Problem Tutorial: "Swapping Inversions"

Obviously the order of swapping does not matter. The answer is just the sum of the differences of all inversions, which equals to $\sum_{i=1}^{n}(x_i - i)x_i$.

Problem Tutorial: "Hamiltonian Path"

If $p = 1$, we can simply choose the path $0, 1, \cdots, n-1$; also if $q = 1$, we can choose the path $n-1, n-2, \cdots, 0$. And if $gcd(p,q) > 1$, we surely cannot find a Hamilton path. So let's suppose that $p, q > 1$ and $gcd(p,q) = 1$.

First of all, let us consider the case that $n = p + q$. We can easily prove (using $gcd(p,q) = 1$) that the graph forms a cycle, so the answer is obvious.

Let the answer be $p_0, p_1, \cdots, p_{n-1}$. As you can only move from x to $x+p$ or $x-q$, the value $(p_{i+1} - p_i) \mod (p+q)$ must be p, so that the value $(p_i - p_0) \mod (p+q)$ must be $ip \mod (p+q)$.

Let us define V_i as the set of vertices whose number $\mod (p+q)$ is i. So the path must visit $V_{p_0 \mod (p+q)}, V_{(p_0+p) \mod (p+q)}, \cdots, V_{(p_0+(n-1)p) \mod (p+q)}$ in order. Consider the number of times to visit V_i, we can easily prove (using $p, q > 1$) that there will be only three cases:

- $|V_0| = |V_1| = \cdots = |V_{p+q-1}|$
- $|V_1| = |V_2| = \cdots = |V_{p+q-1}| = |V_0| - 1$
- $|V_0| = |V_1| = \cdots = |V_{p+q-2}| = |V_{p+q-1}| + 1$

For the first case, we can split the graph into $\frac{n}{p+q}$ cycles. Starting from $p-1$ and go along the cycle, we will finally stop at $p+q-1$. And then, instead of going back to $p-1$, we jump into the next cycle and start from $p+q+p-1$. After repeating it $\frac{n}{p+q}$ times, we can get the answer.

The second and the third case is similar. The only thing we need to do is to change the starting vertex and jump one step more/less than the first case.

The complexity is $\mathcal{O}(n)$.

Problem Tutorial: "Minimal Cyclic Shift"

In order to compute the answer, we should find $P(f(i) = f(i \bmod n + 1)), \forall 1 \le i \le n$.

For a string s of length n, we find that $P(f(i) = k | s\ is\ d-period) = \begin{cases} \frac{1}{d}, 1 \le k \le d \\ 0, else \end{cases}$

Let $g(d)$ be the number of $d-period$ strings of length n.

$\sum_{d|n} g(d) = 26^n$,

so we can have $g(n)$ by Moebius inversion.

$g(n) = \sum_{d|n} 26^d \mu(\frac{n}{d})$,

To compute our answer, we need to calculate $P(f(i) = k) = \sum_{d|n, d \ge k} 26^{-n} \frac{g(d)}{d}$, for every s and every k.

For every s, we can split the value $P(f(i) = k)$ into $d_0(n)$ blocks, the values are the same inside each block. So we can compute all the values in a block together and solve the problem in time complexity $O(n\sqrt{a_i})$.

Problem Tutorial: "Interval"

Consider the contribution of a fixed interval $[p, p+1]$.

Let the indices of the intervals covering $[p, p+1]$ be $i_1, i_2, \cdots, i_k$. Then for a $[L, R]$ covering at least one of the indices, we should add the answer by 1. Note that the indices covered by $[L, R]$ must be consecutive. We can use the following trick to avoid double counting:

For those $[L, R]$ covering i_1, add the answer by 1. Do the same thing for $i_2, i_3, \cdots, i_k$. Then for those $[L, R]$ covering both i_1 and i_2, subtract the answer by 1. Do the same thing for $[i_2, i_3], [i_3, i_4], \cdots, [i_{k-1}, i_k]$.

