Problem A. Square Sum

Let $m = p_1^{k_1} \dots p_n^{k_n}$, where p_i are prime and $k_i > 0$. Due to Chinese remainder theorem, we can solve the problem modulo $p_i^{k_i}$ independently, and multiply the results. Let $z = z_0 + p^k z_1$ and consider the equation

$$x^2 + y^2 \equiv z \pmod{p^{k+1}}.$$

If we represent $x = x_0 + p^k x_1$ and $y = y_0 + p^k y_1$, we can notice that

$$x_0^2 + y_0^2 \equiv z_0 \pmod{p^k},$$

thus we can also rewrite it as

$$2p^k(x_0x_1 + y_0y_1) \equiv up^k \pmod{p^{k+1}},$$

where $up^k \equiv z - x_0^2 - y_0^2 \pmod{p^{k+1}}$. We may further reduce this equation by p^k, getting

$$2(x_0x_1 + y_0y_1) \equiv u \pmod{p}.$$

There are several cases to consider now.

$p > 2$ case

With known x_0, y_0 and u, we can rewrite it as

$$\alpha x_1 + \beta y_1 \equiv \gamma \pmod{p},$$

where α, β and γ are constants. Note that when $\alpha \not\equiv 0 \pmod{p}$ or $\beta \not\equiv 0 \pmod{p}$, there are exaclty p valid pairs of (x_1, y_1), as we can uniquely recover at least one of the indeterminates from the other.

Now, what if $\alpha \equiv \beta \equiv 0 \pmod{p}$? Any such solution modulo p^{k+1} can be obtained from the solution to

$$\left(\frac{x}{p}\right)^2 + \left(\frac{y}{p}\right)^2 = \frac{z}{p^2} \pmod{p^{k-1}},$$

so we can get such solutions from the solutions for $\frac{z}{p^2}$ modulo p^{k-1} in p^2 ways each. In other words, let $f(z,k)$ be the number of solutions to $x^2 + y^2 \equiv z \pmod{p^k}$ such that at least one of x and y is not divisible by p, and $g(z,k)$ be the total number of solutions. Then $f(z,k+1) = pf(z,k)$, and $g(z,k)$ is

$$g(z,k) = \begin{cases} f(z,k) + p^2 g\left(\frac{z}{p^2}, k-2\right), & z \equiv 0 \pmod{p^2}, \\ f(z,k), & z \not\equiv 0 \pmod{p^2}. \end{cases}$$

Note that from $f(z,k+1) = pf(z,k)$ it follows that $f(z,k+1) = p^k f(z,1)$, so only $k = 1$ matters.

$p = 2$ case

Note that for $p = 2$ it generally holds that

$$x^2 + y^2 \equiv (x_0 + 2^k x_1)^2 + (y_0 + 2^k y_1)^2 \equiv x_0^2 + y_0^2 \pmod{2^{k+1}}$$

It means that the k-th bit of x and y can be arbitrary as long as $x_0^2 + y_0^2 \equiv z \pmod{2^{k+1}}$. Thus, we may search for x and y as $x = x_0 + 2^{k-1}x_1$ and $y = y_0 + 2^{k-1}y_1$, where $x_0^2 + y_0^2 \equiv z \pmod{2^k}$ and $x_0, y_0 < 2^{k-1}$. Due to the fact above, there are exactly $\frac{f(z,k)}{4}$ such (x_0, y_0) pairs. In such way, we get:

$$2^k(x_0x_1 + y_0y_1) \equiv u2^k \pmod{2^{k+1}}.$$

The identity above holds as long as $2(k-1) \geq k+1$, that is $k+1 \geq 4$, and it has 8 solutions for $x_1, y_1 < 4$. Using it with the fact that there are $\frac{f(z,k)}{4}$ such pairs of (x_0, y_0). Therefore, for $k > 3$ it also holds that $f(z, k+1) = 2f(z, k)$, and for $k \leq 3$ the answer may be found with brute force.

$k = 1$ case

The solution above developed a recurrence for p^k, but it still requires handling base case of $k = 1$. For $p > 2$, it can be proven (see e.g. at https://math.stackexchange.com/questions/398200/) that

1. If $p \equiv 1 \pmod 4$, then $f(z, 1) = 2(p-1)$ for $z \equiv 0 \pmod p$ and $f(z, 1) = p - 1$ otherwise;
2. If $p \equiv 3 \pmod 4$, then $f(z, 1) = 0$ for $z \equiv 0 \pmod p$ and $f(z, 1) = p + 1$ otherwise.

Problem B. Super Meat Bros

Formal statement You maintain two sequences. There are a_k ways to append k objects to the first sequence and b_k ways to do so for the second sequence. You're given $a_1, \dots, a_n$ and $b_1, \dots, b_n$. You generate a new sequence in the following way:

1. Zero or more times append any number of objects to the first sequence;
2. Zero or more times append any number of objects to the second sequence;
3. Merge two sequences into one single sequence, preserving the internal order of initial sequences.

How many different final sequences are there that consist of exactly m objects?

1. Binomials Let $A(x) = a_1x + a_2x^2 + \dots + a_nx^n$ and $B(x) = b_1x + b_2x^2 + \dots + b_nx^n$. Then

$$C = \frac{1}{1-A} = 1 + A + A^2 + \dots = c_0 + c_1x + c_2x^2 + \dots$$

is the generating function for the sequence obtained on the first step and

$$D = \frac{1}{1-B} = 1 + B + B^2 + \dots = d_0 + d_1x + d_2x^2 + \dots$$

is the generating function for the sequence obtained on the second step. Finally,

$$F = \sum_{i,j} \binom{i+j}{i} c_i d_j x^{i+j} = f_0 + f_1x + f_2x^2 + \dots$$

is the generating function for the final sequence, thus our final objective is to compute f_m. For smaller m, all f_m can be computed simultaneously as a convolution of sequences $\frac{c_k}{k!}$ and $\frac{d_k}{k!}$. However, in this problem, m might be arbitrarily large. To tackle it, one should note that $f_0, f_1, \dots$ is actually a linear recurrence of degree at most n^2.

