Code Jam 2021 - Round 1C

Analysis: Roaring Years

Test Set 1

A straightforward approach

Notice that 1234567 is a roaring year that happens after every possible input. That means that if we simply check every year after $\mathbf{Y}$ to see if it is roaring or not, we need to check no more than 1234567 years per case. It is a lot less in practice, but that is also harder to prove.

To check whether a year y is roaring, we can check for each prefix p of y whether "completing it" by appending $p+1$, $p+2$, etc., would work. Since only prefixes that are at most half the length of y can work, this is a small number of things to check. If we are unconvinced about the running time, we can simply run a check of roaringness for every integer up to 1234567 once initially and remember it, after which each test case is simply looking for the next "true" in a rather small table, making the running time mostly independent of the actual input, so a test run can help convince us that it will run in time without risking penalty with a real submission.

An approach with a more clear time complexity

Another way to go is to first use the observation at the end of the solution above: the length of the first number in the concatenation is small — at most 3 digits. We can try each of the 999 possibilities and concatenate numbers to it until it goes above the 1234567 threshold. Notice that this is at most 6. This builds a set of all possible roaring years that matter in very little time. After that, we can use linear search in this very small list (less than 6000 elements) to find the smallest roaring year that is larger than $\mathbf{Y}$ for each test case. Of course, we could use binary search to make each test case run even faster, but that is not required.

Test Set 2

None of the approaches above work for Test Set 2. The gap between two consecutive roaring years can easily be large (for example, you can prove as an exercise that there are no other roaring years between 100000000100000001 and 100000001100000002) and checking an individual one, while quick, takes a bit more time than before. For the second approach, there are now $10^9 - 1$ candidates for the first number in the concatenation, and the amount of concatenated numbers can also be much larger for some of those, making the full set of roaring years too big to even fit in memory, not to mention the computation time.

A case-based approach

The last sentence of the previous paragraph hints at an idea. While the full set of roaring years in range is too big, the full set of roaring years that are the concatenation of 3 or more numbers is not: for that case, the maximum possible starting number has 6 digits. We can use the second Test Set 2 solution with this change to solve for those. We still need to check roaring years that are the concatenation of exactly two consecutive numbers. Notice that $f(x) =$ the concatenation of x and $x+1$ is an increasing function. Therefore, we can use the bisection method to efficiently find the minimum of that subset of roaring years that is above $\mathbf{Y}$. After we found the best candidate from each subset, we simply return the smallest of them.

A general approach

We can also generalize the second case of our first approach for Test Set 2. The family of functions $f_n(x) =$ concatenation of $x, x+1, \ldots, x+n-1$ are all increasing. We can therefore use bisection to find the best candidate for each possible amount of consecutive numbers n, and then answer the best of those. Since n is at most $\log \mathbf{Y}$ this yields a method that runs in $O(\log^2 \mathbf{Y})$ time, which means it can work for really large bounds.