

Problem A. DeepTreek

Define:

- d_u to be the depth of u .
- S_u to be the set of vertices in the subtree of u .
- $g_u = \max_{w \in S_u} d_w - d_u$.
- $h_u = \max_{w \notin S_u} d_w$.

Consider a valid pair (u, v) . With the above definition, we can write $f(u, v)$ as $f(u, v) = \max(h_u, g_u + d_v) = h_u + \max(0, d_v - (h_u - g_u))$.

Thus:

$$\begin{aligned}
 \sum_{v \notin S_u} f(u, v) &= \sum_{v \notin S_u} h_u + \sum_{v \notin S_u} \max(0, d_v - (h_u - g_u)) \\
 &= (n - |S_u|) \cdot h_u + \sum_{v \notin S_u, d_v > h_u - g_u} (d_v - (h_u - g_u)) \\
 &= (n - |S_u|) \cdot h_u + \sum_{v \notin S_u, d_v > h_u - g_u} d_v - \sum_{v \notin S_u, d_v > h_u - g_u} (h_u - g_u) \\
 &= (n - |S_u|) \cdot h_u + \sum_{v \notin S_u, d_v > h_u - g_u} d_v - (h_u - g_u) \cdot \sum_{v \notin S_u, d_v > h_u - g_u} 1
 \end{aligned}$$

All of d_u , S_u , g_u , and h_u can be precomputed in $O(n)$ via depth-first search. The last two terms of the expression represent the sum of depths and count of all vertices outside of a subtree, with depth larger than a threshold. This can be calculated efficiently by dsu on tree and fenwick tree. Overall time complexity $O(n \log^2 n)$.

Problem B. Odd Cycle

Consider a naive way of solving this problem: for every vertex, we start a dfs from that vertex. If we walk an odd number of steps and come back to the starting vertex, then it is contained in an odd cycle.

This inspires us to split each vertex u into two: u_0 and u_1 , and create a new graph G' . For each edge $u \rightarrow v$, we will add two edges: $u_0 \rightarrow v_1$ and $u_1 \rightarrow v_0$. u is contained in at least one odd cycle if and only if there is a path starting from u_0 and ending at u_1 .

However, if u_1 is reachable from u_0 , u_0 will also be reachable from u_1 . This means that u_0 and u_1 will be in the same strongly connected component. Simply find all strongly connected components of G' , and for each vertex u , check whether u_0 and u_1 are in the same SCC. Overall complexity $O(n + m)$.

Problem C. Count Cubes

Firstly, for each row, either both x -view and y -view contain a cell or neither of them contains a cell. Also, a cell is either supported by the cell below it, or is already on the ground. If any of these are violated, the answer is -1 -1.

Here is the solution: for a row i , let the number of cells in the x -view be x_i , and the number of cells in the y -view be y_i . Then, the maximum number of cubes is:

$$\sum_{i=1}^c x_i y_i$$

The calculation of the minimum number of cubes is slightly more complicated. Consider two rows i and $i + 1$, with row i on top of row $i + 1$. Let $dx = x_{i+1} - x_i$, and $dy = y_{i+1} - y_i$. In addition to the number of cubes placed in row i , you need to place $\max(dx, dy)$ cubes in row $i + 1$ to account for the extra cells. Thus, if the minimum number of cells placed in row i is r_i , we obtain:

$$r_{i+1} = r_i + \max(x_{i+1} - x_i, y_{i+1} - y_i)$$

The proofs for these analyses are left as an exercise for the participants (definitely not because the author is lazy). Overall complexity $O(c \cdot (a + b))$.

Problem D. Xor And Mul

The answer is simply $n + m + 1$, since at least one of x and y must be 0.

Proof

Let a contain all common bits of x and y , b contain all bits only in x , and c contain all bits only in y . For example, if $(x, y) = (1011_2, 1100_2)$, then $(a, b, c) = (1000_2, 0011_2, 0100_2)$. We can then deduce the equalities:

- $x = a + b$.
- $y = a + c$.
- $x \oplus y = b + c$.
- $x \& y = a$.

Substituting them into the given equality gives:

$$\begin{aligned}(x \& y) \cdot (x \oplus y) &= x \cdot y \\ a \cdot (b + c) &= (a + b) \cdot (a + c) \\ a \cdot (b + c) &= a^2 + b \cdot c + a \cdot (b + c) \\ a^2 + b \cdot c &= 0\end{aligned}$$

$a^2 = 0$ implies $a = 0$, which then means $b = x$ and $c = y$. From $b \cdot c = 0$, we then obtain $x \cdot y = 0$, which means that at least one of x or y must be 0.

Problem E. MiniC

This problem asks you to solve the next-smaller-element problem in $O(n^2)$, but without using any control flow, loop, or arithmetic operations except for incrementing and decrementing integers.

A substitute for loops will be recursions. However, without control flow, it seems hard to end the recursion (there is no `if (condition) return;`).