2. Umbral calculus Let $T : R[c, d] \to R$ be a linear functional such that

$$T(c^i d^j) = c_i d_j.$$

Then one can write

$$T((c + d)^k) = f_k.$$

From the problem statement we know that

$$\begin{aligned} c_m &= a_1 c_{m-1} + a_2 c_{m-2} + \cdots + a_n c_{m-n}, \\ d_m &= b_1 d_{m-1} + b_2 d_{m-2} + \cdots + b_n d_{m-n}. \end{aligned}$$

Let $a(x) = x^n - a_1 x^{n-1} - \cdots - a_n$ and $b(x) = x^n - b_1 x^{n-1} - \cdots - b_n$ be the characteristic polynomials of c_i and d_j correspondingly. Then it holds that

$$T(c^i d^j a(c)) = T(c^i d^j b(d)) = 0$$

for any $i, j \geq 0$. Same would also be true for the span of such polynomials. In other words,

$$T(X) = T(Y)$$

for any $X(c, d)$ and $Y(c, d)$ such that $X - Y \in \langle a(c), b(d) \rangle$, where $\langle a, b \rangle$ is the ideal generated by a and b.

3. Composed sum To prove that f_m is a linear recurrence, we should find $f(c + d)$ such that

$$T(X) = T(Y)$$

for any $X(c + d)$ and $Y(c + d)$ if $X - Y \in \langle f(c + d) \rangle$. It is obviously true if $f(c + d) \in \langle a(c), b(d) \rangle$, so the problem boils down to finding any polynomial $f(c + d)$ in the $\langle a(c), b(d) \rangle$ ideal. Let

$$\begin{aligned} a(c) &= \prod_{i=1}^{n} (c - \lambda_i), \\ b(d) &= \prod_{j=1}^{n} (d - \mu_j). \end{aligned}$$

In this representation, you may define a polynomial

$$f(c + d) = \prod_{i=1}^{n} \prod_{j=1}^{n} [(c + d) - (\lambda_i + \mu_j)].$$

Alternatively you may rewrite it as

$$f(c + d) = \prod_{i=1}^{n} \prod_{j=1}^{n} [(c - \lambda_i) + (d - \mu_j)] = \sum_{d_{ij} \in \{0,1\}} \prod_{i=1}^{n} \prod_{j=1}^{n} (c - \lambda_i)^{d_{ij}} (d - \mu_j)^{1-d_{ij}}.$$

The sum has 2^{n^2} summands, each divisible by either $a(c)$ or $b(d)$, thus $f(c + d) \in \langle a(c), b(d) \rangle$.

Side note: One can prove that

$$f(c \diamond d) = \prod_{i=1}^{n} \prod_{j=1}^{n} [(c \diamond d) - (\lambda_i \diamond \mu_j)] \in \langle a(c), b(d) \rangle$$

for pretty much arbitrary well-defined binary operation $\diamond$.

4. Logarithms Computing f_m now reduces to the following tasks:

1. Compute $f_0, f_1, \dots, f_{n^2-1}$,
2. Compute the polynomial $f(x)$,
3. Compute $x^m \mod f(x)$ to get coefficients of $f_0, f_1, \dots, f_{n^2-1}$ that make up f_m.

First step is done as the binomial convolution of $c_0, c_1, \dots$ and $d_0, d_1, \dots$, the third step can be done with the process described on CP-Algorithms (`https://cp-algorithms.com/algebra/polynomial.html`).

Finally, for the second step, let's look on the logarithm of the reversed polynomial $x^n a(x^{-1})$:

$$\log x^n a(x^{-1}) = \log \prod_{i=1}^{n} (1 - \lambda_i x) = \sum_{i=1}^{n} \log(1 - \lambda_i x) = \sum_{i=1}^{n} \sum_{k=1}^{\infty} \frac{\lambda_i^k x^k}{k}.$$

Let $s_i = \lambda_1^i + \dots + \lambda_n^i$, $t_j = \mu_1^j + \dots + \mu_n^j$ and

$$r_k = \sum_{i=1}^{n} \sum_{j=1}^{n} (\lambda_i + \mu_j)^k = \sum_{i=1}^{k} \binom{k}{i} s_i t_{k-i},$$

then

$$\log x^n a(x^{-1}) = \sum_{k=1}^{\infty} \frac{s_k x^k}{k},$$
$$\log x^n b(x^{-1}) = \sum_{k=1}^{\infty} \frac{t_k x^k}{k},$$
$$\log x^{n^2} f(x^{-1}) = \sum_{k=1}^{\infty} \frac{r_k x^k}{k}.$$

The sequence $r_0, r_1, \dots$ can be computed as the binomial convolution of $s_0, s_1, \dots$ and $t_0, t_1, \dots$, after which $f(x)$ can be recovered with the polynomial exponent.

Problem C. Testing Subjects Usually Die

Formal statement You're given $p_1, \dots, p_n$ and c. Initially, number p is chosen in such a way that p has a probability of p_k to be k. You pick a distribution $q_1, \dots, q_n$. Then the number q is chosen such that q has a probability q_k to be k. If $p = q$, the game ends. Otherwise, the number p is renewed with probability c and stays the same with the probability $1 - c$. Then, the number q is chosen again. What is the minimum possible expected value E of tries until $p = q$?

Solution If $c = 1$:

$$E = \frac{1}{p_1 q_1 + \dots + p_n q_n}.$$

If $c = 0$:

$$E = \frac{p_1}{q_1} + \dots + \frac{p_n}{q_n}.$$

Otherwise:

$$\begin{gathered}
E = \sum_{i=1}^{n} p_i[1 + (1 - q_i)d_i], \\
d_i = cE + (1 - c)[1 + (1 - q_i)d_i], \\
d_i = \frac{cE + (1 - c)}{1 - (1 - c)(1 - q_i)}, \\
1 + (1 - q_i)d_i = \frac{1 + (1 - q_i)pE}{c + q_i - cq_i}.
\end{gathered}$$

Substituting it back into the sum, it is possible to get an explicit expression for E. Let $a_i = \frac{1}{c+q_i-cq_i}$ and $b_i = \frac{q_i}{c+q_i-cq_i}$, then $E = \frac{a^T p}{b^T p}$, thus

$$\begin{gathered}
E = \frac{\sum\limits_{i=1}^{n} \frac{p_i}{c+(1-c)q_i}}{\sum\limits_{i=1}^{n} \frac{p_i q_i}{c+(1-c)q_i}} = \frac{f(q)}{g(q)} \to \min, \\
q_1 + \dots + q_n = 1
\end{gathered}$$

