Problem A. Ascending Matrix

Let's first forget the constraint $A_{R,C} = V$. For each $i = 1, 2, \cdots, K-1$, consider the boundary between cells $\leq i$ and $i <$. Now the problem is to calculate the number of ways to write boundary paths so that they don't *cross over*. If we shift these paths so that the i-th path starts at $(N-(i-1), -(i-1))$, the problem is just counting of non-intersecting paths, which can be done by the LGV formula.

Remember we have $A_{R,C} = V$. We can rephrase this condition as "exactly $V-1$ paths pass the lower right side of the point p, and other paths pass the upper left side (no paths pass the p)". Here p is a point determined by R, C and V. By introducing the invariant x and multiplying the weight of lower right paths by x, our task boils down to calculating the determinant of a matrix as a polynomial of x. We can do this by substituting $x = 0, 1, \cdots$ and interpolating the polynomial.

The whole complexity is $O(K^2(N+M) + K^4)$.

Problem B. Bit Operation

We can rephrase the operation as "choose an element, and then delete its left or right neighbor". So we need to know, for each i, in how many ways the i-th element remains in the last. It's easy to derive a recurrence formula, and it turns out the coefficient is https://oeis.org/A059366. So you can calculate everything in $O(N)$ time.

Problem C. Count Min Ratio

First, consider the following problem:

- you are given non-negative intgegers W, A and B. Count the number of lattice paths from $(0,0)$ to $(W, AW+B)$ that doesn't pass above the line $y = Ax + B$.

We can see that the answer to this problem is $\binom{W+AW+B}{W} - A \times \binom{W+AW+B}{W-1}$. The proof is similar to that of the Catalan Number. Consider a path that goes from $(0,0)$ to $(W-1, H+1)$, and let p be the x-coordinate where this path first pass above the line $y = Ax + B$ (that is, move from $(p, Ap+B)$ to $(p, pA+B+1)$). If we fix p, the number of such paths is $Z \times \binom{(W-p)(A+1)-1}{W-p-1}$, where Z denotes the number of paths from $(0,0)$ to $(p, Ap+B)$ that doesn't pass above the line $y = Ax + B$. Using the fact that $A \times \binom{(W-p)(A+1)-1}{W-p-1} = \binom{(W-p)(A+1)-1}{W-p}$, we can prove the formula above.

Next, consider the following problem:

- You are given non-negative integers H, W, A and B. It is guaranteed that $0 \leq B \leq AW + B \leq H$. Consider a lattice paths from $(0,0)$ to (W, H). We define the weight of a path as the number of times the path passes a point $Ax + B$ for some x. Find the sum of weights of all paths.

Let's denote by $f(H, W, A, B)$ the answer to the above problem. Then consider the value $z = f(H, W, A, B) - f(H+1, W-1, A, B) \times A$. Obviously, we have

$$\begin{aligned} z = \sum_{0 \leq x \leq W} & (\text{the number of paths from } (0,0) \text{ to } (x, Ax+B)) \times \\ & ((\text{the number of paths from } (x, Ax+B) \text{ to } (W, H)) \\ & - (\text{the number of paths from } (x, Ax+B) \text{ to } (W-1, H+1)) \times A) \end{aligned}$$

The second multiplicand can be rephrased as the number of paths from $(x, Ax+B)$ to (W, H) that does not pass below the line $Ax + B$. Now, we can see that $z = \binom{W+H+1}{W}$. Furthermore, we have

$$f(H, W, A, B) = \sum_{0 \leq i \leq W} \binom{W+H+1}{i} A^{W-i}$$

Let's get back to the original problem. By the observation above, we need to find the value of $\sum_{1\le x\le N} g(x)$, where $N = \lfloor R/B \rfloor$ and $g(x) = (R+1-Bx) \times \left(\sum_{0\le i\le B} \binom{R+B+1}{i} x^{B-i}\right)$. To this end, we want to know, for each $k = 0, 1, \ldots, B+1$, the value of $\sum_{1\le i\le N} i^k$. This can be done by calculating the FPS of $exp(0x) + exp(1x) + \cdots + exp(Nx) = (1 - exp((N+1)x))/(1 - exp(x))$. The whole complexity is $O(BlogB)$.

Problem D. Do Use FFT

The main idea is to apply the technique in Tellegen's Principle into Practice (https://specfun.inria.fr/bostan/publications/BoLeSc03.pdf).

