Tritwise Mex Consider collection of k sets $H^{(1)}, \dots, H^{(k)}$ such that $H^{(i)} = \mathbb{Z} \cap [0; n_i)$ and k algebraic operations $f^{(i)} : H^{(i)} \times H^{(i)} \to H^{(i)}$. let's construct cartesian product $H = H^{(1)} \times \dots \times H^{(k)}$ which consists of vectors with i-th component lying in $H^{(i)}$. There is total of $n = n_1 \times \dots \times n_k$ such vectors and we may enumerate them by numbers from $\mathbb{Z} \cap [0; n)$ in lexicographical order. Let's define operation $f : H \times H \to H$ obtained by component-wise applying of operations $f^{(i)}(\cdot, \cdot)$. Now for two sequences $a = \{a_1, \dots, a_n\}$ and $b = \{b_1, \dots, b_n\}$ from elements over integral domain $\mathbb{K}$ we have to find their convolution $c = \{c_1, \dots, c_n\}$ over operation $f(\cdot, \cdot)$:

$$c_k = \sum_{f(i,j)=k} a_i \cdot b_j$$

Common idea here would be to make a transition into space where it is made by component-wise multiplication. Thus we have to find three families of mappings U_i, V_i and W_i from $\mathbb{K}^n$ to $\mathbb{K}$ such that:

$$\begin{cases} W_1(c) = U_1(a) \cdot V_1(b), \\ W_2(c) = U_2(a) \cdot V_2(b), \\ \dots, \\ W_t(c) = U_t(a) \cdot V_t(b). \end{cases}$$

We may rewrite this system via Hadamard (component-wise) product: $W(c) = U(a) \circ V(b)$. Let's only consider linear mappings, thus U, V and W are simply matrices. Many examples show that:

Theorem. *Let $W^{(t)}(c) = U^{(t)}(a) \circ V^{(t)}(b)$ where $a, b \in H^{(t)}$, c are convolutions of a and b over function $f^{(t)}$, $U^{(t)}$, $V^{(t)}$ and $W^{(t)}$ are some linear mappings. If we consider U, V and W such that:*

$$\begin{aligned} U &= U^{(1)} \otimes \dots \otimes U^{(k)}, \\ V &= V^{(1)} \otimes \dots \otimes V^{(k)}, \\ W &= W^{(1)} \otimes \dots \otimes W^{(k)}. \end{aligned}$$

where $A \otimes B$ is Kronecker product of matrices A and B, then for any $a, b \in H$ holds $W(c) = U(a) \circ V(b)$.

Proof. We will use some tensors here.[1] Consider operator $\operatorname{ravel}(x_{i_1 \dots i_t})$ which writes down all components of tensor in lexicographical order of their indices and returns tensor of first rank (which is vector) made up of these components. You may check that:

$$\operatorname{ravel}\left(A^{(1)}_{j_1 i_1} \dots A^{(t)}_{j_t i_t} x_{i_1 \dots i_t}\right) = (A^{(1)} \otimes \dots \otimes A^{(t)}) \cdot \operatorname{ravel}(x_{i_1 \dots i_t})$$

In this way Kronecker product of two matrices corresponds to their tensor product. Note that a, b and c may be presented by the tensor of rank k with dimensionalities $n_1, \dots, n_k$ which allows us to write:

$$\begin{aligned} c_{l_1, \dots, l_k} &= s^{(1)}_{l_1 i_1 j_1} \dots s^{(k)}_{l_k i_k j_k} a_{i_1 \dots i_k} b_{b_1 \dots b_k}, \\ s^{(t)}_{kij} &= \begin{cases} 1, \text{for } k = f^{(t)}(i,j), \\ 0, \text{for } k \neq f^{(t)}(i,j) \end{cases} \end{aligned}$$

[1]Here tensors are considered not as objects of linear algebra which hold mappings but rather as multidimensional arrays to which we apply Einstein summation rules. Thus in terms of tensors we should say that basis in our space is chosen once and for all. Otherwise we couldn't say about Hadamard product at all because it doesn't hold invariance under the change of coordinates and therefor can't be written by pure tensors.

Nonetheless given that basis is fixed, "tensors" we consider are correct presentations for some specific mappings we are interested in. Moreover we will say that basis is orthonormal and the space is Euclidean thus we may not distinguish covariant and contravariant components of tensors.