Let's use Lagrange multipliers to solve the constrained minimization problem. Lagrangian:

$$L(q, \lambda) = \frac{f(q)}{g(q)} - \lambda(\vec{1}^T q - 1)$$

Its gradient:

$$\nabla L = \frac{g(q)\nabla f(q) - f(q)\nabla g(q)}{g^2(q)} - \lambda\vec{1}$$

Derivatives are:

$$\begin{gathered}
\frac{\partial f(q)}{\partial q_i} = \frac{-p_i(1-c)}{[c + (1-c)q_i]^2}, \\
\frac{\partial g(q)}{\partial q_i} = \frac{p_i c}{[c + (1-c)q_i]^2}
\end{gathered}$$

Placing it back:

$$p_i \frac{cf + (1-c)g}{[c + (1-c)q_i]^2} = \lambda$$

Thus for arbitrary i and j it holds that

$$\frac{p_i}{[c + (1-p)q_i]^2} = \frac{p_j}{[c + (1-c)q_j]^2} = \frac{\lambda}{cf + (1-c)g}$$

From this,

$$c + (1-c)q_i \sim \sqrt{p_i}$$

So the solution is

$$\boxed{q_i = \frac{C\sqrt{p_i} - c}{1 - c}}$$

Finding the constant:

$$\sum_{i=1}^{n} \frac{C\sqrt{p_i} - c}{1 - c} = 1,$$

$$\boxed{C = \frac{1 + c(n-1)}{\sqrt{p_1} + \dots + \sqrt{p_n}}}$$

This solution doesn't acknowledge $q_i \geq 0$, so it only works when $q_i \geq 0$ is globally optimal with respect to $q_1 + \dots + q_n = 1$. On the other hand, it is "obvious" that if p_i is non-increasing, so should be q_i, thus we can brute-force k such that $q_i = 0$ for $i > k$ and for $i \leq k$ we may use the solution above.

Overall complexity is $O(n \log n)$ due to sorting $p_1, \dots, p_n$.

Problem D. Triterminant

Formal statement Let $A_{-1}, A_0, A_1, A_2, \dots, A_n$ be a sequence of polynomials such that

$$A_{-1} = 1, \ A_0 = x,$$
$$A_k = x \cdot A_{k-1} - b_k \cdot A_{k-2}.$$

A sequence $b_1, b_2, \dots, b_n$ is called good if every coefficient of every A_k does not exceed 1 by absolute value.

You're given $c_1, c_2, \dots, c_n$ such that $c_k \in \{-1, 1\}$. You can multiply any c_k by -1 in one operation.

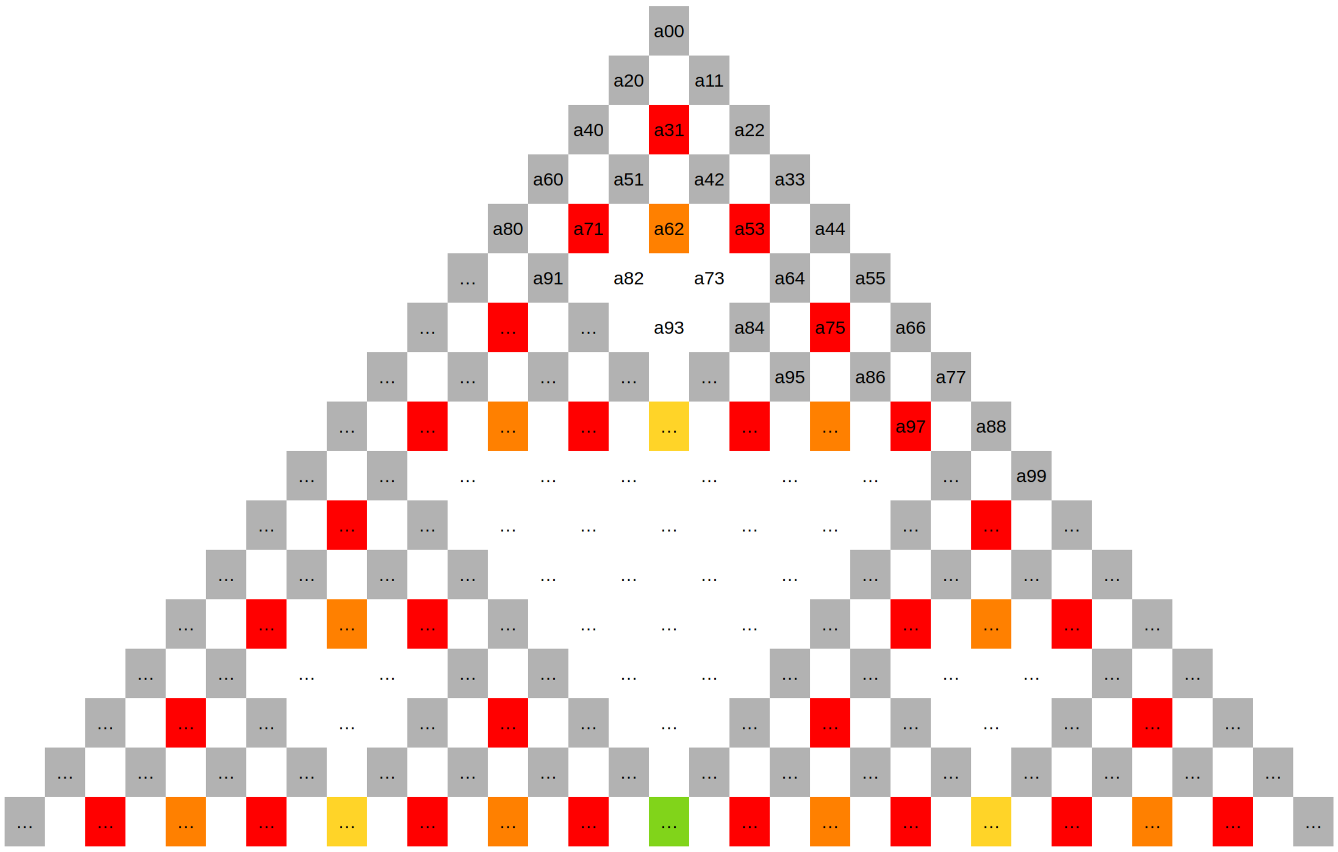
What is the minimum number of operations needed to make $c_1, c_2, \dots, c_n$ good?