Here is when function arrays become useful. Consider the following piece of C code:

```
#include <stdio.h>

typedef void (*func)();

func actions[2];
int loop;
int masks[1001];
void do_nothing() {

}
void work() {
    printf("%d\n", loop);
}
void for_each() {
    work();
    int mask = masks[loop];
    func action = actions[mask];
    loop++;
    action();
}

int main() {
    int n;
    scanf("%d", &n);
    actions[1] = do_nothing;
    actions[0] = for_each;
    n--;
    masks[n] = 1;
    n++;
    for_each();
}
```

This piece of code reads an integer n and prints $0, 1, 2, \dots, n - 1$ separated by spaces.

Observe how the for loop is achieved. When `loop` is not $n - 1$, `mask` = 0, and thus `action` = `for_each`. This will continue the recursion. Otherwise, `mask` = 1, and thus `action` = `do_nothing`, which ends the recursion.

There is another difficulty, which is that comparisons are not supported in MiniC. How do we circumvent this? Instead of following the problem description and calculating $f(i)$ in increasing order of i , we can calculate $f(i)$ in increasing order of p_i . The C code to achieve this is shown below:

```
#include <stdio.h>

int p[1000];
int b[1000];
int v[1001];
int f[1001];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &p[i]);
        b[p[i]] = i;
    }
    v[n] = 1;
    for (int i = 0; i < n; i++) {
        int idx = b[i];
        for (int j = idx; j <= n; j++) {
            if (v[j]) {
                f[idx] = j;
                break;
            }
        }
        v[idx] = 1;
    }
    for (int i = 0; i < n; i++) {
        printf("%d\n", f[i]);
    }
}
```

It is then not hard to translate this to MiniC, using the for loop trick mentioned above.

Problem F. Random Walk

Let $(X_k)_{k \geq 0}$ be a random walk on $\mathbb{Z}^2$. That is:

- $X_0 = (0, 0)$
- $X_{k+1} - X_k \in \{(0, 1), (0, -1), (1, 0), (-1, 0)\}$

Let $S_n = \{X_0, X_1, \dots, X_n\}$, and $R_n = |S_n|$. We need to find $\mathbb{E}[R_n]$.

Let the return probability $a_n = \Pr[X_n = (0, 0)]$, first return distribution $f_n = \Pr[\text{the first time returning to } (0, 0) \text{ occurs at } n]$, and $s_n = \Pr[X_1 \neq (0, 0), X_2 \neq (0, 0), \dots, X_n \neq (0, 0)]$.

Lemma 1. $a_{2m} = \frac{\binom{2m}{m}^2}{4^{2m}}$, $a_{2m+1} = 0$

Proof.

One obvious observation is that for odd n , $a_n = 0$. Then consider under what circumstances a random walk would return to $(0, 0)$ after the n -th step: there must be an equal number of $(0, 1)$ steps and $(0, -1)$ steps, as well as an equal number of $(1, 0)$ steps and $(-1, 0)$ steps. For simplicity, let $n = 2m$.

Let the number of $(0, 1)$ steps be i . We can write a_{2m} as follows:

$$\begin{aligned}
4^{2m} \cdot a_{2m} &= \sum_{i=0}^m \binom{2m}{i} \binom{2m-i}{i} \binom{2m-2i}{m-i} \\
&= \sum_{i=0}^m \left(\frac{(2m)!}{i!(2m-i)!} \right) \left(\frac{(2m-i)!}{i!(2m-2i)!} \right) \left(\frac{(2m-2i)!}{(m-i)!(m-i)!} \right) \\
&= \sum_{i=0}^m \frac{(2m)!}{(i!)^2((m-i)!)^2} \\
&= \sum_{i=0}^m \frac{(2m)!}{(m!)^2} \left(\frac{m!}{i!(m-i)!} \right)^2 \\
&= \binom{2m}{m} \sum_{i=0}^m \binom{m}{i}^2 \\
&= \binom{2m}{m}^2
\end{aligned}$$

where the last step uses the Vandermonde Convolution trick.

Lemma 2. $s_n = 1 - \sum_{i=1}^n f_i$

Proof.

A random walk can be classified into $n+1$ categories: first return to origin at 1 (f_1), first return to origin at 2 (f_2), ..., first return to origin at n (f_n), first return to origin after n (s_n).

Lemma 3. $a_n = \sum_{i=1}^n f_i a_{n-i}$

Proof.

Consider a random walk that returns to the origin at n . Let the first return to origin be at i . Then, the path from X_i to X_n forms a fresh new random walk of length $n-i$.

Lemma 4. $\mathbb{E}[R_n] = \sum_{i=0}^n s_i$

Proof.

Let $t_i = \Pr[X_i \notin \{X_0, X_2, \dots, X_{i-1}\}]$, then $\mathbb{E}[R_n] = \sum_{i=0}^n t_i$. Consider X_i for a fixed i . Let $Y_j = X_i - X_{i-j}$ be the reversed walk from X_i to the origin. Since the probability of going in each of the four directions is the same, $(Y_j)_{0 \leq j \leq i}$ is also a random walk of length i . Thus, $X_i \notin \{X_0, X_1, \dots, X_{i-1}\}$ is equivalent to Y_j not returning to the origin, which has a probability of s_i .

