z, \min(x_i^*, z+h)). \quad (6)$$

Indeed, the merging argument above remains valid with this extra restriction: if u, v are in the interval, their replacement $w \in [u, v]$ is also in the interval. Thus the *same* sequence of block merges preserves an optimum even for the interval-restricted problem. Once the blocks have increasing medians, the best common value for each block in $[z, z+h]$ is its median clipped to that interval. These clipped medians remain nondecreasing, so choosing them for all blocks simultaneously is optimal. This proves (6).

Finding the best interval. It remains to minimize

$$F(z) = \sum_{i=1}^k |r_i - \max(z, \min(x_i^*, z+h))|.$$

Each summand is piecewise linear as a function of z . For a fixed $v = x_i^*$, the clipped value is $z+h$ when $z < v-h$, equals v when $v-h \leq z \leq v$, and is z when $z > v$. After taking the absolute difference from r_i , there are at most three points where the slope changes.

Generate these $\mathcal{O}(k)$ slope-change events, sort them by z , and sweep from left to right, maintaining the current value and slope of F . Between events the function is linear, so an optimal integer z occurs at an event position (or anywhere in a flat interval). This takes $\mathcal{O}(k \log k)$ time.

Overall the algorithm uses $\boxed{\mathcal{O}(n \log^2 n)}$ time and $\mathcal{O}(n)$ memory. Use 64-bit integers for costs.

L. Lost Cargo

Query exactly the cells whose row and column numbers are both even:

$$(2i, 2j), \quad 1 \leq 2i \leq n, \quad 1 \leq 2j \leq m.$$

There are $\lfloor n/2 \rfloor \lfloor m/2 \rfloor \leq \lfloor nm/4 \rfloor$ such cells. No two are adjacent, so each reply identifies its whole domino, consisting of the queried cell and one unqueried neighbour. Mark both cells as *covered*.

After these queries, the unknown dominoes form a perfect matching of the graph whose vertices are the *uncovered* cells and whose edges connect side-adjacent uncovered cells. We show that this graph is a forest. A forest has at most one perfect matching: each leaf must be paired with its only neighbour, and removing that pair leaves the same task on a smaller forest. Repeating this rule reconstructs all remaining dominoes.

First, an auxiliary directed graph. Its vertices are the queried cells, considered at their *cell centres* $(2i, 2j)$ in integer coordinates. A queried cell c is paired with an adjacent unqueried cell u . If

there is another queried cell c' directly beyond u in the same direction, draw $c \rightarrow c'$. Thus the three centres c, u, c' lie on a horizontal or vertical segment of length 2, and u is its midpoint. When c' is outside the board, draw no arrow. Every queried cell has at most one outgoing arrow.

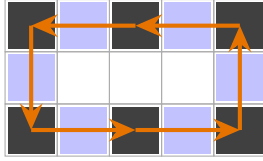
There is no directed cycle. Suppose there were a directed simple cycle with s edges. Draw the polygon through the *centres of its queried cells*, joining consecutive centres by straight segments of length 2. This is a simple axis-parallel lattice polygon. Each vertex lies at two even integer coordinates. Each edge contains exactly one further integer point in its interior, namely the midpoint u , the centre of the cell paired with the queried cell at the tail of that arrow.

Consequently, *all boundary cell centres belong to dominoes entirely on the boundary*: the s queried centres are paired with the s edge-midpoint centres. No other domino can use a boundary cell. Since grid-adjacent centres are one integer step apart and the polygon boundary follows grid lines, any domino with a centre strictly inside the polygon and a centre outside would have to pass through a boundary centre. Thus all strictly interior cell centres must be tiled amongst themselves, so their number I must be even.

But these centre coordinates make Pick's formula apply directly. Write A for the polygon's area and B for its number of boundary lattice points. Because the vertices are even-even, scaling down by 2 gives a lattice polygon, so A is a multiple of 4. Each edge of length 2 contributes two boundary lattice steps, hence $B = 2s$. The queried-cell lattice is bipartite, so every cycle has even length s . Pick's formula gives

$$I = A - \frac{B}{2} + 1 = A - s + 1 \quad \text{odd,}$$

a contradiction.



The polygon follows the arrows through **cell centres**.

Dark cells are queried; light-blue cells are their domino partners.

Every boundary cell is already paired along the polygon.

Here $A = 8$, $B = 12$, and $s = 6$, so Pick's formula gives $I = A - B/2 + 1 = 3$ interior cell centres: an odd number.

The uncovered cells also contain no cycle. Suppose their adjacency graph had a simple cycle. It encloses at least one queried cell: the cycle is a grid polygon containing at least one unit grid square, and each such square has an even-even corner. The cycle itself contains only uncovered, hence unqueried, cells, so such a corner is strictly inside.

Every queried cell c strictly inside the cycle is paired with a cell u also strictly inside: otherwise the straight one-step segment cu would cross the polygon boundary, requiring a boundary cell to be covered. Its arrow $c \rightarrow c'$, where defined, also stays inside the cycle, since the two-step segment from c to c' passes through u , and cannot cross the integer-coordinate polygon boundary without meeting a boundary cell centre. Moreover c' exists: a cell strictly inside the cycle cannot be close enough to the outside edge of the board for this two-step continuation to leave the board.

There are finitely many queried cells inside the cycle, and every one has an outgoing arrow to another one inside. Following arrows must produce a directed cycle, contradicting what we have proved. Therefore the uncovered graph is a forest.

Recover its unique perfect matching by repeatedly removing leaves, and fill in the corresponding U/D/L/R characters for each domino. With the already recorded dominoes this reconstructs the entire tiling. All work takes $\mathcal{O}(nm)$ time and memory. Flush after every query and after the final answer.

M. Museum Tickets

Either buy one group ticket for b euros or buy n individual tickets for na euros. A group ticket already admits everybody, so there is no advantage in combining these two options. The answer is simply

$$\boxed{\min(na, b)}.$$

This takes $\mathcal{O}(1)$ time and memory.