Observations Let's multiply all b_i with -1, so that the recurrence has $+$ sign instead of $-$.

Let's analyze how $b_1, \dots, b_n$ affect coefficients of $A_1, \dots, A_n$. Let $a_{ij} = [x^j]A_i$, then

$$a_{ij} = a_{(i-1)(j-1)} + b_i a_{(i-2)j}.$$

Note that when both $a_{(i-1)(j-1)}$ and $a_{(i-2)j}$ are non-zero, b_i must be such that $a_{ij} = 0$. From this follows, that when fully expanded, every a_{ij} is either 0, or equates to a product of all elements of some subset of the sequence $b_1, b_2, \dots, b_n$. Keeping this in mind, we can, assuming $b_1, b_2, \dots$ to be infinite, organize all values of a_{ij} in a triangular shaped table, in which above the cell corresponding to a_{ij} are the cells corresponding to $a_{(i-1)(j-1)}$ to the left and $a_{(i-2)j}$ to the right:



On the picture above, for convenience, the polynomials are considered starting from $A_0 = 1$ and $A_1 = x$, rather than $A_{-1} = 1$ and $A_0 = x$ to maintain the property $\deg A_k = k$. Then, a special color designation is used for each cell. If the corresponding coefficient must be non-zero, the cell is colored grey.

Otherwise, the color of the cell depends on the structure of its parents. Generally, each zero cell has t non-zero parents in both directions (left and right). For example, the cell a_{62} has non-zero parents a_{40} and a_{51} to the left, and non-zero parents a_{42} and a_{22} to the right. So, on the picture above:

- White cells have 0 such parents (i. e. they are directly below cells that also have value 0),
- Red cells have 1 such parent on each side,
- Orange cells have 2 such parents on each side,
- Yellow cells have 4 such parents on each side,
- Green cells have 8 such parents on each side.

Note that when we descend from a grey node to the right into another grey node, it copies the value from that node, and if we descend to the left into the node a_{ij}, it gets multiplied by b_i. On the other hand, of a non-grey node has t grey parents, they will also reach a common parent in t more steps. In other words, each non-grey node a_{ij} with $t > 0$ grey parents defines an equation of form

$$b_{i-t-2(t-1)}b_{i-t-2(t-2)}b_{i-t-2(t-3)}\dots b_{i-t} + b_i b_{i-2} b_{i-4} \dots b_{i-2(t-1)} = 0.$$

The first summand here is obtained by multiplying t pieces of b_k from the left path to the common parent, and the second summand is obtained by multiplying t pieces of b_k from the right path to the common parent. For example, the cell a_{97} with $t = 1$ defines the equation $b_8 + b_9 = 0$, and the cell a_{62} defines the equation $b_2 b_4 + b_4 b_6 = 0$. These equations are not in a very convenient form (yet), but adhering to them is necessary and sufficient for a_{ij} to maintain the property that each a_{ij} is either -1, 0 or 1.

Note that, very conveniently, the equations defined by a_{ij} do not depend on j at all!

Now, these equations may be simplified by induction. First of all, we should notice that red equations only occur in odd i, so assuming $i = 2k + 1$ we may rewrite them as

$$b_{2k+1} + b_{2k} = 0$$

for every $k \geq 1$. What about orange equations? First such equation occurs in a_{62} with $i = 6$, where it looks like $b_2 b_4 + b_4 b_6 = 0$ and repeats with steps of 4, where it writes as

$$b_{4k-2}b_{4k} + b_{4k}b_{4k+2} = 0 \iff b_{2(2k-1)} + b_{2(2k+1)} = 0.$$

First yellow equation occurs with $i = 12$ and repeats with steps of 8, and rewrites as

$$b_{8k-6}b_{8k-4}b_{8k-2}b_{8k} + b_{8k-2}b_{8k}b_{8k+2}b_{8k+4} = 0.$$

On the other hand we know that $b_{2(4k-3)} + b_{2(4k-1)} = 0$ and $b_{2(4k-1)} + b_{2(4k+1)} = 0$, which rewrites

$$b_{8k-4}b_{8k} + b_{8k}b_{8k+4} = 0 \iff b_{4(2k-1)} + b_{4(2k+1)} = 0.$$

So, the equation with $t = 2^{n-1}$ first occurs in $i = 2^n + 2^{n-1}$ and repeats every 2^n steps, hence

$$b_{2^n k-2(t-1)} \dots b_{2^n k} + b_{2^n k-(t-2)} \dots b_{2^n k+(t-2)} b_{2^n k+t} = 0.$$

From this, we may by induction prove that such equation simplifies as

$$b_{2^{n-1}(2k-1)} + b_{2^{n-1}(2k+1)} = 0.$$

To do this, we should note that we may meticulously cancel out every odd multiplier until only two multipliers are left in each summand, one of them being $b_{2^n k}$. For example, with $t = 8$, we start with

$$b_{16k-14}b_{16k-12} \dots b_{16k-2}b_{16k} + b_{16k-6}b_{16k-4} \dots b_{16k+6}b_{16k+8} = 0.$$

On the first step, we cancel out the following pairs:

1. In the first summand, $b_{2(8k-7)}$ and $b_{2(8k-5)}$, then $b_{2(8k-3)}$ and $b_{2(8k-1)}$,
2. In the second summand, $b_{2(8k-3)}$ and $b_{2(8k-1)}$, then $b_{2(8k+1)}$ and $b_{2(8k+3)}$,

after which we're left with the equation

$$b_{16k-12}b_{16k-8}b_{16k-4}b_{16k} + b_{16k-4}b_{16k}b_{16k+4}b_{16k+8} = 0,$$

and we again cancel out:

1. In the first summand, $b_{4(4k-3)}$ and $b_{4(4k-1)}$,
2. In the second summand, $b_{4(4k-1)}$ and $b_{4(4k+1)}$,

after which we are left with

$$b_{16k-8}b_{16k} + b_{16k}b_{16k+8} = 0 \iff b_{8(2k-1)} + b_{8(2k+1)} = 0.$$

Formal proof for all $n, k \geq 1$ is a bit tedious, but should follow from the procedure above.