Now we can use sweep line and a set to maintain the indices covering the current $[p, p+1]$. Note that we don't need to compute the contributions of $[i_j, i_{j+1}]$ all the time: we can calculate it only when such an interval is created or deleted. Finally, we will get $O(n)$ events like: for those $[l, r]$ satisfying $l \leq x, r \geq y$, add its answer by v. This can be easily maintained via a Fenwick tree.

Problem Tutorial: "PlayerUnknown's Battlegrounds"

Enumerate the upper boundary i and the lower boundary j of the submatrix, and use an array $min[1..j]$ to dynamically maintain the minimum value of each column with the downward expansion of the lower boundary j.

Next, consider how far each element in mincan be extended to the left and right as the minimum element of the submatrix.

Take extending to the right as an example. The minimum value of $min[k]$ in a column of k will not affect the rightward expansion of the smaller elements on the left side of k but will affect the rightward expansion of the larger elements on the left side. A monotone stack is used to maintain an ascending sequence, which can be processed efficiently.

Similarly, it deals with the position where an element extends farthest to the left.

In this way, the number of submatrix with upper and lower bounds of i, j, and minimum value of $min[k]$ is $(right[k] - k + 1) * (k - Left[k] + 1)$.

Add the answer to ans [min [k]]

Finally, output the answers in the order of $1/simn/timesm$

Time complexity $O(n^2m)$

Problem Tutorial: "Sum"

When K is less than the rank of the matrix, there is obviously no solution. When the original matrix is 0 and $K = 1$, there is no solution. When the original matrix is a modulo 2 matrix of 1×1 and K and its elements There is no solution to parity at the same time.

For the remaining cases, there must be a solution:

When the original matrix is a 0 matrix, if $p \neq 2$, take $1, 1, \cdots, 1, -(K-1)$ and $1, 1, \cdots, 2, -K$ must have One satisfies the conditions; if $p = 2$, then K is an even number, just take $1, 1, \cdots, 1$.

When the original matrix is not a 0 matrix, using 1 and -1 can repeatedly reduce K by 2, so we only need to consider the case of $K = \text{rank}A + 1$.

If one of n, m is not 1, you might as well set $n \geq 2$, and then select the first or second column from the last group.

If $n = m = 1$, then $p \neq 2$, at this time, set the number of the last group as x, split into $x - 1, 1$ or $x + 1, -1$.

Time complexity $O(nmK)$.

Problem Tutorial: "Reasonable Workplace Relationship"

Obviously, the number of expected happy nodes in a subtree is the sum of the probability of each node in the subtree being happy.

Each query is essentially a subtree summation. We can get the summation during a DFS, then you can answer each group of queries by $O(1)$.

Then we consider how to quickly find out the probability of every node happy, that is, for each node u, how many nodes v in its subtree satisfy $a_v \leq a_u + w_u$

Maintain a binary indexed tree labeled with weights. DFS traverses the whole tree. When we enter a node u, the value of position a_u at the binary indexed tree should $+1$.

To get the number of node v in the subtree of u satisfied$a_v \leq a_u + w_u$, we just need to get the prefix-sum of $a_u + w_u$ with the BIT and denoted it as S_u.

After travel the whole subtree of node u, we get the prefix sum again, denoted t as E_u. Then the number of node v in the subtree of u satisfied$a_v \leq a_u + w_u$ is $E_u - S_u$.

Problem Tutorial: "Historic Breakthrough"

Note that if s is not a prime and $x^2 \equiv 1 \pmod{s}$, then if $x \not\equiv \pm 1 \pmod{s}$, we have $1 < \gcd(x-1, s) < s$, which means that we can find a non-trivial factor of s.

Moreover, by Euler's theorem, when n, a coprime, we have $a^{\varphi(n)} \equiv 1 \pmod{n}$, let $\varphi(n) = d \times 2^c$ where d is odd. With probability $> \frac{1}{8}$ a random a is coprime with n. In addition, consider a prime factor p^t dividing n, then there exists g such that $g^{\frac{1}{2}p^{t-1}(p-1)} \not\equiv 1 \pmod{p^t}$, which shows that there exists g such that $g^d \not\equiv 1 \pmod{n}$, and therefore at least half of a will satisfy $a^d \not\equiv 1 \pmod{n}$.

