Editorials of NTU contest

National Taiwan University PECaveros

February 23, 2018

A Almost String Matching

Replace `a` by 1, `b` by 2, and so on. Two characters x, y are almost same if $(x-y)^2 \times ((x-y)-1)^2 = 0$. Expanding the above formula results in $x^4 - 4x^3y + 6x^2y^2 - 4xy^3 + y^4 - x^2 + 2xy - y^2$. Then, string a, b are almost same if $\sum_{i=1}^{|a|}(a_i - b_i)^2 \times ((a_i - b_i) - 1)^2 = 0$. After reversing the string b, the above summation will become a convolution form. Thus, we can find out each term of above formula by FFT or NTT. If the summation of those term equals 0, the corresponding position is matched. Note that we only need at most 7 times FFT or NTT by this method.

There's also another solution requiring $O(|\Sigma|)$ times FFT or NTT which won't fit in TL.

B BWT

First, we need to check if the input string is the result of BWT of some string. This can be done in linear time with a reverse BWT. If not, the answer is the input and we're done. If yes, we have to replace some characters. In general, we can replace the first character to 'a' to achieve the goal. The others are just trivial cases so we can handle them one by one.

C Calculus is Quite Easy

Let $p = a_1 + a_2 + \ldots + a_t$. The problem is equivalent to whether we cannot pick k terms $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ from p (repeat is allowed) such that $a_{i_1}a_{i_2}\ldots a_{i_k} = x_1 \ldots x_n y_1 \ldots y_n$ because all other terms in p^k become zero after partial derivation and variable substitution.

As a consequence, for $k \notin [n, 2n]$ the answer is always `YES` (result being zero).

Since each term in p is x_i, y_j or $x_i y_j$, we can model the problem as a minimum cost flow problem and use any existed algorithm to solve it.

D Guess the Answer

The following is a strategy that works within the query limit. First, we can transform our query into querying the number of "Yes" in a given subset of problems. Assume we have a strategy that determines k answers with m subset queries (one being the subset of all k questions). Then, there's a strategy to determine $2k+m-1$ answers in $2m$ queries. Partition $2k+m-1$ problems into three subsets A, B, C each of size $k, k, m-1$. Let $A_i, B_i (i = 1, 2, \ldots, m-1)$ be the subsets required to determine A, B respectively, C_i be the subset containing only the i-th element in C. Then we can determine the answer to A_i, B_i, C_i with only two subset queries: $A_i \cup B_i$ and $(A \backslash A_i) \cup B_i \cup C_i$ (due to the help of parity). With this strategy, we can determine 1000 answers in less than 300 queries.

E Count on the integers

Considering the parity of number of bit in binary form, the xor result of two integers with same parity will have even number of bits while two integers with different parity will have odd number of bits. Therefore, we should find the number of integer with odd/even number of bits in binary form of $[l_1, r_1] \cup [l_2, r_2] \cup \cdots \cup [l_i, r_i]$ for each i. There's lots of way to implement. We can maintain the active intervals by `set` or maintain a dynamic segment tree storing the number of odd/even(number of bits) active integer in each interval. Either way results in $O(N \lg N)$.

F Almost GCD

Keep each possible GCD so far. Let the set of possible GCD from first i integers be S_i. $S_{i+1} = gcd(S_i, a_{i+1}) \cup gcd(S_i, a_{i+1} - 1) \cup gcd(S_i, a_{i+1} + 1)$, where $gcd(s, v)$ means that $\cup_{i \in s} gcd(i, v)$.

G Valid Parenthesis Substrings

First, we can check whether a string $s = c_1 c_2 \ldots c_m$ is valid in linear time with a stack S: For $i = 1, 2, \ldots, m$, push c_i into S if it's empty or $c_i \neq top(S)$, pop from S otherwise. s is valid if and only if the resulting stack is empty.

From this algorithm, we can make a simple observation: If both $S[i, j]$ and $S[i, j + t]$ are valid, $S[j + 1, j + t]$ is also valid. Consider running the above algorithm on $S[i, j + t]$, the stack becomes empty after $S[j]$ and $S[j + t]$. Therefore $S[j + 1, j + t]$ is valid.