Solution Summarizing the above, we reduced everything to a set of equations:

$$\begin{cases} b_{2k+1} + b_{2k} = 0, & \forall k \geq 1, \\ b_{2^n(2k-1)} + b_{2^n(2k+1)} = 0, & \forall n, k \geq 1, \end{cases}$$

which is necessary and sufficient for the specified condition to hold. For a given $c_1, \ldots, c_n$ it means that we have at most $O(n \log n)$ constraints of kind "these two elements should have different colors", which may then be resolved by finding connected components of the graph defined like this, and comparing the colors of its vertices in bipartite coloring to the given $c_1, \ldots, c_n$, while choosing to recolor the minimum number of them necessary to make it into proper coloring.

Note that the coloring always exists, because the constraints above define a forest.

Problem E. Garbage Disposal

You may always pair up garbage piece i with bin type $i+1$ and vice versa.

If $R = L \neq 1$, or if $L \equiv R \equiv 0 \pmod 2$, disposal is impossible.

Otherwise, you may pair up i and $i+1$ as a "cross", and also $L, L+1, L+2$ in a cycle shift.

Problem F. Palindromic Polynomial

Hints

1. Use the property of palindromic polynomial that $A(x) = x^d A(1/x)$.
2. Assume $m = d+1$, and for every pair (x, y) in the input, there is also $(x^{-1}, y \cdot x^{-n})$. Prove that if you interpolate this polynomial, you will get a palindromic polynomial (although it might have zero leading coefficients).
3. Try solving the problem when degree d is given.
4. Input may implicitly constrain d.

Step 1

Let's call a polynomial $A(x) = \sum_{i=0}^{d} a_i x^i$ "k-palindromic" if $d \leq k$ and for all $i = 0 \ldots k$, $a_i = a_{k-i}$.

For example, $A(x) = 2x^4 + 3x^3 + 3x^2 + 2x$ is 5-palindromic, because $[0, 2, 3, 3, 2, 0]$ is a palindrome.

Palindromic polynomial of degree k is k-palindromic.

Sum of k-palindromic polynomials is k-palindromic. Multiplying k-palindromic polynomial by a number results in k-palindromic polynomial.

Lemma 1. If $A(x)$ is k-palindromic and $x \neq 0$, then $A(x) = x^k A(x^{-1})$.

Proof:

$$A(x) = \sum_{i=0}^{d} a_i x^i = \sum_{i=0}^{d} a_{k-i} x^i = \sum_{i=0}^{d} a_i x^{k-i} = x^k \sum_{i=0}^{d} a_i x^{-i} = x^k A(x^{-1}).$$

The main idea of the solution is to leverage this property to complete the given set of points to a set which for every x contains also x^{-1} and then interpolate the new set. Under certain conditions the result of interpolation will be automatically palindromic.

Step 2. Interpolation

Let's call a list of x-y pairs $X = \{(x_i, y_i)\}_{i=1\ldots m}$ a "dataset".

Let $A = \text{Interpolate}(X)$ - a polynomial which is result of interpolation of dataset X, that is a polynomial of smallest degree that passes through all points in the dataset. Then $\deg A < |X|$. Moreover, there exists exactly one polynomial of degree less than $|X|$ that satisfies X, and this polynomial is A.

We propose to use Newton's interpolating polynomials to calculate Interpolate(X). This method has complexity $O(|X|^2)$.

Step 3

Let's call a dataset k-palindromic if:

- All x_i are distinct.
- It doesn't contain point $x = 0$.
- For any i there is such j that $x_i x_j = 1$ and $y_i = x_i^k y_j$.

Lemma 2. Interpolating $(m-1)$-palindromic dataset of size m results in $(m-1)$-palindromic polynomial.

Proof. Let $X = \{(x_i, y_i)\}_{i=1\dots m}$ — $(m-1)$-palindromic dataset. Let $A(x) = \sum_{i=0}^{m-1} a_i x^i$ - the result of interpolating it. Then $degA \le m-1$ and $A(x_i) = y_i$ for all $(x_i, y_i) \in X$.

Let $A'(x) = \sum_{i=0}^{m-1} a_i x^{m-1-i} = x^{m-1} A(x^{-1})$. It has degree at most $m-1$.

For any $i \in 1\dots m$, by definition of k-palindromic dataset, $A'(x_i) = x^{m-1} A(x^{-1}) = x_i^{m-1} A(x_j) = x_i^{m-1} y_j = y_i = A(x_i)$, where $x_j = x_i^{-1}$.

Consider polynomial $f(x) = A(x) - A'(x)$. It has degree at most $m-1$. For $i = 1\dots m$, $f(x_i) = A(x_i) - A'(x_i) = A(x_i) - A(x_i) = 0$. So, $f(x)$ has at least m roots, but has degree at most $m-1$.

Polynomial of degree $(m-1)$ can have at most $m-1$ roots, unless it's constant zero. So, $f(x) \equiv 0$. Therefore, $A(x) \equiv A'(x)$, so $A(x)$ is $(m-1)$-palindromic.

Lemma 3. Interpolating m-palindromic dataset of size m, **which does not contain point x=1** results in m-palindromic polynomial.

Proof. Proof is almost identical to proof of Lemma 2.

Now degree of $A'(x)$ and $f(x)$ is at most m. But using the fact $A(1) = A'(1) = \sum_{i=0}^{m-1} a_i$, we conclude that $f(x)$ has at least $m+1$ roots (m points of dataset, plus x=1). So $f(x) \equiv 0$, which means $A(x)$ is palindromic.

Note. Polynomial from Lemma 3 will have degree less than m, so it will have $a_0 = a_m = 0$.

Step 4. Preparing the dataset

Let X - the dataset in the problem input. Let's pre-process it a follows:

- If it contains $(0, 0)$ - there is no solution (because that would mean $a_d = a_0 = 0$).
- If it contains $(0, y)$ - remove this point from the dataset and add it to a special one element dataset $X_0 = \{(0, y)\}$.
- If it contains $(1, x)$ - do nothing (don't add any new point or any constraint on d).
- If it contains $(-1, y)$ - add constraint $[y \cdot (-1)^d = y]$.
- If it contains $(x, y), x \ne 0, x \ne \pm 1$, but does not contain x^{-1}, add additional point $(x^{-1}, x^{-d} y)$.
- If it contains two points (x, y_1) and (x^{-1}, y_2), add a constraint $[y_1 = x^d y_2]$.

