Documentation

To solve the problem of counting the occurrences of the digit "1" in all numbers from (0) to (n), we can utilize an efficient mathematical approach based on positional analysis rather than counting the occurrences of "1" individually in each number. The core idea is to assess how frequently "1" appears at each digit place (units, tens, hundreds, etc.) across all numbers up to (n), leveraging patterns that emerge with each digit place's cycle. By calculating contributions for each digit position, we avoid iterating through every number up to (n), which would be too slow for large inputs, especially when (n) reaches values up to (10^9).

To calculate the contribution of each position systematically, we break down (n) into three parts relative to the position of interest: the digits to the left (higher), the digit at the current position (current), and the digits to the right (lower). This breakdown allows us to observe how "1" behaves at each place in different numbers, forming patterns that repeat based on the digit place. For example, the unit place will encounter a "1" every ten numbers, the tens place every hundred numbers, and so forth. By leveraging these patterns, we can calculate the number of times "1" appears without checking each number individually.

The contribution of each place depends on the value of the current digit in that place. If the current digit is zero, the count of "1"s in that position depends solely on the higher digits. If the current digit is exactly one, both the higher and lower digits contribute to the final count, reflecting numbers that are influenced by the current position. When the current digit is greater than one, we observe that the number of "1"s in that position is higher due to a complete cycle contributed by the higher digits. This differentiation allows us to calculate the exact contribution for each digit place based on the three parts, summing up contributions for all digit places.

With this approach, the algorithm efficiently iterates over the logarithmic positions of (n), as each place (from units to the highest place in (n)) is evaluated in constant time. This results in a time complexity of (O(log_{10}(n))), making the solution highly efficient for large inputs. Unlike a brute-force solution, this approach only requires a few mathematical operations per position, which saves both time and memory, ensuring the solution remains within feasible bounds even for large (n) values.

In summary, this solution relies on recognizing cyclical patterns in the occurrence of the digit "1" at each positional level. By leveraging the split of (n) into higher, current, and lower digits, we compute contributions based on a pattern-driven, constant-time evaluation per position, achieving an optimal solution. This method illustrates how mathematical analysis of positional values can streamline what might otherwise be an intractable counting problem, making it both practical and efficient for large values of (n).