Let Z_k be the answer for a k. Now consider the following problem:

- You are given integers $D_1, D_2, \cdots, D_N$. Find the value of $\sum_{1\le k\le N} D_k Z_k$.

The outline of the solution to this problem is:

- Calculate the polynomial $f(x) = \sum_{1\le k\le N} \prod_{1\le i\le k} (x + B_i)$. We can do this by Divide-and-Conquer and FFT in $O(N \log^2 N)$ time.
- Calculate the value of $\sum_{1\le i\le N} C_i \times f(A_i)$. We can do this by multipoint-evaluation in $O(N \log^2 N)$ time.

Thus, transposing this algorithm gives a solution to the original problem which runs in $O(N \log^2 N)$ time. More specifically, the transposition of the second part is to calculate $\sum_{1\le i\le N} C_i A_i^j$ for each j and can be done by calculating $\sum_{1\le i\le N} \frac{C_i}{1-A_i x}$. For the transposition of the first part, the following expression might help you understand:

$$\begin{bmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ B_1 & 1 & & \\ & & 1 & \\ & & B_3 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ B_1B_2 & B_1+B_2 & 1 & \\ & & B_1B_2 & B_1+B_2 \end{bmatrix} \begin{bmatrix} 1 \\ & 1 \\ & & 1 \\ & & & 1 \end{bmatrix} \begin{bmatrix} \sum_i C_i A_i^0 \\ \sum_i C_i A_i^1 \\ \sum_i C_i A_i^2 \\ \sum_i C_i A_i^3 \end{bmatrix}$$

Problem E. Edge Subsets

We assume $A < B$ and $gcd(A, B) = 1$. Let's say we map a vertex v to the point (x, y), where $x = \lfloor v/B \rfloor$, $y < B$ and $yA \equiv v \bmod B$. Then the graph is almost like a grid graph, except for the edges like $(0, B-1) \to (1, 0)$. So we can do a bitmask DP. If we do DP from left to right, we can do a simple bitmask DP and the complexity is $O(N2^B)$. If we do DP from top to bottom, we need to remember the information of the top row and the complexity is $O(N4^{N/B})$. If we adopt the smaller one, the whole complexity is $O(N2^{\sqrt{2N}})$.

Problem F. Find the LCA

Let's take an arbitrary subset S of vertices. We are going to calculate the number of trees such that subtree of $LCA(N-1, N) = S$.

The crucial part is to count the number of trees with $n = |S|$ vertices such that $LCA(n-1, n) = 1$. It turns out that the count is $(n-1)!/2$ when $n \ge 3$. We can see that by creating a bijection between trees with the property and trees without the property. More precisely, if $LCA(n-1, n) = 1$, let y be the unique child vertex of 1 that have $n-1$ in its subtree. Then, take the subtree of y and insert it into the path between vertices 1 and n. It's easy to see that this is a bijection.

To compute the sum over all possible S, we can use Divide-and-Conquer and FFT, and the total complexity is $O(N \log^2 N)$.

Problem G. Games

For a state of the game, the second player will win iff, $(*)$ considering the binary representation of the number of the stones in each pile, for each bit position, the number of piles with that bit set is divisible by 7.

Proof sketch. For a state satisfying $(*)$, any move results in a state not satisfying $(*)$ because at least 1 and at most 6 piles unset the highest bit affected by the move. For a state not satisfying $(*)$, we can construct a move resulting in a state satisfying $(*)$ as follows: We adjust each bit from the highest to the lowest. Looking at the highest bit where the number of piles with that bit set is not divisible by 7, we can choose between 1 and 6 piles to unset that bit. We can now assume that all lower bits are set for those piles. Continuing to the lower bits, we can always prioritize already chosen piles to unset the bit so that the number of chosen piles does not exceed 6.

The counting problem is then just a convolution on $(\mathbb{Z}/7\mathbb{Z})^d$ where $d = \lceil \log_2(A_N + 1) \rceil \leq 7$. Luckily we have a primitive 7-th root of unity in $\mathbb{Z}/998244353\mathbb{Z}$ and we can compute DFT. You can either

- do the usual multidimensional DFT with naive DFTs on each dimension, taking $7^{d+1}d$ multiplications, or
- make use of the sparsity and add contribution of each A_i, taking $7^d N$ multiplications.