So as a result we got:

- Dataset X'. X' is d-palindromic. $0 \le |X'| \le 2n$. Some values in this set are symbolical expression, depending on d, which is not known at this point.
- Dataset X_0 which is either empty, or contains single pair $(0, a_0)$, $a_0 \ne 0$.

- Set of constraints C on d, all of form $[y_1 = x^d y_2]$.

If there exists palindromic polynomial A of degree d which passes through all points in X, it also must pass through points in $X' \cup X_0$ and d must satisfy all the constraints in C.

Step 5. Solving for fixed d

Now let's fix the degree of the desired polynomial to $d \in [0, d_max]$, such that d satisfies all constraints in C. Let's try to solve the problem for given d. Now we can evaluate all values in X' which depended on d.

Denote $m = |X'|$. There are 6 cases:

Case 1. $d < m$ or ($d = m$ and $|X_0| = 1$).

In this case $d < |X' \cup X_0|$. Let $A(x) = \text{Interpolate}(X' \cup X_0)$. If A has degree d and is palindromic, then it's the answer. Otherwise, there is no solution.

Proof. No polynomial of degree d, other than $A(x)$, satisfies X. So if $A(x)$ doesn't satisfy all the conditions, no other polynomial of degree d will.

Warning. In this case we need to specifically check whether the interpolated polynomial is constant zero (which can happen if $d = 0$. In this case we need to return "No solution" for given d.

Case 2. $d = m, X_0 = \emptyset, (1, y) \notin X'$.

In this case solution always exists and is

$$A(x) = a_0 x^d + R(x) + a_0,$$

where $a_0 \neq 0$ can be arbitrarily chosen (e.g. $a_0 = 1$), and $R(x) = \text{Interpolate}(X'')$, $X'' = \{(x, y - a_0(x^d + 1)) | (x, y) \in X'\})$.

Proof. By Lemma 3, $R(x) = \text{Interpolate}(X'')$ is m-palindromic and has $r_0 = r_m = 0$. So $A(x)$ is palindromic. $A(x)$ satisfies dataset X', because $A(x_i) = a_0 x_i^d + R(x_i) + a_0 = a_0(x_i^d + 1) + y_i - a_0(x^d + 1) = y_i$.

Case 3. $d = m, X_0 = \emptyset, (x_1 = 1, y_1) \in X'$.

Assume that solution exists and is $A(x)$. Let $a_0 = A(0)$. $a_0 \neq 0$, because $a_0 = a_d \neq 0$. Then $A(x) = \text{Interpolate}(X \cup \{(0, a_0)\})$. Then it can be written using Lagrange interpolating polynomials as:

$$A(x) = P(x) + a_0 L_0(x),$$

where $L_0(x) = \frac{(x - x_1) \ldots (x - x_m)}{(0 - x_1) \ldots (0 - x_m)}$ - basis Lagrange polynomial polynomial for point $x = 0$, $P(x) = \text{Interpolate}(\{X \cup \{(0, 0)\}\})$. Let's explicitly evaluate both of them.

Numerator of $L_0(x)$ has degree m. It is a product of polynomials of form $(x - x_i)(x - x_i^{-1}) = x^2 - (x_i + x_i^{-1}) + 1$, maybe $(x + 1)$, and necessarily $(x - 1)$ (because X' contains 1). Product of palindromic polynomials is palindromic. So, $L_0(x)$ is product of palindromic polynomial of degree $m - 1$ by $(x - 1)$. So, $l_d = -l_0$ and $l_m \neq 0$. What's necessary for the proof is that $l_d \neq l_0$.

Look at coefficients of $A(x)$. It must be that $a_0 = a_m$. So, $p_0 + a_0 l_0 = p_d = a_d l_d$. Therefore, $a_0 = \frac{p_0 - p_d}{l_d - l_0}$.

Now as we know a_0, evaluate $A(x)$. By construction it satisfies X'. We need to check that it has degree d and is palindromic. If so, it's the answer. Otherwise, answer does not exist.

Case 4. $d = m + 2k - 1$ $(k \geq 1)$.

In this case solution always exists and is

$$A(x) = a_0 x^d + R(x) \cdot x^k + a_0,$$

where $R(x) = \text{Interpolate}(X'')$, $X'' = \{(x, (y - a_0(x^d + 1)) \cdot x^{-k}) | (x, y) \in X'\})$.

Value a_0 is taken from X_0 if $X_0 \neq \emptyset$, otherwise a_0 can be chosen arbitrarily, e.g. $a_0 = 1$.

Proof. X'' is $(m-1)$-palindromic dataset. By Lemma 2, $R(x)$ is $(m-1)$-palindromic polynomial of degree at most $m-1$. So, $R(x) \cdot x^k$ is $(m+2k-1)$-palindromic. Therefore, $A(x)$ is palindromic as sum of palindromic polynomials $a_0x^d + a_0$ and $R(x)$.

Less formally, we are computing m coefficients of R, which are a palindrome, then adding k zeros on both sides, and then appending a_0 from both sides.

Case 5. $d = m + 2k$ $(k \geq 1), (1, y) \notin X'$.

In this case solution always exists and is

$$A(x) = a_0x^d + R(x) \cdot x^k + a_0,$$

where $R(x) = \text{Interpolate}(X'')$, $X'' = \{(x, (y - a_0(x^d + 1)) \cdot x^{-k}) | (x, y) \in X'\})$.

Note that formula to compute solution is identical to the Case 4. Proof is alsmost identical, except we must use Lemma 3 (that's why we need to require that 1 is not in X').

Case 6. $d = m + 2k$ $(k \geq 1), (1, y_1) \in X$.

In this case solution always exists and is

$$A(x) = R(x) + b \cdot S(x) \cdot x^k,$$

where $R(x)$ is the solution for dataset with removed pair $(1, y_1)$ (see Case 4)

$S(x)$ is a palindromic polynomial of degree m which is constructed in such a way that for all points x_i in dataset except 1, $S(x) = 0$, and $S(1) \neq 0$:

$$S(x) = (x - x_2)(x - x_3) \dots (x - x_m) \cdot (x + 1).$$

Finally, $b = \frac{y_1 - R(1)}{S(1)}$.