Let $f[i]$ be the smallest j such that $S[i, j - 1]$ is valid or -1 if there is no such j. From previous observation we can conclude that very valid substring starting from i must have form $S[i, f[i] - 1]S[f[i], k]$.

Suppose we have already computed $f[i]$ for every i, we can compute the answer using dynamic programming by computing $dp[i]$ as the number of valid substring starting from i.

To compute $f[\cdot]$, we again use dynamic programming. Let $g[i, c]$ be the minimum j such that $S[i, j-1]$ is valid and $S[j] = c$. Therefore $f[i] = g[i+1, S[i]]$ for every i and $g[i, c]$ can be maintained simultaneously.

H Mouse Trap

Considering the convex hull of the N points. If the convex hull contains all of the N points, the answer will be `-1`. Otherwise, considering each segment of the convex hull, the best way is to find some inner points and connect those two points with it. The best choice is the one closest two this segment. Therefore, the best choice for each segment will be one of points on the convex hull of those inner points. Thus, we can use two pointer tecnique to find out the best choice of each segment in $O(N)$ and choose the one results in maximum possible area.

I LCM and GCD

The idea is to compute contribution of each prime p to the answer. Let $f(p)$ be the exponent of p in the final answer, i.e, $ans = \prod_{p \leq M} p^{f(p)}$. Define $(x)_p$ to be the exponent of p in factorization of x, $[M]$ to be the set $\{1, 2, \ldots, M\}$, a/b to be the integer division (round down the result), $\mu(x)$ to be the Mobius function, $\phi(x)$ to be the Euler's totient function and Q to be modulo we use: $10^9 + 7$.

Obviously, $f(p) = \sum_{p^r, g} r \times g \times \#[\vec{X} \in [M]^n$ such that $lcm(\vec{X})_p = r, gcd(\vec{X}) = g]$. Or,
$\sum_{p^r, g} g \times \#[\vec{X} \in [M]^n$ such that $lcm(\vec{X})_p \geq r, gcd(\vec{X}) = g]$ to remove r in summation.

How can we compute the $\#[\ldots]$ term? Suppose we are now free of the lcm condition, $\#[\vec{X} \in [M]^n, gcd(\vec{X}) = g]$ is $\sum_{t, gt \leq M} \mu(t) \times \#[\text{multiples of } gt]^n$.

How we take the lcm condition into account? $\sum_{t, gt \leq M} \mu(t) \times (\#[\text{multiples of } gt]^n$
$-\#[\text{multiples of } gt \text{ but not multiple of } p^r]^n)$. Since the lcm condition indicates that some of X_i is divided by p^r, so we can count this by subtracting the case that none of X_i is divided by p^r.

Obviously, #[multiples of gt] $= M/gt$. #[multiples of gt but not multiple of p^r] = #[multiples of gt]− #[multiples of $lcm(gt, p^r)$] $= M/gt - (M/gt)/p^{r-(gt)_p}$ if $(gt)_p \leq r$; otherwise, $LHS = 0$ due to $p^r | gt$.

Replace the above formula back to $f(p)$ we got

$$f(p) = \sum_{p^r \leq M} \sum_{g=1}^{M} \sum_{t=1}^{M/g} g\mu(t) \times ((M/gt)^n - (M/gt - (M/gt)/p^{\max(r-(gt)_p,0)})^n) =$$

$$\sum_{g=1}^{M} \sum_{t=1}^{M/g} \left(g\mu(t) \times \sum_{p^r \leq M} ((M/gt)^n - (M/gt - (M/gt)/p^{\max(r-(gt)_p,0)})^n) \right)$$

.