The pointwise exponentiation takes $O(7^d \log K)$ time (or $7^d \log 998244353$). The final answer can be found in $O(7^d)$ time as we do not need the whole inverse DFT.

Problem H. Harsh Comments

The answer is N plus the expected number of deleted comments other than yours. By the linearity of expectation, it suffices to consider the case where $M = 1$.

Let $S = \sum_{i=1}^{N} A_i$.

Instead of actually deleting the comments, consider adding a "deleted" tag whenever a comment is chosen. The problem becomes to find the probability that there exists a moment where exactly your N comments are tagged. This is equal to $\frac{B_j}{S+B_j}$ times the expected time the state continues. The expected time can be found by the principle of inclusion-exclusion: For each $I \subseteq \{1, \ldots, N\}$, find the expected time where the comments in I and the other's comment are kept untagged, multiply by $(-1)^{|I|}$ and add to the answer. As this only depends $\sum_{i \in I} A_i$, we just need to find their distribution by $O(NS)$ time DP. The total time complexity of $O(NS + SM \log \mathrm{MOD})$ is enough to pass.

Another solution. Consider a comment with x downvotes as x balls labeled by the comment. The deletion in the statement is equivalent to lining up all the balls in a row and then looking them one by one to delete the corresponding comment if it is not deleted yet. The problem becomes to find the probability that the first occurrence of a ball of the other's comment comes after the first occurrence of a ball of any of your comments. Using the principle of inclusion-exclusion, we can obtain the same formula as the solution above.

Bonus. Achieve $O(S \log S + SM + \log \mathrm{MOD})$ time.

Problem I. Inverse Problem

Let L be the largest integer such that $X_1 < X_2 < \ldots < X_L$. Then, the desired permutation is of the form (zero or more elements larger than X_1), X_1, (zero or more elements larger than X_2), X_2, ..., (zero or more elements larger than X_L), X_L, (zero or more elements larger than X_L) $X_{L+1}, X_{L+2}, \cdots, X_M$. We can see that such permutations do satisfy the condition and other permutations don't.

If we insert integers not present in X from smaller to larger, we know for each integer how many positions we can insert it to, and the answer is just a product of them.

Problem J. Japanese Knowledge

For simplicity, we assume A_N is $O(N)$.

Let $f_k(A)$ be the answer for a k, and let $g(A) = \sum_{0 \le k \le N} f(A)$ (that is, g is an easy version of f where you don't need to care about k). We can prove that $f_k(A) = g((A_{k+1} - 1, A_{k+2} - 1, \cdots, A_N - 1))$. We can see this by induction, but here we present a more elegant approach.

Consider the following problem:

- You are given a sequence X of 0's and 1's and have a stack S which is initially empty. For each $i = 1, 2, \cdots, |X|$, you'll do the following operation:
  - $X_i = 0$: Push i to the S.
  - $X_i = 1$: Pop zero or more elements from S. If the S becomes empty after the operation, we call i good, and otherwise not.

  Find the number of ways to perform operations so that exactly k indices are good.

It's easy to see that this problem is equivalent to the original problem: finding $f_k(A)$. Next, let's interpret this problem in reverse order, and then the problem will be like this:

- You are given a sequence X of 0's and 1's and have a stack T which is initially empty. T will maintain the candidates of good indices. For each $i = |X|, |X| - 1, \cdots, 1$, you'll do the following operation:
  - $X_i = 0$: Pop zero or more elements from T.
  - $X_i = 1$: Push i to the T.

  Find the number of ways to perform operations so that T contains exactly k elements at the end.

We can see that this is equivalent to finding $g((A_{k+1} - 1, A_{k+2} - 1, \cdots, A_N - 1))$.

Now our task is to count the number of paths in the young tableaux defined by A. We can do it by Divide-and-Conquer and FFT. Precisely speaking, we split the tableaux into three parts:

- Lowerl left part: $(A_1, \ldots, A_{N/2})$
- Middle part: $(A_{N/2}, A_{N/2}, \ldots, A_{N/2})$
- Upper right part $(A_{N/2+1} - A_{N/2}, A_{N/2+2} - A_{N/2}, \ldots, A_N - A_{N/2})$

We first compute the upper right part by recursion, then the middle part by FFT, and then the lower left part by recursion. The total complexity is $O(N \log^2 N)$.