CodeForces Educational Round 178 F

omeganot

July 23, 2025

When thinking about cases where $S(x) = S(y)$, we may consider the actual process of computing $S(x)$. When you add 1 to x, you only change the suffix of the string. You change the only suffix of the strings that takes the form $d9\ldots9$, where d is a digit from 0 to 8.

Thus, each permutation of the prefix (without leading zeros) will all double count $S(x)$, and we wish to consider the smallest prefix only. In other words, We may have $x = 2109$, which generates $S(x) = 00111229$, but the prefix is 21 and the suffix is 09, meaning $x = 1209$ is a smaller number that would create the same $S(x) = 00111229$.

I originally assumed that using Digit DP to count all numbers where the prefix is lexicographically minimum would not create duplicates, but as it turns out, it does, take $x = 199$ and $x = 901$ for example. As such, we must find another way. However, with the code already implemented, I tested $n = 10^9$ to see exactly how many strings the DP did count, it was less than $7 \cdot 10^5$.

Thus, we could potentially brute force all the strings consisting of a minimum prefix and a valid suffix, then manually remove duplicates.

This seemed most easy to do with a recursive function. We would pass the current string, as well as whether we were on the prefix or the suffix. To create a minimum prefix, all digits would have to be in increasing order with the exception of 0's, as we can't have leading 0's, thus, we can split the string into 4 parts: leading zeros + the first nonzero digit, zeros, the rest of the prefix (the first nonzero digit concatenated with this part must form a sorted string), and the suffix.

After putting all valid strings into an array, you can then create a new array a consisting of all x which are the smallest x that generate their $S(x)$ using a set to check duplicates. Each query can then be answered with a simple binary search. If m is the number valid strings, then our code runs in $O(m(\log m + \log n))$ precomputation and $O(\log m)$ per test case.