Remove K Digits - Complete Documentation

Problem Statement

The problem requires removing k digits from a given non-negative integer (represented as a string) to obtain the smallest possible number. The output must not contain leading zeros unless the result is "0". The constraints ensure that the input size can be large, making efficiency a key factor in the solution.

Intuition

The goal is to remove digits strategically so that the remaining sequence forms the smallest number. A simple left-to-right scan combined with a **monotonic increasing stack** helps achieve this efficiently. By removing digits that create a larger value when followed by a smaller one, we ensure the smallest possible number is retained.

Key Observations

A digit should be removed if it is larger than the next digit, as this reduction makes the overall number smaller. If k digits are still left to be removed after a full pass through the number, they should be removed from the end. Additionally, leading zeros should be stripped from the final result to maintain correct formatting.

Approach

The best method to solve this problem efficiently is to use a **monotonic stack**. As we iterate through the digits of num, we maintain a stack where digits are stored in an increasing order. If the current digit is smaller than the top of the stack, we remove the top digit (pop) until k digits have been removed. If k digits remain after iterating through the number, they are removed from the rightmost part of the stack. Finally, the result is constructed by converting the stack into a string while ensuring that any leading zeros are removed. If the final result is an empty string, we return "0".

Edge Cases

Several edge cases must be considered while implementing the solution. If the digits are already in increasing order, we simply remove the last k digits. Removing any k digits still results in a number with the same repeating digits if all digits are identical. If leading zeros appear in the final result after removal, they must be stripped to prevent incorrect formatting. Additionally, when k equals the length of num, removing all digits results in an empty string, which must be converted to "0". Lastly, if the input contains a single digit and k is 1, removing it should return "0".

Complexity Analysis

The **time complexity** of this approach is **O(N)**, where N is the length of num, since each digit is pushed and popped from the stack at most once. This ensures a linear-time solution, making it highly efficient. The **space complexity** is also **O(N)** in the worst case, where all digits are stored in the stack. However, in most cases, the stack will only store N-k elements, making the space usage manageable.

Alternative Approaches

A brute-force approach would involve generating all possible numbers after removing k digits and selecting the smallest one, but this is impractical due to its exponential complexity. Another method is a **greedy left-to-right scan**, where we iteratively remove the largest peak in the number until k digits are removed. However, this approach can be inefficient for larger inputs, making the stack-based method the optimal solution.

Final Thoughts

This problem is best solved using a **monotonic stack**, which efficiently maintains a sorted sequence of digits while ensuring that removals are made optimally. The approach runs in **O(N) time**, making it suitable for large constraints. Special care is taken to handle **leading zeros, removing all digits, and single-digit inputs**. Alternative approaches exist, but they are either too slow or inefficient compared to the stack-based method. The proposed solution is optimal, scalable, and ensures that the smallest possible number is returned after k deletions.