It almost allow us to formulate theorem in tensors. Let's rewrite Hadamard product via tensors:

$$(a \circ b)_\alpha = \delta_{\alpha i j} a_i b_j,$$
$$\delta_{\alpha i j} = \begin{cases} 1, \text{for } \alpha = i = j, \\ 0, \text{otherwise} \end{cases}$$

Now statement which we aim to prove may be formulated in the following way:

$$W^{(1)}_{\alpha_1 l_1} \dots W^{(k)}_{\alpha_k l_k} s^{(1)}_{l_1 i_1 j_1} \dots s^{(k)}_{l_k i_k j_k} a_{i_1 \dots i_k} b_{j_1 \dots j_k} = \delta_{\alpha_1 x_1 y_1} \dots \delta_{\alpha_k x_k y_k} U^{(1)}_{x_1 i_1} \dots U^{(k)}_{x_k i_k} a_{i_1 \dots i_k} V^{(1)}_{y_1 j_1} \dots V^{(k)}_{y_k j_k} b_{j_1 \dots j_k}$$

Where $U^{(t)}$, $V^{(t)}$ and $W^{(t)}$ satisfy following equation:

$$W^{(t)}_{\alpha l} s^{(t)}_{l i j} a'_i b'_j = \delta_{\alpha x y} U^{(t)}_{x i} a'_i V^{(t)}_{y j} b'_j$$

Here a' and b' are arbitrary vectors. Unlike matrix equations, we may freely rearrange multipliers in tensor formulas as long as we keep the order of their indices. Thus this equation due to a' and b' being arbitrary implies equity of considered tensors:

$$W^{(t)}_{\alpha l} s^{(t)}_{l i j} = \delta_{\alpha x y} U^{(t)}_{x i} V^{(t)}_{y j}$$

Multiplying left and right parts over all t we obtain what we need:

$$W^{(1)}_{\alpha_1 l_1} \dots W^{(k)}_{\alpha_k l_k} s^{(1)}_{l_1 i_1 j_1} \dots s^{(k)}_{l_k i_k j_k} = \delta_{\alpha_1 x_1 y_1} \dots \delta_{\alpha_k x_k y_l} U^{(1)}_{x_1 i_1} \dots U^{(k)}_{x_k i_k} V^{(1)}_{y_1 j_1} \dots V^{(k)}_{y_k j_k}$$

□

That means that if we may quickly calculate convolution in each $H^{(i)}$ separately, then we may also do it for the whole H. In practice it may be done via simple recursion which you should be familiar with if you know about Walsh-Hadamard transform.

Now to our particular problem. Let's write c_i for operation $\text{mex}(\cdot, \cdot)$:

$$\begin{cases} c_0 = a_1 b_1 + a_1 b_2 + a_2 b_1 + a_2 b_2, \\ c_1 = a_0 b_0 + a_0 b_2 + a_2 b_0, \\ c_2 = a_0 b_1 + a_1 b_0. \end{cases}$$

We may immediately see main invariant $(c_0 + c_1 + c_2) = (a_0 + a_1 + a_2)(b_0 + b_1 + b_2)$. Unfortunately we can't stay in 3-dimensional space to solve the problem, thus we should introduce new variable c_3:

$$\begin{cases} c_0 = (a_1 + a_2)(b_1 + b_2), \\ (c_0 + c_1 + c_2) = (a_0 + a_1 + a_2)(b_0 + b_1 + b_2), \\ (c_1 + c_3) = (a_0 + a_2)(b_0 + b_2), \\ c_3 = a_2 b_2 \end{cases}$$

Note that for this problem we've got $U = V \neq W$. Also note that solution actually works in $O(4^k)$.

Associativity Degree Using some brute-force we may note the following properties:

1. Solution always exists when $n > 2$.

2. Solution still exists if we say $i \circ j \leq 3$.

3. Solution still exists if we say $i \circ j = 1$ for $i > 3$.

Furthermore you may consider operations described by matrices of the following specific kind:

1. First row of matrix is of form $1^a2^b3^c$.
2. Second row of matrix is of form $1^d3^e1^f$.
3. Third row of matrix is of form $1^g3^h2^i$.
4. $\min(a, b) = 0$
5. $d \le 2$

Matrices of such form proved to cover all possible k at least for $n \le 100$ and as you may see, there are only $O(n^4)$ of them. You may come up with algorithm to check associativity degree of such matrices in $O(1)$ which will provide you $O(n^4)$ solution to the problem overall.