With all the above, we are ready to derive the final formula.

Observe that Lemma 3 can be written in the following generating function form:

$$\begin{aligned}
A(z) &= 1 + A(z)F(z) \\
A(z)(1 - F(z)) &= 1 \\
1 - F(z) &= \frac{1}{A(z)}
\end{aligned}$$

Writing Lemma 2 in generating function form as well:

$$\begin{aligned}
S(z) &= \sum_{i \geq 0} s_i z^i \\
&= \sum_{i \geq 0} \left(1 - \sum_{j=1}^i f_j \right) z^i \\
&= \sum_{i \geq 0} z^i - \sum_{j \geq 1} f_j \sum_{i \geq j} z^i \\
&= \frac{1}{1-z} - \frac{F(z)}{1-z} \\
&= \frac{1-F(z)}{1-z} \\
&= \frac{1}{A(z)(1-z)}
\end{aligned}$$

which can be calculated using inverse of formal power series.

After finding $S(z)$, we can just take the prefix sum of its coefficients to calculate $\mathbb{E}[R_n]$ for all n in $O(n \log n)$.

Problem G. HDZ's Legendary Problem

For each query, we need to enumerate all distribution schemes. There are $O(n^3)$ total enemy configurations. When processing a specific enemy configuration, simulating the battle round-by-round is unwise. The complexity of this naive approach is $O(qn^3)$.

Notice that the defense-related attributes are Defense, Health, and Speed, while the attack-related attributes are only Attack and Speed. When enumerating the uncertain enemy, we can first fix their offensive attributes (Attack and Speed points).

Once the enemy's offense is fixed, we can determine exactly how many rounds (*round*) the Hero can survive. The remaining points must be distributed between the enemy's Health and Defense to determine how many rounds the Monster can survive. We then compare the Monster's survival rounds with the Hero's survival rounds (*round*).

The problem now transforms into finding: Given the Monster's initial Health, Defense, remaining enhancement points n , the damage taken per round (*damage*), and a duration of r rounds, how many distribution methods allow the Monster to stay alive?

Consider how many points are added to Health. We discover this problem naturally splits into two intervals:

- Interval $[0, C]$: In this range, the Defense is low enough that every attack deals at least 1 damage (or the damage formula hasn't hit the floor of 1 yet).
- Interval $[C + 1, n]$: In this range, Defense becomes high. The latter half of this interval is strictly worse than the former half for survival because if $round \geq 1$, adding 1 point to Defense reduces total damage taken by r (since it reduces damage by 1 per round for r rounds), whereas adding 1 point to Health only allows taking 1 more damage total.

We can use binary search (not guaranteed to pass, but likely possible) or an $O(1)$ mathematical formula to directly calculate the "death position" (the threshold of points where the monster dies), and thereby calculate the contribution of the second type of interval.

Let us denote this solution function as $f(damage, r, n)$.

- $f(damage, round - 1, n)$ represents the number of scenarios where the Hero wins.

- $f(\text{damage}, \text{round}, n) - f(\text{damage}, \text{round} - 1, n)$ represents the number of draws.
- The rest are losses.

This allows us to only enumerate the Monster's points assigned to Attack and Speed, and calculate the answer internally in $O(1)$. The complexity becomes $O(qn^2)$.

Can we speed up the enumeration further? Yes! Observe that when we enumerate Speed in the outer loop and Enemy Attack in the inner loop, the number of rounds (*round*) the enemy takes to kill us remains the same for many values.

This is because the Speed determines the damage multiplier (*fac*). Let $DPR = fac \times atk_{opponent} - def_{me}$. The formula for survival rounds is $round = \lceil \frac{hp_{me}}{DPR} \rceil$. From number theory (harmonic lemma), we know that *round* takes at most $2\sqrt{hp_{me}}$ distinct values.

If we enumerate Speed in the outer loop and iterate over the distinct values of *round* in the inner loop, the loop complexity drops to $O(n\sqrt{skill})$. However, inside this loop, we find that the available enhancement points for the remaining attributes are no longer a fixed value, but a continuous range. To calculate the contribution of a range, we can precompute all values of $f(\text{damage}, r, n)$ and take the prefix sum over the last dimension. This allows direct calculation of a range's contribution.

The total complexity becomes $O((n + skill)skill^2 + qn\sqrt{skill})$.

In fact, the outer enumeration of Speed can also be optimized. Enumerating Speed is essentially enumerating the pair of multipliers: (My damage multiplier against opponent, Opponent's damage multiplier against me).

- When $speed_{me} \geq speed_{opponent}$, similar to the optimization for Attack above, there are at most $O(\sqrt{skill})$ possible multiplier pairs.
- When $speed_{me} < speed_{opponent}$, the worst case is $d = D$ (attribute ranges match). In this case, the enumerated multiplier pairs are $(1, 1), (1, 2) \dots (1, n)$, totaling n pairs. However, we notice that for the inner enumeration of *round*, this relates to the harmonic series. Therefore, the total enumeration cost in this case is $O(n \log n)$.