Let V be M/gt, we can express $\sum_{p^r \leq M} ((M/gt)^n - (M/gt - (M/gt)/p^{\max(r-(gt)_p,0)})^n)$ as $(gt)_p V^n + \sum_{1<p^e \leq V} V^n - (V - V/p^e)^n$

Now we can write answer as:

$$\prod_p p^{f(p)} = \prod_p p^{\sum_{g=1}^{M} \sum_{t=1}^{M/g} \left(g\mu(t) \times \sum_{p^r \leq M} ((M/gt)^n - (M/gt - (M/gt)/p^{\max(r-(gt)_p,0)})^n) \right)} =$$

$$\prod_p \prod_{g=1}^{M} \prod_{t=1}^{M/g} p^{\left(g\mu(t) \times \sum_{p^r \leq M} ((M/gt)^n - (M/gt - (M/gt)/p^{\max(r-(gt)_p,0)})^n) \right)} =$$

$$\prod_{gt \leq M} \left(\prod_p p^{\sum_{p^r \leq M} ((M/gt)^n - (M/gt - (M/gt)/p^{\max(r-(gt)_p,0)})^n)} \right)^{g\mu(t)} =$$

$$\{\text{Let } V = M/gt\} \prod_{gt \leq M} \left(\prod_p p^{(gt)_p V^n + \sum_{1<p^e \leq V} V^n - (V - V/p^e)^n} \right)^{g\mu(t)} =$$

$$\prod_{gt \leq M} \left(\prod_p p^{\sum_{1<p^e \leq V} V^n - (V - V/p^e)^n} \times \prod_{p|gt} p^{(gt)_p V^n} \right)^{g\mu(t)}$$

Let $\gamma(V) = \prod_p p^{\sum_{1<p^e \leq V} V^n - (V - V/p^e)^n}$ and $\omega(x) = \gamma(M/x) \times \prod_{p|x} p^{x_p (M/x)^n}$. The answer is now $\prod_{gt \leq M} (\omega(gt))^{g\mu(t)}$.

By the fact that there are only $O(\sqrt{M})$ kinds of V we will encounter, we can precompute all possible $\gamma(V)$ in $O(\sqrt{M} \times \#[p \leq M] \times \log_p M) = O(M\sqrt{M})$ time by another precomputed table of $x^n \mod \phi(Q)$ for all $x \in [M]$. And we can compute $\omega(x)$ for all x in $O(M \log M)$ time using precomputed γ table and the fact that there are $O(\log x)$ prime divisor of x.

To compute the answer we simply iterate over all $O(M \log M)$ pairs of (g, t) and compute $(\omega(gt))^{g\mu(t)}$ in $O(\log Q)$ time per pair, this give us a $O(M \log M \log Q)$ time solution.

A simple speed-up is to aggregate $g\mu(t)$ for equal gt and compute power once per different gt. This gives us an $O(\log M)$ speed-up.

J String Game

In the first phase, we can permute the string S_1 arbitrarily as we want. Therefore, we only need to make sure the occurence of each characters in S_1 and T_1 is the same. Considering the operation on second phase, after choosing some j, the corresponding position j will have $S_2[j] = T_2[j]$. Then, this j is no longer useful. Thus, we can model it as a cost flow model. There are 26 vertices standing for each characters. For each vertice i, add an edge with capacity c and cost 0 from source if character i occurs in S_1 c times, and add an edge with capacity c' and cost 0 to sink if it occurs c' times in T_1. For each i, j, add an edge from vertex i to vertex j with capacity c and cost 1 if there are c integers k in $[0, |S_2| - 1]$ s.t. $T_2[k] = i$ and $S_2[k] = j$. If the max flow is less than $|S_1|$, the answer will be `-1`. Otherwise, the answer is the minimum cost.

K Let's PK!

Let $C(i, j)$ be the probability to get from $(1, i)$ to (M, j). Then the probability of the balls reaching the destinations without colliding is the determinant of the square matrix $[C(i, a_j)]_{i,j}$.

L Coin Tossing

Let's randomly guess the first N steps and choose a constant $C < N$. Then, Considering $x = (48271^{N-C+1})^i$ where $(N - C + 1) \times i \leq 2147483646$, there are about $\frac{2\times10^9}{N-C+1}$ possible x. We can set `seed` as each x and enumerate out first C terms. Then, check whether those C terms match any consecutive C terms among those first N results. Once we found a seed match the results, we can perfectly guess out all the following results.

By choosing $N = 800, C = 100$, the expected failure among first N steps is only about 400 which is acceptable while the probability that we found a wrong seed is about 2^{-C} which is small enough. But, it may exceed the time limit. One way to speed up is that we can choose some $C' < C$ like $C' = 32$. And, check only first C' terms. If some seed matchs, we check $C - C'$ terms to make sure.