MHC Consider series $\sum\limits_{k=0}^{\infty} a_k x^k$ where a_k is the charge in $(k+1)$-th section. initially $a_0 = -1$ and all the others are equal to 0. Each second all beams go forward which corresponds to multiplying this series by x, after that they're either copied which corresponds to addition of the same series to the new one, or copied with inversion which corresponds to subtraction of the same series from the new one. Thus after n seconds we'll have the following series:

$$-(x+1)^\alpha (x-1)^{n-\alpha}$$

Here α is the total number of copies without inversion. As you see, order of signs doesn't matter. Now if we know α we may calculate values in all sections. Let's write down the value in $(k+1)$-th section:

$$-[x^k](x+1)^\alpha(x-1)^{n-\alpha} = (-1)^{n+\alpha+k+1}\sum_{i=0}^{k}(-1)^i\binom{\alpha}{i}\binom{n-\alpha}{k-i}$$

Up to $(-1)^\alpha$ multiplier it's the polynomial of α, let's denote it as $P_k(x)$. Now our problem is: we have a family of polynomials $P_i(x)$ which we may calculate explicitly and unknown number α. By our query we may calculate arbitrary $P_k(\alpha)$, in the end we'll have to recover α. To handle this let's recall that $P(x) = P(\alpha) \iff P(x) - P(\alpha) \equiv 0 \pmod{x-\alpha}$. Thus knowing $P_i(\alpha)$ we may calculate greatest common divisor of polynomials $P_i(x) - P_i(\alpha)$, which must be divisible by $x - \alpha$. given that we may choose k arbitrary from sufficiently large pool, calculated gcd will be almost always equal to $x - \alpha$, which may be checked explicitly for given constraints.

Basis Change For linear recurrence $F_n = \sum\limits_{i=1}^{k} a_{k-i}F_{n-i}$ let:

$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ a_0 & a_1 & a_2 & \dots & a_{k-1} \end{pmatrix}, \mathbf{F}_{n-k} = \begin{pmatrix} F_{n-k} \\ \dots \\ F_{n-2} \\ F_{n-1} \end{pmatrix}$$

Then one can see that $\mathbf{F}_n = A\mathbf{F}_{n-1} = A^n\mathbf{F_0}$. We want to find such $\{c_i\}_{i=1}^k$ that $F_n = \sum\limits_{i=1}^{k} c_i F_{n-b_i}$. It can be reformulated in matrix form as:

$$A^n\mathbf{F}_0 = \left(\sum_{i=1}^{k} c_i A^{n-b_i}\right)\mathbf{F}_0$$

This equation must hold for all $n \geq b_k$ and every possible $\mathbf{F}_0$. Note that if this equation holds for some n, it also holds for all $n' > n$. Thus we have to resolve it for the minimum possible n which is b_k. Also since $\mathbf{F}_0$ is arbitrary we can consider it as matrix equation:

$$A^{b_k} = \sum_{i=1}^{k} c_i A^{b_k - b_i}$$

This one can be resolved in $O(n^4)$ via gaussian elimination. But we want to improve that. Let's see what happens to the first row of A^n when we multiply this matrix by A. It will change like:

$$(r_0, r_1, \dots, r_{n-1}) \to r_{n-1} \cdot (a_0, a_1, \dots, a_{k-1}) + (0, r_1, \dots, r_{n-2})$$

One can see that it is clearly the same transform as for:

$$P(x) = \sum_{i=0}^{n-1} r_i x^i \to x \cdot P(x) \mod \left(x^k - \sum_{i=0}^{k-1} a_i x^i\right)$$

With $P(x) = 1$ for $A^0 = E$ we can see that first row of A^n composed of coefficients in:

$$x^n \mod \left(x^k - \sum_{i=0}^{k-1} a_i x^i\right)$$

If we denote $C(x) = x^k - \sum\limits_{i=0}^{k-1} a_i x^i$ then we can say that:

$$A^n = \begin{pmatrix} x^n \mod C(x) \\ x^{n+1} \mod C(x) \\ \dots \\ x^{n+k-1} \mod C(x) \end{pmatrix}$$

Thus if equation holds for first row of the matrix it will also hold for other rows as well. So the only thing left for us to do is to resolve equation:

$$x^{b_k} = \sum_{i=1}^{k} c_i x^{b_k - b_i} \mod \left(x^k - \sum_{i=0}^{k-1} a_i x^i\right)$$

This equation actually can be considered as system of linear equations with k variables and k equations if we calculate each summand modulo $C(x)$ and will set an equation for each term. Thus the problem can be solved in $O(k^3)$ with gaussian elimination.