Each case has time complexity $O(m^2 + d) = O(n^2 + d_{max})$ (recall that $m \leq 2n$).

To summarize, if d meets all constraints in C and $d > m$, solution always exists. If $d \leq m$, solution may or may not exist.

Step 6. Putting it all together

We could just check all values d from 0 to d_{max}, and for every of them check if d satisfies all the constraints and then check if solution exists. But that would be too long $(O(d_{max} \cdot n^2))$.

Instead, we can compute the set D of all d meeting all constraints in C by directly checking all values d from 0 to d_{max}. This will take $O(n \cdot d_{max})$ time. We could also use discrete logarithm and get complexity $O(n\sqrt{M})$, but with given constraints that would not be faster.

If D is empty, then there is no solution.

If there is any $d \in D$, such that $d > m$, then just pick that d and solve for it using Case 4, 5 or 6. Solution will always exist.

Otherwise, there will be exactly one $d \leq m$ which satisfies all the constraints in C (see Lemma 4). We just need to check that value using Cases 1, 2 or 3. If solution exists for that d, output it. Otherwise there is no solution.

So, we got solution which does $O(n \cdot d_{max})$ preprocessing and then solves the problem for exactly one value of d, which has complexity $O(n^2 + d_{max})$. Overall complexity is $O(n \cdot d_{max})$. This assumed that arithmetical operations are $O(1)$. If we account for the fact that divisions modulo MOD are done in $O(\log MOD)$, the complexity is $\boxed{O(n \cdot d_{max} \cdot \log(MOD))}$.

Lemma 4. If $D \neq \emptyset$ and $d \leq m$ $\forall d \in D$, then $|D| = 1$.

Proof. We will prove that if D has at least two numbers in $[0, m]$ then it must have at least one more number in $[m+1, d_{max}]$.

Assume there are d_1, d_2 such that $d_1 < d_2 \le m$.

Every constraint has the form $x_k^d = r_k$. If both d_1 and d_2 meet this constraint, then $x_k^{d_1} = x_k^{d_2} \Rightarrow x_k^{d_2 - d_1} = 1$. Denote $\Delta = d_2 - d_1 \le m$. Then for any integer k, $x^{d_1 + k \cdot \Delta} = x^{d_1} = r_k$. This applies for every constraint, so $d_1 + k \cdot \Delta \in D$.

Take $k = \lceil \frac{m+1-d_1}{\Delta} \rceil$. Denote $d_3 = d_1 + \Delta k$. Then $\frac{m+1-d}{\Delta} \le k < \frac{m+1-d}{\Delta} + 1 \Rightarrow m+1 \le d_3 < m+1+\Delta$. Recall $\Delta \le m$ and $m \le 2n$. So $m+1 \le d_3 \le 2m \le 4n$.

As long as $d_{max} \ge 4n$, this means that there exists $d_3 \in [m+1, d_{max}]$ that satisfies all the constraints.

In this problem $n \le 1000$, $d_{max} = 10000$, so condition $d_{max} \ge 4n$ is satisfied.

Problem G. Palindromic Differences

Hints

1. How to tell whether answer is 0 or at least 1?
2. Which permutations of the array preserve the property "difference array is a palindrome"?
3. Solve this problem for the case when $a = [1, 2, \ldots, n]$.
4. When answer is positive and all elements are distinct, answer is the same as for the case $a = [1, 2, \ldots, n]$. How to deal with possible duplicate elements?

Step 1 - Permutations

Let's, for brevity, call an array whose difference array is a palindrome a **good** array.

Let's consider the case when $a = [1, 2, \ldots, n]$. Denote $k = \lfloor \frac{n}{2} \rfloor$. a itself is good.

There are two kinds of transformations which preserve the goodness of an array:

1. Swap two elements in mirror positions, i.e. a_i and a_{n+1-i}.
2. Swap any two elements a_i, a_j, while also simultaneously swapping another two elements that are in mirror positions to them, a_{n+1-i}, a_{n+1-j}.

It can be shown that all good arrays are reachable from $[1, 2, \ldots, n]$ by applying these operations.

In other words, a good array can be constructed in the following way. Consider k pairs: $(1, n); (2, n-1); \ldots; (k, n+1-k)$. Elements of every pair must be inserted on mirror positions. If n is odd, there will be also number $k+1$ which must always be in $(k+1)$-th position.

How many are there ways to construct a good array? There are $k!$ ways to distribute pairs of permutations among pairs of mirror positions. And there are additional 2^k ways to swap elements in each pair. So the answer is $k! \cdot 2^k$.

Step 2 - Arbitrary array, no duplicates

Now assume a can have arbitrary integer elements, but no duplicates. If there is a way to rearrange it to make it good, then using the same operations as in Step 1 we can reach all good arrays.

Every good array has the property that elements in mirror positions are mirrored relative to the same number mid: $a_i = mid - \Delta_i, a_{n+1-i} = mid + \Delta_i$, $i = 1 \ldots k$. In case of odd n the central element must be equal mid.

So, to check whether array can be rearranged to become good, we can just sort it and check if sorted array is good.

Step 3 - Handling duplicates

Now assume array a is sorted, is good, but has duplicates. Then there are still $k! \cdot 2^k$ permutations of this array which are good, but some of those permutations are identical. How to account for that?

First, we need to replace $k!$ with number of unique permutations of first k elements of a. This is the multinomial coefficient $\frac{k!}{c_1!c_2!\ldots c_d!}$, where d is number of distinct values in $a[1..k]$, $c_1, \ldots c_d$ are frequencies of different values.

Second, we need to account for the fact that some swaps are swaps of distinct elements. So we need to replace 2^k with 2^{k-l}, where l is the number of $i \in [1, k]$ such that $a_i = a_{n+1-i}$.

Step 4 - Solution

Sort the array a. Compute its difference array. if it is not a palindrome, the answer is 0.

Otherwise, the answer is

$$\frac{k!}{c_1!c_2!\ldots c_d!} \cdot 2^{k-l},$$

where $k = \lfloor \frac{n}{2} \rfloor$, d — the number of distinct elements in $a[1..k]$, $c_1, c_2, \ldots c_d$ — frequencies of distinct elements in $a[1..k]$, l — the number of $i \in [1, k]$ such that $a_i = a_{n+1-i}$.