So, if we enumerate the multiplier pairs first, and then enumerate *round*, the worst-case enumeration cost is only $O(n \log n)$.

However, the contribution calculation inside the innermost loop becomes very complex.

This contribution has a classic property and solution: the weight of the contribution of each position first rises in steps $(1, 2, 3, \dots)$, then stays flat, then descends in steps back to 1. This can be calculated in $O(1)$ using a weighted prefix sum, i.e., $prew_i = a_0 + 2a_1 + \dots + (i+1)a_i$, combined with a standard prefix sum.

With this, we optimize the solution to $O((n + skill)skill^2 + qn \log skill)$, which passes all test cases.

Problem H. Distinct Substrings

For a string $s \in \Sigma^+$, let $n = |s|$. Define a *run* as a tuple of three integers (p, i, j) , such that:

- The minimum period of $s[i..j]$ is p ;
- $i = 1$ or $s[i - 1] \neq s[i - 1 + p]$;
- $j = n$ or $s[j + 1] \neq s[j + 1 - p]$.
- $j - i + 1 \geq 2p$.

It is well-known that there are at most $O(|s|)$ distinct runs for any string s , and we can compute all of them in $O(|s| \log |s|)$.

For simplicity, we will not provide detailed proofs for the lemmas.

Lemma 1. For a fixed position k , the number of non-distinct substrings covering k is $k \cdot (n - k + 1)$.

Lemma 2. For a fixed position k , consider two equal strings $s[l..r]$ and $s[l + x..r + x]$ that both cover k . Then $s[l..r]$ and $s[l + x..r + x]$ must belong to a run with period $p < r - l + 1$.

Lemma 3. For a fixed position k and a string t . Consider the set $\text{occ}_s(t, k) = \{j \in \text{occ}_s(t) \mid j \leq k \leq j + |t| - 1\}$. If $|\text{occ}_s(t, k)| \geq 2$, then there exists a run with period $p < |t|$, such that the run contains all occurrences of t in $\text{occ}_s(t, k)$.

Lemma 4. Fix a position k . Let t be a substring of s such that $\text{occ}_s(t, i) = \{a_0 < a_1 < \dots < a_{m-1}\}$, $m \geq 2$.

By Lemma 3, these starting positions form an arithmetic progression $a_j = a_0 + jp$ for some $p < |t|$, where p is the minimal period of the run containing these occurrences.

Then t contributes exactly $m - 1$ duplicates to the multiset of substrings covering k , and these duplicates can be bijectively charged to the adjacent pairs $(a_0, a_1), (a_1, a_2), \dots, (a_{m-2}, a_{m-1})$.

Lemma 5. Let $\rho = (p, i, j)$ be a run of s , and let k be a position. Define $d(\rho, k)$ as the number of duplicates charged to run ρ at position k .

$$\text{Then } d(\rho, k) = \begin{cases} 0, & i \leq k \leq i + p - 1, \\ (k - i - p + 1)(j - p + 1 - k), & i + p \leq k \leq j - p, \\ 0, & j - p + 1 \leq k \leq j. \end{cases}$$

Lemma 6. Let s be a string of length n . For each position k , let $f_s(k)$ denote the number of distinct substrings of s that cover k . Then

$$f_s(k) = k(n - k + 1) - \sum_{\rho} d(\rho, k).$$

Lemma 7. For any run $\rho = (p, i, j)$ and position k , the difference array of $d(\rho, i + p), d(\rho, i + p + 1), \dots, d(\rho, j - p)$ forms an arithmetic progression with common difference -2 .

With the above lemmas, we can compute the contribution of each run to the answer separately. Overall complexity $O(n \log n)$. There is a solution using suffix automaton, but unfortunately the constant factor might be too big to fit in the time limit.

Problem I. Operating System

This phenomenon is known as Belady's Paradox in Operating Systems.

Firstly, when $m < k + 2$ or $k \leq 2$, there is no solution. The first condition is easier to see. The second condition is left as an exercise for the participants.

There are many ways to construct an answer; here is one of them:

Let $a = [1, 2, \dots, k + 1]$, $b = [1, 2, \dots, k - 1]$, and $c = [k + 2]$. The sequence $a + b + c + a + c$ works as a solution. Participants can verify its correctness by themselves.

Problem J. Someone's Favourite Problem

Let's call a vertex special if its in-degree is $n - 1$ and out-degree is 0.

Firstly, observe that there can be at most one special vertex.

Fix some pair of vertices (u, v) :

- If there exists an edge from u to v , it means that u is not special.
- Otherwise, v is not special.

Since each query eliminates at least one candidate vertex, we can use $n - 1$ queries to eliminate $n - 1$ vertices, leaving only one candidate vertex. Then, we use $2n - 2$ queries to determine its in-degree and out-degree. Total number of queries will be $3n - 3$.

Problem K. One Line

This is a somewhat well-known problem in mathematics. The largest size of S for a fixed n is still an open problem.