Note that calculation of the summand with given modulo can be done in $O(k^2 \log b_k)$ in naive way with binary exponentiation or in $O(k \log k \log b_k)$ with FFT approach. First one will actually give you $O(k^3 \log b_k)$ solution, which was ok for the problem.

Scored Nim It may be proven by induction that greedily taking one stone at time from any heap with odd number of stones (or any heap if there is none) is sufficient. Thus the answer is $\left\lceil \left(\sum\limits_{i=1}^{n} a_i\right) / 2 \right\rceil$.

Milliarium Aureum We may consider each minor road (u, v) separately. Each road may ban some vertices w from being Rome, namely those that these road may change widest distance to u or v. That is, for $d(w, u) > d(w, v)$ must hold $\min(d(w, u), c) > d(w, v)$. That means that vertex w is affected if $d(w, u) \neq d(w, v)$ and $\min(d(w, u), d(w, v)) < c$. Let the thinnest edge on the path from u to v be x. Obviously, if $x \geq c$, we may ignore this minor road, it'll never relax distances because you may get not worse answer going from u to v via major roads instead. Now what if $x < c$? Then at least u and

v may no longer be Rome. What else? Consider connected components we have if we consider only major roads with length more than x. You may see that vertices in connected components of u and v will be banned and only those, because for all other vertices will hold $d(w,u) = d(w,v) = \min(x,y)$ where y is the thinnest edge on the path from w to some vertex lying on the path between u and v.

One of ways to find out which vertices were killed is to add major roads one by one from largest to smallest keeping connected components of vertices and process queries of kind "kill all vertices in connected component of v". If you merge components efficiently, it will take $O(n \log n)$ overall.

Permutant You may prove that if permutation is not cyclic, then the answer is 0. Otherwise it's the determinant of some cyclic matrix up to -1 multiplier. Cyclic matrix is the matrix of form:

$$C_n = \begin{pmatrix} a_1 & a_2 & \dots & a_{n-1} & a_n \\ a_n & a_1 & \dots & a_{n-2} & a_{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_3 & a_4 & \dots & a_1 & a_2 \\ a_2 & a_3 & \dots & a_n & a_1 \end{pmatrix}$$

Let $A(x) = \sum\limits_{i=0}^{n} a_i x^i$, it may be proven that $\det C_n = \prod\limits_{k=0}^{n-1} A\left(\omega^k\right)$ where $\omega = e^{\frac{2\pi i}{n}}$ is the n-th complex root of unity. It is known that $B(x) = \prod\limits_{i=0}^{n-1} (x - \omega^i) = x^n - 1$. Let's generalize it to arbitrary polynomials $A(x)$ and $B(x)$ so we may reduce our problem to the simpler one.

Consider polynomials $A(x) = \sum\limits_{i=0}^{n} a_i x^i$ and $B(x) = \sum\limits_{i=0}^{m} b_i x^i$ such that $a_n \neq 0$ and $b_m \neq 0$. Let $\lambda_1, \dots, \lambda_n$ be the roots of $A(x)$ and $\mu_1, \dots, \mu_m$ be the roots of $B(x)$ counted with their multiplicities. Then we may define *resultant* of polynomials $A(x)$ and $B(x)$:

$$\mathcal{R}(A,B) = b_m^n \prod_{i=1}^{m} A(\mu_i) = a_n^m b_m^n \prod_{\substack{1 \le i \le n \\ 1 \le j \le m}} (\mu_j - \lambda_j) = (-1)^{nm} a_n^m \prod_{i=1}^{n} B(\lambda_i)$$

It's immediately seen from definition that:

1. $\mathcal{R}(A,B) = (-1)^{nm} \mathcal{R}(B,A)$.
2. $\mathcal{R}(A,B) = a_n^m b_m^n$ when $n = 0$ or $m = 0$.
3. If $b_m = 1$ then $\mathcal{R}(A - CB, B) = \mathcal{R}(A,B)$ for arbitrary polynomial $C(x)$ and $n, m \ge 1$.
4. From this follows $\mathcal{R}(A,B) = b_m^{\deg(A) - \deg(A-CB)} \mathcal{R}(A - CB, B)$ for arbitrary $A(x)$, $B(x)$, $C(x)$.