Problem H. Graph Isomorphism

Graph has at most at most n graphs isomorphic to it $\iff$ it has at least $(n-1)!$ automorphisms. This is a very specific limitation and for $n > 4$ it essentially means that either any permutation is an automorphism, or all permutations with a certain fixed points are automorphisms.

For $n > 4$, the only graphs for which this is true are:

1. Empty graph;
2. "Star" graph;
3. "Star" graph complement;
4. Complete graph.

For $n \leq 4$, one should manually check the number of graph's automorphisms.

Problem I. DAG Generation

Let's solve this problem for each possible DAG independently. What is the probability that it will be generated?

It may be written as

$$\frac{\text{number of outcomes in which the graph is generated}}{\text{total number of outcomes}}$$

So, what exactly is an outcome of the generation? It is a DAG + one of its topological orderings. Note that all outcomes are uniformly distributed, as there is exactly 1 way to get each of the outcomes.

What is the number of outcomes that produce the given graph? It is the number of its topological orderings.

So, we need to sum up the following fraction over all DAGs:

$$\frac{(\text{number of DAG's topological orderings})^2}{(\text{total number of outcomes})^2}$$

Total number of outcomes may be shown to be $2^{\binom{n}{2}} n!$. Indeed, for each of $n!$ possible topological orderings there are $2^{\binom{n}{2}}$ DAGs that are compatible with it. Thus, we should focus on the square number of DAG's topological orderings. Counting topological orderings for a specific graph is a difficult task. At the same time, counting graphs for a given ordering is not.

The square of the number of DAG's topological orderings is essentially the number of pairs of its orderings. Using this, we may change the summation order and instead of counting pairs of orderings for each DAG, we will count DAGs for each pair of orderings. Next observation is that we may fix the first ordering to be $e = (1, 2, \dots, n)$ and then multiply the number by $n!$, as we check for all possible values of the first orderings and the sum over the second ordering will not change.

So, now we want to compute over all orderings $p = (p_1, p_2, \dots, p_n)$ the number of DAGs that satisfy both p and e.

What is the set of vertices that may be connected with pi? They should come earlier than p_i in both first and second orderings. In other words, it is such vertices that $j < i$ and $p_j < p_i$ simultaneously. Counting compatible DAGs over all permutations p yield the product:

$$(1)(1+2)(1+2+4)...(1+2+...+2^{n-1}) = (2^1-1)(2^2-1)(2^3-1)...(2^n-1).$$

Here $1+2+...+2^{k-1}$ denotes all possible options for the number of $\{p_j < p_i, j < i\}$ pairs with $p_i = k$.

That being said, the final answer is

$$1 - \frac{(2^1-1)(2^2-1)\dots(2^n-1)}{2^{n(n-1)} n!}.$$

Problem J. Persian Casino

Before all rewinds are used up, it is optimal to bet everything you got and amend in case it was not successful. After that, we bet 1 coin until the end of the game. Indeed, before each roll we can decide if we're going to amend it in case it fails. If we're going to amend it, it is optimal to bet everything, as we're guaranteed to win. Otherwise the expected value of winning is 0 regardless of how much we bet, so it is optimal to bet 1 to avoid running out of money.

After that, the expected value of each bet is 0, so it doesn't really matter how much to bet, if Prince may guarantee that he doesn't bankrupt.

If $2^M < N - M$, it would mean that prince may go broke in a very unlikely scenario and it should be reported.

Otherwise the answer is the sum $2^k p_k$ for $M \le k \le N$, where p_k is the probability that Prince uses M-th rewind on k-th bet plus 2^N times the probability that during the game he would lose less than M bets. The later is

$$\frac{\binom{N}{0} + \binom{N}{1} + \dots + \binom{N}{M-1}}{2^N}.$$

And for p_k, the formula is $\frac{\binom{k-1}{M-1}}{2^k}$.

That being said, the answer to the whole problem should be

$$\binom{N}{0} + \binom{N}{1} + \dots + \binom{N}{M-1} + \binom{N}{M}.$$

Because summing up $2^k p_k$ together, we get

$$\binom{M-1}{M-1} + \binom{M}{M-1} + \dots + \binom{N-1}{M-1} = \binom{N}{M}$$

It is also possible to rationalize it through dynamic programming.

Problem K. Determinant, or...?

The matrix described in such way is, generally, a block matrix

$$\begin{bmatrix} A & B \\ B & B \end{bmatrix},$$

where A and B are similar matrices of smaller size. From Gaussian elimination it is known that subtracting one row of the matrix from another does not change its determinants, so we can start by subtracting the bottom half of the rows from the upper one:

$$\begin{bmatrix} A-B & 0 \\ B & B \end{bmatrix}.$$

We may then repeat this process for $A - B$ and B, until they're bottom triangular, after which the determinant is computed recursively as $\det(A - B) \det B$.

Problem L. Directed Vertex Cacti

The answer is exactly $\binom{\binom{n}{2}}{m} n!$.

Arguably nicest way to obtain it:

For each of $\binom{\binom{n}{2}}{m}$ sets of non-cycle edges, there are exactly $n!$ ways to pick their orientation and add some cycles, so that one gets a directed cactus.

It is true for $m = 0$, because there are no non-cycle edges, and there are $n!$ ways to connect some of the vertices into cycles (corresponding to cycle presentation of a permutation). Now let's say we showed that there are $n!$ ways for some graph G, and we add an edge (u, v) to the graph.

- If u and v belong to different comparable cycles, we orient the (u, v) edge in a single possible way;
- If u and v belong to different incomparable cycles, we orient the (u, v) edge from smaller vertex to larger;
- If u and v belong to the same cycle, we apply the swap (uv) to the permutation represented by the strongly connected components as cycles, and orient the (u, v) edge from larger vertex to smaller;

This procedure is revertible, hence there will be the same amount of ways to pick orientation and assign cycles for all possible undirected graphs of non-cycle edges.

Problem M. Siteswap

Siteswap sequence $a_1, \dots, a_n$ defines a permutation $p_i \equiv i + a_i \pmod n$. One needs to find cycles of the permutation and check whether the cycle contains two elements of different parities. If it does, the ball on the cycle changes hands. Otherwise it's thrown by the same hand. Note that if balls do not change hand, but n is odd, it would mean that half of balls go to the left hand and half of balls go to the right hand, as hands alternate when the pattern is repeated.