The original proof can be found at

<https://www.sciencedirect.com/science/article/pii/0097316575900436?via%3Dihub>

We first show that for every $p \geq 3$ be an odd prime, we can construct a set of at least $3p - 3$ points for $n = 2p$.

Let $p \geq 3$ be an odd prime and fix $c \in \mathbb{F}_p^\times$. Define the **modular hyperbola**

$$H(c, p) = \{(x, y) \in \mathbb{Z}^2 \mid xy \equiv c \pmod{p}\}.$$

A lattice point (x, y) is called **normal** if $p \nmid x$ and $p \nmid y$.

Two lattice points are **congruent mod p** if their coordinates are congruent componentwise: $(x_1, y_1) \equiv (x_2, y_2) \pmod{p}$ if and only if $x_1 \equiv x_2 \pmod{p}$ and $y_1 \equiv y_2 \pmod{p}$.

Lemma 1. Let ℓ be any line. Then $\ell \cap H(c, p)$ is contained in at most two congruence classes mod p .

Proof.

Write ℓ in integer form $ux + vy + w = 0$ with integers u, v, w and $\gcd(u, v) = 1$.

- If $v \equiv 0 \pmod{p}$, then $u \not\equiv 0 \pmod{p}$ (since $\gcd(u, v) = 1$), hence $ux + w \equiv 0 \pmod{p}$ forces a unique residue $x \equiv x_0 \pmod{p}$. On $H(c, p)$ we additionally have $x_0 y \equiv c \pmod{p}$, which forces a unique $y \pmod{p}$ (since $x_0 \not\equiv 0$). So $\ell \cap H(c, p)$ lies in at most one congruence class.
- If $v \not\equiv 0 \pmod{p}$, then we can solve $y \equiv \alpha x + \beta \pmod{p}$ where $\alpha \equiv -uv^{-1}$ and $\beta \equiv -wv^{-1}$. Substituting into $xy \equiv c$ gives

$$x(\alpha x + \beta) \equiv c \pmod{p} \iff \alpha x^2 + \beta x - c \equiv 0 \pmod{p},$$

a quadratic congruence over the field $\mathbb{F}_p$, hence it has at most two solutions for $x \pmod{p}$. Each such x determines a unique $y \pmod{p}$. Therefore $\ell \cap H(c, p)$ lies in at most two congruence classes.

Corollary 1. If a line contains at least three points of $H(c, p)$, then at least two of them are congruent mod p .

Proof. By pigeonhole principle.

Let $h = \frac{p-1}{2}$ and define the window

$$G = \{(x, y) \in \mathbb{Z}^2 : -h \leq x \leq 2p - h - 1, 0 \leq y \leq 2p - 1\}.$$

This is a $2p \times 2p$ grid. Fix a **normal** congruence class (a, b) with $a, b \in \{1, 2, \dots, p-1\}$. (a, b) has exactly four representative points in G . These four points are the vertices of an axis-aligned $p \times p$ square. Thus:

Lemma 2. Within one congruence class in G :

- No three points are collinear;
- Any line through two congruent points has slope in $\{0, \infty, 1, -1\}$ (horizontal, vertical, or a diagonal of the square).

Partition the residue box $\{1, \dots, p-1\}^2$ into four quadrants using h :

- "small" means $\leq h$, "big" means $\geq h + 1$.
- Type C : (a, b) small-small.
- Type D : (a, b) big-small.
- Type A : (a, b) small-big.
- Type B : (a, b) big-big.

For integers r, s , define coordinate intervals

$$X_r^{\text{sm}} = \{rp + 1, \dots, rp + h\}, \quad X_r^{\text{bg}} = \{rp + h + 1, \dots, rp + p - 1\},$$

$$Y_s^{\text{sm}} = \{sp + 1, \dots, sp + h\}, \quad Y_s^{\text{bg}} = \{sp + h + 1, \dots, sp + p - 1\}.$$

Define blocks (rectangles):

$$C_{r,s} = X_r^{\text{sm}} \times Y_s^{\text{sm}}, \quad D_{r,s} = X_r^{\text{bg}} \times Y_s^{\text{sm}}, \quad A_{r,s} = X_r^{\text{sm}} \times Y_s^{\text{bg}}, \quad B_{r,s} = X_r^{\text{bg}} \times Y_s^{\text{bg}}.$$

(All points in these blocks are normal, since neither coordinate is a multiple of p .)

Now define the **selection region** (12 blocks) by

$$\begin{aligned} T_2 = & (A_{0,1} \cup A_{1,0} \cup A_{1,1}) \\ & \cup (B_{-1,0} \cup B_{-1,1} \cup B_{0,1}) \\ & \cup (C_{0,0} \cup C_{1,0} \cup C_{1,1}) \\ & \cup (D_{-1,0} \cup D_{-1,1} \cup D_{0,0}). \end{aligned}$$

Finally define the point set

$$S = S_2(c, p) := H(c, p) \cap T_2 \subseteq G.$$

Lemma 3. There are $p - 1$ normal residue classes on $H(c, p)$.

