Problem A. Alice, Bob and Game

It is found that the data range is very small, with four suits, and each suit (the cards that have been played) must form a continuous segment and must include the number 7.

In other words, each suit of cards has only 50 possible states, resulting in a total of only 6 250 000 states.

Each state can have at most 8 edges, and the states form a directed acyclic graph, so we can directly use memoized search.

Problem B. Big Data Permutation

We can maintain all maximal segments in a for which $a_{i+1} = b_{a_i}$ is satisfied after each modification operation.

For the query operation, specifically handle segments that cross l or r, and then query the complete segments within the interval.

Replace modification queries with queries that add and remove maximal segments.

Note that each query corresponds to a single cycle in permutation b — for segment addition and removal, all their elements are on one of the cycles, and for query — x is in one of the cycles. We can group all queries by a cycle and solve a problem independently for each cycle.

We can replace all values of x and all a_i by an index in the cycle; then every maximal segment will be denoted with $[l, r]$ and $[a, b]$, where the latter is a range of indices in the cycle.

Consider a cycle of length c. Now, to answer all queries for this cycle, we do divide and conquer on $[0, c)$. In other words, we break $[0, c)$ into $O(c)$ ranges as in a segment tree, and a node of a segment tree corresponding to a range $[\alpha, \beta]$ will consider all queries — modification, such that $[\alpha, \beta] \subset [a, b]$, and query such that $x \in [\alpha, \beta]$. This will mean we break $[a, b]$ into $O(\log c)$ nodes of a segment tree and put the query there, and for a single x, we will put a query to the x-th leaf and all it's ancestors.

For every node, answer all the queries in the same order as the original order. To answer the query, create a segment tree that supports setting a single value and querying a range maximum. For $([a, b], [l, r])$, we will set at l the value $r - l + 1$ on addition, and the value 0 on removal. For $([l, r], x)$ query, we will query the maximum on $[l, r]$.

The maximum value of all query answers for each occurrence of the query will be an answer to the query.

Time complexity is $O((n + m) \log^2 n)$.

Problem C. Corrupted Order

The upper bound for the answer is $2n(n - 1)$.

First, based on the target matrix, each position is cyclically shifted right by one position, and then cyclically shifted down by one position.

In this case, both row swaps and column swaps require at least $n(n - 1)$ swaps (the proof involves considering the set of numbers in each row and the theory of permutation cycles).

Then, from top to bottom, restore each row one by one, with the last row requiring $n - 1$ swaps, and each of the other rows requiring $2n - 1$ swaps.

The method for the last row is evident, so we only need to discuss how to restore the previous rows.

Assume that the numbers in this row after restoration are from $i \cdot n + 1$ to $i \cdot n + n$, and find their positions in the table.

If the position of $i \cdot n + x$ is cell (z, y), then connect point x to point y.

Since there are n points and n edges, there must be a cycle, and restore the numbers not in the cycle and those in the cycle (note the order).

For the numbers not in the cycle:

According to the topological order, each number can be restored in 2 swaps, first restoring the row (moving it to the bottom), and then restoring the column (also noting the order).

Since the restoration is done in topological order, it will not disrupt the numbers that have already been restored.

For the numbers in the cycle:

First, restore the entire row, then restore the entire column (the former does not create a situation where two numbers are in the same column due to the influence of points outside the cycle, while the latter is equivalent to the method for the last row).

There must be a cycle, so it is easy to prove that the number of swaps to restore a row is strictly less than $2n$.

Problem D. Difficult Password

Let $dp_{i,j}$ represent the number of ways for the first i numbers with state j (the state includes the last digit and the lengths of increasing/decreasing/repeated sequences).

However, since the range of L and R can go up to 10^9, we need to find a way to speed up this recurrence relation.

The Berlekamp-Massey algorithm can solve this problem.

Problem E. Edges and Divisors

Let `dp[i,j,k]` represent the maximum score when currently at point i, in the j-th round of operations, with the path edge weight sum remainder k. The transition is evident.

Note that when $k = 0$, update the answer and proceed to the next round, and although $k = 0$ at the beginning, one cannot directly proceed to the next round.

Problem F. Find And Modify

Perform block decomposition on the sequence, with block size B.

Contribution of complete blocks:

For each block, preprocess the contributions of (i, a_i) within the block to each b_j without restricting r. This requires $O(B)$ rectangle additions, followed by $O(n)$ point queries. The sweep line technique reduces this to $O(B)$ prefix additions and $O(n)$ single point queries.

For each modification, consider the contributions of $O(\frac{n}{B})$ complete blocks that cover the interval for each b_j. Maintain the contribution count from blocks to points, requiring one rectangle addition; during queries, check $O(\frac{n}{B})$ positions (i.e., the contribution of each block to the query point).

Contribution of scattered parts:

For each modification, also consider the contributions of the B scattered modifications before and after to all points, requiring $O(B)$ rectangle additions, and one single point query during the query.

If n and m are of the same order, take $B = \sqrt{n}$. The above problems can be handled with standard techniques, resulting in a time complexity of $O(n^{\frac{3}{2}})$.

Problem G. Generate Optimal Key

It is clear that each position is independent and does not affect the others; the cost of taking 0 or 1 at each position is easy to calculate.

However, there are some strings that cannot be taken, and all the forbidden strings have length L.

Insert these forbidden strings into a trie, and then let `dp[i]` represent the minimum cost of moving down from the i-th node of the trie.

Then consider whether to go left or right (which corresponds to taking 0 or 1 at the corresponding position), and be careful to handle the case where the subtree is empty.

Problem H. Huge Sequences

Use a sweep line algorithm to sweep r, maintaining the bitwise AND, bitwise OR, and GCD for each corresponding interval $[l, r]$.

The bitwise AND, bitwise OR, and GCD all form a semi-lattice of height $O(\log n)$, so the value for each left endpoint changes only $O(\log n)$ times. At each step of the sweep, the value changes for a suffix of the current position being swept. Use a timestamped prefix sum to support end insertions and deletions, maintaining the prefix history sum (for each l, maintain the sum of values for $[l', r']$ where $l' \leq l,\ r' \leq r$), and reduce the queries for all subintervals to two prefix history sum queries.

The time complexity is $O(n \log n + m)$.

Problem I. Integration of Lines and Poker

A relatively easy big simulation, without any disgusting details.