These properties allow us calculate resultant alongside Euclidean algorithm, which works in $O(n^2)$.

The Game of Chance Let's calculate expected value for n turns:

$$dp_n = \frac{1}{m} \sum_{i=1}^{m} \max(i - dp_{n-1}, dp_{n-1} - i) = \frac{1}{m} \sum_{i=1}^{m} |i - dp_{n-1}|$$

We will search for answer as the fixed point x of such transform.

$$mx = \sum_{k=1}^{\lfloor x \rfloor} (x - k) + \sum_{k=\lfloor x \rfloor + 1}^{m} (k - x) = x(2\lfloor x \rfloor - m) - \frac{\lfloor x \rfloor(\lfloor x \rfloor + 1)}{2} + \frac{(m + \lfloor x \rfloor + 1)(m - \lfloor x \rfloor)}{2}$$

$$x = \frac{m + \lfloor x \rfloor + 1}{4} - \frac{\lfloor x \rfloor(\lfloor x \rfloor + 1)}{4(m - \lfloor x \rfloor)}$$

Taking a note that $|x - \lfloor x \rfloor| < 1$ we may say that $\lfloor x \rfloor \sim x$ and make an estimate of it by solving $4(m-x)x = (m+x+1)(m-x) - x(x+1)$. We will have that (you should check for ± 1 manually):

$$\lfloor x \rfloor \sim \left\lfloor \frac{2m+1-\sqrt{2m^2+2m+1}}{2} \right\rfloor$$

After we found $\lfloor x \rfloor$, we may simply put it in explicit formula for x above. This will give you $O(1)$ solution, it is also possible to find $\lfloor x \rfloor$ with binary search to has $O(\log m)$ solution.

Incomparable Pairs I tried to go easy on you with string challenges this time :)

Let total amount of distinct substring of s be x, then the answer is sum over all distinct substrings t of s of distinct substrings of t subtracted from $x(x+1)/2$. Now to get the sum of distinct substrings over all distinct substrings, let's for each prefix count number of new substrings which occured in it for the first time. Let this amount on prefix $[1,i]$ be j we may say that new substrings are $[1,i], [2,i], \dots, [j,i]$. There is a way to keep track in $O(n \log^2 n)$ of the number of substrings which lastly occured in position k when prefix $[1,i]$ via some additions on segments, it's used, for example, when you need to count the number of distinct substrings of some particular substring. Now if that number is a_k then for prefix $[1,i]$ you have to add to the answer value $\sum\limits_{k=1}^{i} \min(k,j)a_k = \sum\limits_{k=1}^{j-1} ka_k + \sum\limits_{k=j}^{i} ja_k$. That will provide you with $O(n \log^2 n)$ solution to the problem.

The Zong of the Zee There are two possibilities. Either we may get rid of all question marks or all question marks must be replaced by the same letter. To check this we should keep count of how many times each letter occured in each row and taking maximum of this value over all rows. In the end sum of all such values will be either n, so we may get rid of question marks or $n-1$, so sets of letters on each row are the same. Assume we got rid of all question marks. Then for each prefix and suffix of each column of length $m-1$ we should calculate how many times does it occur, it may be efficiently done via hashes. Then for each string which occurs as either prefix or suffix we should check that it occurs as prefix and suffix same amount of times. Let this amount be x, we should multiply $x!$ over all possible strings. Note that some permutations may be counted more than once, but we may resolve it because if permutation occurs for two different letters then it's always the permutation we may get by pasting some arbitrary letter which didn't occur among ones in input, so let amount of such permutations be π_0, we should sum up $\pi_0 + \sum\limits_{c=33}^{126} (\pi_c - \pi_0)$ where π_c is amount of permutations satisfying substitute of question marks by c. Overall complexity is $O(n^2)$.

Expected Value It may be proven by induction that answer is $a_1 - a_2$.