Over $\mathbb{F}_p^\times$, for each $a \in \mathbb{F}_p^\times$ there is a unique $b \in \mathbb{F}_p^\times$ with $ab = c$, namely $b = ca^{-1}$. Thus $H(c, p)$ has exactly $p - 1$ normal congruence classes.

Lemma 4. Each normal class contributes exactly 3 points to S .

Fix a normal class with base representative $(a, b) \in \{1, \dots, p - 1\}^2$ and let its type be A, B, C , or D .

Inside G , the class has four representatives, one in each of the four blocks of that type with indices:

- if $a \leq h$ then $r \in \{0, 1\}$, else $r \in \{-1, 0\}$;
- always $s \in \{0, 1\}$.

For each type, T_2 contains **exactly three** of these four blocks and omits exactly one:

- type A : omits $A_{0,0}$;
- type B : omits $B_{0,0}$;
- type C : omits $C_{0,1}$;
- type D : omits $D_{0,1}$.

Therefore each normal class contributes exactly three points to S .

Combining Lemmas 3 and 4 gives $|S| = 3(p-1)$.

Lemma 5.

Fix $\tau \in \mathbb{F}_p$ and consider the line $y = x + \tau$ over $\mathbb{F}_p$. Points of $H(c, p)$ on this line correspond to solutions of

$$x(x + \tau) = c \iff x^2 + \tau x - c = 0 \quad \text{in } \mathbb{F}_p.$$

If this quadratic has two distinct roots $x_1 \neq x_2$, and we set $y_i = x_i + \tau$, then

$$(x_2, y_2) \equiv (-y_1, -x_1) \pmod{p}.$$

Proof. By Vieta, $x_1 + x_2 = -\tau$. Hence $x_2 = -\tau - x_1 = -(x_1 + \tau) = -y_1$. Then $y_2 = x_2 + \tau = -y_1 + \tau = -x_1$. So $(x_2, y_2) = (-y_1, -x_1)$ in $\mathbb{F}_p^2$.

On base representatives in $\{1, \dots, p-1\}^2$, the map $(a, b) \mapsto (-b, -a) \equiv (p-b, p-a)$:

- swaps $B \leftrightarrow C$ (big-big $\leftrightarrow$ small-small),
- fixes A and fixes D .

Also note it preserves the difference:

$$(p-a) - (p-b) = b-a,$$

so for a partner pair (a, b) and $(a', b') = (p-b, p-a)$ we have $b' - a' = b - a$.

Lemma 6. Fix $\tau \in \mathbb{F}_p$ and consider the line $y = -x + \tau$ over $\mathbb{F}_p$. Points of $H(c, p)$ on this line correspond to solutions of

$$x(\tau - x) = c \iff x^2 - \tau x + c = 0 \quad \text{in } \mathbb{F}_p.$$

If it has two distinct roots $x_1 \neq x_2$, and we set $y_i = -x_i + \tau$, then

$$(x_2, y_2) \equiv (y_1, x_1) \pmod{p}.$$

Proof. By Vieta, $x_1 + x_2 = \tau$. Then $y_1 = \tau - x_1 = x_2$ and $y_2 = \tau - x_2 = x_1$. $\square$

On base representatives, the map $(a, b) \mapsto (b, a)$:

- swaps $A \leftrightarrow D$,
- fixes B and fixes C .

It preserves the sum: $a' + b' = a + b$.

—

Theorem. No line contains three points of S .

Proof.

Assume for contradiction that a line ℓ contains three distinct points of $S \subseteq H(c, p)$.

By Lemma 1, $\ell \cap H(c, p)$ lies in at most two residue classes, so among the three points at least two are congruent mod p . Fix two congruent points of S on ℓ ; they belong to some normal residue class with base representative (a, b) .

By Lemma 2, the slope of ℓ must be in $\{0, \infty, 1, -1\}$. We treat cases.

Case 1: slope 0 or ∞

If ℓ is horizontal or vertical, then the proof of Lemma 1 shows $\ell \cap H(c, p)$ lies in at most one residue class. Hence all three points on ℓ would be congruent.

But within one residue class, the four representatives form a $p \times p$ square in G , and a horizontal or vertical line can meet the four corners in at most two points. In particular, it cannot contain three points of S . Contradiction.

Case 2: slope +1

Write ℓ as $y = x + t$ (over $\mathbb{R}$). For any integer point (x, y) on ℓ , the invariant $y - x$ is constant.

The two congruent points from the same residue class must be the two corners on a slope +1 diagonal of its congruence square.

Equivalently, for those two points the tile indices (r, s) satisfy the same value of $s - r$:

- If $a \leq h$ (so $r \in \{0, 1\}$), the +1 diagonal uses $(r, s) = (0, 0)$ and $(1, 1)$, hence $s - r = 0$.
- If $a > h$ (so $r \in \{-1, 0\}$), the +1 diagonal uses $(r, s) = (-1, 0)$ and $(0, 1)$, hence $s - r = 1$.