For such a number of a, we can first compute a^d and then square the value until we get $x^2 \equiv 1 \pmod{n}$, then with high probability $x \neq -1 \pmod{n}$ if n is not in the form p^q. Hence we can get a non-trivial factor of n by first enumerate q then randomize a, which shows that given $\varphi(n)$ factoring n is simple.

By similar procedure, we can further infer that given a multiple of $\varphi(n)$, factoring n is simple as well, we are also able to determine a non-trivial factor of a multiple of n. Thus we can use the above algorithm to get a multiset of prime divisors of $n\varphi(n)$ whose product is a multiple of n.

Then to derive n, note that for the largest prime factor p of n, suppose $p^k | n, p^{k+1} \not| n$, then $p^{2k-1} | n\varphi(n), p^{2k} \not| n\varphi(n)$, and we can infer the degree of p in n in the descending order of p, finally reconstruct the value of n. This procedure also shows that $n\varphi(n)$ is distinct and we do not need special judges.

Thus we need $O\left(\frac{\log^2 n \log \epsilon^{-1}}{\log \log n}\right)$ multiplications for each test case to ensure an error probability ϵ.

Problem Tutorial: "Nondeterministic Finite Automaton"

For $n = 6$, it is possible to use different heuristic approaches: simulated annealing, local search, evolutionary algorithms. (It is also claimed that a direct construction is possible, using an example from Černý conjecture with 5 vertices [citation needed]).

For $n = 20$, it's harder (but still might be possible) to use the above heuristics because the time complexity of finding $L(G)$ is around $O(2^n)$. However, there is a constructive approach.

Consider the following structure: a starting vertex and four directed cycles, of lengths 3, 4, 5, and 7. From the starting vertex, there are edges for both bits 0 and 1 going into the first vertex of each cycle, and each cycle's edge is repeated for both bits 0 and 1 as well. The last vertices of all cycles are not accepting, all the other vertices are accepting.

It can be seen that a cycle of length k accepts all words except those whose length is divisible by k. Since we can go from the starting vertex into any cycle, and we have cycles of lengths 3, 4, 5, and 7, our automaton accepts all words except those whose length is divisible by all cycle lengths, that is, divisible by $\text{LCM}(3, 4, 5, 7) = 420 > 400$. The number of vertices in our construction is $1 + 3 + 4 + 5 + 7 = 20$.

Problem Tutorial: "Texas Hold 'em"

Let p_1 be the probability that player 1 wins the hand after 5 random dealt are cards, and p_2 be this probability for player 2. WLOG assume $p_1 > p_2$.

Claim. It's optimal for player 1 to go all-in in the first betting round.

Proof. After player 1 goes all-in, player 2 has two options: either fold or call. Let the expected profit of player 1 in case of fold be x_1, and in case of call be x_2. Player 2 will obviously choose the option that minimizes the profit of player 1. In any case, player 1 can guarantee profit of at least $\min(x_1, x_2)$.

At the same time, player 2 can guarantee that player 1 will not get profit more than $\min(x_1, x_2)$. For example, if $x_1 \le x_2$, player 2 can just fold regardless of what player 1 does, and if $x_1 > x_2$, player 2 can just go all-in.

Thus, the expected profit of player 1 is exactly equal to $\min(x_1, x_2)$, and it can be forced if player 1 goes all-in in the first betting round.

It remains to calculate p_1 and p_2. We can just try all possible fives of dealt cards, there are $\binom{48}{5}$ of them, however, trying them all directly might be too slow.

One optimization is to note that if out of the 5 dealt cards, each suit is represented at most twice, neither of the players can have flush, and in this case, suits don't matter at all, all hands are decided by card ranks. At the same time, there are considerably less options to assign suits to the dealt cards if we force having at least three of some suit. Thus, we can start with fixing only the ranks of the dealt cards, evaluate the hands without considering flushes, and then only try assigning suits in a way that allows flush.