Now observe that in $S = H(c, p) \cap T_2$, not every diagonal of every square is present, because we omitted one block per type:

- For type A we omitted $A_{0,0}$, so a +1 diagonal requiring $(0, 0)$ cannot be fully present in S .
- For type D we omitted $D_{0,1}$, so a +1 diagonal requiring $(0, 1)$ cannot be fully present in S .

Therefore, **the only possible types that can supply two congruent points on a +1 diagonal inside S are B and C :**

- type B can supply the congruent pair in blocks $B_{-1,0}$ and $B_{0,1}$ (this is the $s - r = 1$ diagonal),
- type C can supply the congruent pair in blocks $C_{0,0}$ and $C_{1,1}$ (this is the $s - r = 0$ diagonal).

We handle these two subcases and show a third point cannot exist.

Subcase 2a: the congruent pair comes from type B (so $s - r = 1$).

Let (a, b) be the base representative (so $a, b \in \{h + 1, \dots, p - 1\}$). The two congruent points in S are in tiles $(-1, 0)$ and $(0, 1)$, hence they both satisfy

$$y - x = (b - a) + (s - r)p = (b - a) + p.$$

If ℓ were to contain a third point of $H(c, p)$, it must lie in the **other** residue class on ℓ (since within one residue class, a +1 diagonal contains exactly two corners). By Lemma 5, the other residue class (if it exists) has base representative

$$(a', b') = (p - b, p - a),$$

which is type C , and satisfies $b' - a' = b - a$.

Any representative of this partner class in tile (r, s) has

$$y - x = (b' - a') + (s - r)p = (b - a) + (s - r)p.$$

To lie on the same line ℓ (same $y - x$), we must have

$$(b - a) + (s - r)p = (b - a) + p \iff s - r = 1.$$

For type C , the available tile indices are $(r, s) \in \{0, 1\} \times \{0, 1\}$, and the only solution of $s - r = 1$ is $(r, s) = (0, 1)$.

Thus the **only** candidate point of the partner class on ℓ inside G lies in block $C_{0,1}$.

But $C_{0,1}$ is precisely the block omitted from T_2 , hence S contains no point of the partner class on ℓ .

So ℓ contains at most the two congruent points from the first class, not three.

Subcase 2b: the congruent pair comes from type C (so $s - r = 0$).

Now (a, b) has $a, b \in \{1, \dots, h\}$ and the two congruent points are in tiles $(0, 0)$ and $(1, 1)$, hence

$$y - x = (b - a) + (s - r)p = b - a$$

(because here $s - r = 0$).

If there is a second residue class on ℓ , by Lemma 5 its base representative is again $(a', b') = (p - b, p - a)$, which is type B and satisfies $b' - a' = b - a$. A representative in tile (r, s) has $y - x = (b - a) + (s - r)p$, so lying on ℓ forces $s - r = 0$.

For type B , the available tile indices are $(r, s) \in \{-1, 0\} \times \{0, 1\}$, and the only solution of $s - r = 0$ is $(r, s) = (0, 0)$. Thus the only candidate partner point on ℓ inside G lies in block $B_{0,0}$, which is omitted from T_2 .

So ℓ cannot contain a third point of S .

This rules out slope $+1$.

Case 3: slope -1

Write ℓ as $y = -x + t$. For integer points on ℓ , the invariant $x + y$ is constant.

Again, the two congruent points from one residue class must be the two corners on a slope -1 diagonal of its congruence square.

For such a diagonal, the two corners have the same value of $r + s$:

- If $a \leq h$ (tiles $r \in \{0, 1\}$), the -1 diagonal uses $(r, s) = (0, 1)$ and $(1, 0)$, hence $r + s = 1$.
- If $a > h$ (tiles $r \in \{-1, 0\}$), the -1 diagonal uses $(r, s) = (-1, 1)$ and $(0, 0)$, hence $r + s = 0$.

Using which blocks were omitted:

- type B omits $B_{0,0}$, so a -1 diagonal needing $(0, 0)$ cannot be fully present for type B ;
- type C omits $C_{0,1}$, so a -1 diagonal needing $(0, 1)$ cannot be fully present for type C .

Therefore, the only types that can supply two congruent points on a -1 diagonal inside S are A and D :

- type A can supply the pair in $A_{0,1}$ and $A_{1,0}$ (here $r + s = 1$),
- type D can supply the pair in $D_{-1,1}$ and $D_{0,0}$ (here $r + s = 0$).

We handle them similarly.

Subcase 3a: the congruent pair comes from type A (so $r + s = 1$).

Let (a, b) be the base representative with $a \in \{1, \dots, h\}$ and $b \in \{h + 1, \dots, p - 1\}$. The two congruent points in S lie in tiles $(0, 1)$ and $(1, 0)$, hence both satisfy

$$x + y = (a + b) + (r + s)p = (a + b) + p.$$

If ℓ contains a third point of $H(c, p)$, it must come from the second residue class on ℓ . By Lemma 6 (slope -1), the partner residue class has base representative

$$(a', b') = (b, a),$$

which is type D , and satisfies $a' + b' = a + b$.

Any representative of this partner class in tile (r, s) has

$$x + y = (a' + b') + (r + s)p = (a + b) + (r + s)p.$$

To lie on the same line ℓ we need

$$(a + b) + (r + s)p = (a + b) + p \iff r + s = 1.$$

For type D , the available tile indices are $(r, s) \in \{-1, 0\} \times \{0, 1\}$, and the only solution of $r + s = 1$ is $(r, s) = (0, 1)$. Thus the only candidate partner point on ℓ inside G lies in block $D_{0,1}$, which is omitted from T_2 .

So ℓ cannot contain a third point of S .

Subcase 3b: the congruent pair comes from type D (so $r + s = 0$).

Now (a, b) has $a \in \{h + 1, \dots, p - 1\}$ and $b \in \{1, \dots, h\}$. The congruent pair lies in tiles $(-1, 1)$ and $(0, 0)$, hence

$$x + y = (a + b) + (r + s)p = a + b$$

(because here $r + s = 0$).

By Lemma 6 the partner residue class is $(a', b') = (b, a)$, which is type A and satisfies $a' + b' = a + b$. A representative of type A lies in tiles $(r, s) \in \{0, 1\} \times \{0, 1\}$, and lying on ℓ forces $r + s = 0$, i.e. only tile $(0, 0)$. Thus the only candidate partner point on ℓ inside G lies in block $A_{0,0}$, which is omitted from T_2 .

So ℓ cannot contain a third point of S .

This rules out slope -1 .

All slope cases lead to a contradiction. Therefore no line contains three points of S .

Thus, the solution to this problem is to first find the largest prime less than or equal to $\frac{n}{2}$, and construct $3p - 3$ points using the method described above. If this is not enough, shuffle the rest of the unpicked points and add them greedily into S until there are m points. However, for some small n this may be too slow. For these special values of n , you can simply hard code them.

Problem L. Perimeter

Consider the contribution of each black cell to the final answer:

- Each black cell contributes 4.
- However, for a pair of adjacent black cells, the sharing edge will be counted twice towards the final answer, whereas its actual contribution is 0. Thus, every pair of adjacent black cells contributes -2 .

Let the total number of cells be $N = nm$, and the total number of adjacent pairs of cells be $E = 2nm - n - m$. Also, let the number of black cells be X , and the number of adjacent pairs of black cells be A .

For a cell v , the probability of it being black after k operations is:

$$\begin{aligned} \Pr(v \text{ black}) &= 1 - \Pr(v \text{ white}) \\ &= 1 - \left(1 - \frac{1}{N}\right)^k \\ &= 1 - \left(\frac{N-1}{N}\right)^k \end{aligned}$$

For a pair of adjacent cells u and v , the probability of both of them being black after k operations can be calculated using inclusion-exclusion:

$$\begin{aligned} \Pr(u \text{ black and } v \text{ black}) &= 1 - \Pr(u \text{ white}) - \Pr(v \text{ white}) + \Pr(u \text{ white and } v \text{ white}) \\ &= 1 - 2\Pr(u \text{ white}) + \Pr(u \text{ white and } v \text{ white}) \\ &= 1 - 2\left(\frac{N-1}{N}\right)^k + \left(\frac{N-2}{N}\right)^k \end{aligned}$$

By linearity of expectation, the final answer is thus:

$$\begin{aligned}\mathbb{E}[P] &= 4\mathbb{E}[X] - 2\mathbb{E}[A] \\ &= 4 \cdot N \cdot \Pr(v \text{ black}) - 2 \cdot E \cdot \Pr(u \text{ black and } v \text{ black}) \\ &= 2(n+m) + 4(nm - n - m) \left(\frac{nm-1}{nm}\right)^k - 2(2nm - n - m) \left(\frac{nm-2}{nm}\right)^k\end{aligned}$$

Overall time complexity $O(\log k + \log M)$.

Problem M. Classic Revisited

By running the naive $O(n^2)$ dynamic programming solution locally, it can be observed that within the given constraints, the answer seems to be around $O(\sqrt{n})$. Recall that the classical way of solving this problem is to define $dp_{l,r}$ as the length of the longest palindromic subsequence of the subarray $a[l..r]$.

We can instead do dynamic programming slightly differently: define $dp_{l,i}$ to be the **maximum** r , such that the length of the longest palindromic subsequence of $a[1..l] + a[r..n]$ is $2i$. The transition is as follows:

$$dp_{l,i} = \max(\{r \mid \exists k < l, dp_{k,i-1} > r\})$$

It's easy to see that if $r = dp_{l,i}$, then $a_l = a_r$. Combining this with the property that a is generated randomly, we can simply enumerate r such that $a_r = a_l$ and maintain a prefix maximum of $dp_{k,i-1}$ to calculate $dp_{l,i}$ from $1 \leq l \leq n$. For each i , the complexity for calculating all values of $dp_{l,i}$ is $O(\sum v^2)$, which is $O(n)$ under the constraints of this problem.

With careful implementation, overall complexity should be $O(n\sqrt{n})$.