2259. Remove Digit From Number to Maximize Result

Easy Greedy String Enumeration

Problem Description

In this problem, you have a string `number` which represents a positive integer, and a character `digit` which is guaranteed to appear at least once in `number`. Your task is to remove exactly one occurrence of `digit` from `number`, such that the new string still represents a positive integer and is as large as possible. The challenge lies in determining which occurrence of the `digit` to remove in order to maximize the resulting integer.

Intuition

To arrive at the solution for this problem, it is beneficial to understand that the value of a digit in a number is dependent on its position. Specifically, the closer a digit is to the start of the number, the greater its impact on the number's overall value. Therefore, to maximize the result, we should prefer to remove a digit that is earlier in the string if it would lead to a larger subsequent digit being moved one position to the left.

The presented code iterates through the `number` string and checks each occurrence of `digit`. The core idea is to find the first instance of `digit` which is followed by a larger digit. When this pattern is found, removing the `digit` would result in increasing the overall number by having the larger digit take a more significant place. If this situation is not encountered, the code defaults to removing the last occurrence of `digit`, because removing a digit closer to the end of the number has the least impact on the number's value.

Thus, by scanning left to right and leveraging the significance of digit positions, we can decide on the optimal digit to remove to maximize the integer's value.

Solution Approach

The implementation of the solution uses a simple for-loop to iterate over each character in the string `number`. The primary data structure used here is the string itself, as we are only interested in reading its characters without the need for additional data structures.

Here is the step-by-step approach of the algorithm:

1. Initialize a variable `last` with `-1`, which will keep track of the index of the `digit` to be removed.
2. Determine the length of `number` and store it in the variable `n`.
3. Loop through each character `d` in `number` using its index `i`. For each character: a. Check if `d` equals `digit`. If it does: b. Update `last` with the current index `i`. c. If this is not the last character in the string, and if the current `digit` is less than the character following it, break the loop.
4. After exiting the loop, return the resulting string by concatenating the substring of `number` before the index `last` and the substring of `number` after index `last`. We skip `last` itself to "remove" the digit.

This approach leverages pattern recognition – specifically, that removing a lower digit before a higher digit will maximize the resulting number. Furthermore, it falls back on the greedy algorithm principle where local choices are made (removing the first `digit` before a larger digit) in hopes of finding a global optimum – the largest possible number after removal.

This methodology guarantees that the most significant (and the first removable) `digit` that leads to an increase in value is removed. The simplicity and efficiency of iterating once through the string make this solution optimal for the given task.

Example Walkthrough

Let's use a small example to illustrate the solution approach.

Suppose the `number` given is "2736" and the `digit` we need to remove is '3'. Following the solution approach described:

1. We initialize `last` with `-1`. This variable will eventually hold the index of the '3' we choose to remove.
2. We determine that the length of `number` (`n`) is 4.
3. We loop through each character `d` in "2736" using its index `i`:
   - At index `i = 0`, `d` is '2'. It is not equal to '3', so we continue.
   - At index `i = 1`, `d` is '7'. Again, it is not '3', so we move on.
   - At index `i = 2`, `d` is '3'. This matches our `digit`. We update `last` to 2. Now, we look ahead to the next character.
   - The character following '3' is '6', which is greater than '3'. This means removing '3' from this position will maximize our result, as '6' will take a more significant place. We break the loop.
4. Exiting the loop, we now know the index at `last` (2 in this case) is where we want to remove our `digit`. We return the resulting string by concatenating the substring before index `last` ("27") with the substring after index `last` ("6"), effectively skipping the '3'. The result is "276".

By this approach, we have successfully removed one instance of '3' from the `number` "2736" to get the largest possible new number, which is "276". The algorithm smartly picked the '3' that preceded a larger number, thus ensuring the maximization of the resulting integer.

Solution Implementation

Python

```python
class Solution:
    def removeDigit(self, number: str, digit_to_remove: str) -> str:
        # Initialize variable to keep track of the position where
        # the last occurrence of the digit to remove is found
        last_occurrence_index = -1
        # Calculate the length of the input number string
        number_length = len(number)
        # Loop through each character in the number string by index and value
        for index, digit in enumerate(number):
            # If the current digit is the one we want to remove, update the last occurrence index
            if digit == digit_to_remove:
                last_occurrence_index = index
                # Check if there's a next digit and if it's greater than the current digit,
                # in which case, we break out of the loop to remove this particular occurrence
                if index + 1 < number_length and digit < number[index + 1]:
                    break
        # Return the number string with the digit removed at the last occurrence index
        # This concatenation skips the digit to remove
        return number[:last_occurrence_index] + number[last_occurrence_index + 1:]
```

Java

```java
class Solution {
    public String removeDigit(String number, char digit) {
        int lastIndexOccurrence = -1; // Initialize the last index of the digit to be removed
        int stringLength = number.length(); // Get the length of the number string

        // Iterate through each character of the string
        for (int i = 0; i < stringLength; ++i) {
            char currentDigit = number.charAt(i); // Get the character at the current index

            // Check if the current character matches the digit we want to remove
            if (currentDigit == digit) {
                lastIndexOccurrence = i; // Update the last index occurrence of the digit

                // If the digit to remove is smaller than the next digit,
                // and there is a next digit, break the loop
                if (i + 1 < stringLength && currentDigit < number.charAt(i + 1)) {
                    break;
                }
            }
        }

        // Remove the digit at the last index occurrence and return the new string
        return number.substring(0, lastIndexOccurrence) + number.substring(lastIndexOccurrence + 1);
    }
}
```

C++

```cpp
class Solution {
public:
    /**
     * Remove a single occurrence of the digit in the string such that the result is the largest possible number.
     *
     * @param number The string representation of the number.
     * @param digit The digit to remove.
     * @return The result string after the digit is removed.
     */
    string removeDigit(string number, char digit) {
        int numSize = number.size(); // Get the size of the number string.
        int lastOccurrence = -1; // Track the last occurrence index of the digit.

        // Iterate through the string.
        for (int i = 0; i < numSize; ++i) {
            char currentDigit = number[i]; // Store the current digit.

            // Check if the current digit matches the digit we want to remove.
            if (currentDigit == digit) {
                lastOccurrence = i; // Update the last occurrence index.

                // Check if removing the current digit makes the number larger
                // by comparing it with the next digit.
                if (i + 1 < numSize && number[i] < number[i + 1]) {
                    // If the next digit is larger, break the loop as we've found the optimal digit to remove.
                    break;
                }
            }
        }

        // Remove the digit from the number using the last occurrence found.
        // Form a new string without the digit at the last occurrence index.
        return number.substr(0, lastOccurrence) + number.substr(lastOccurrence + 1);
    }
};
```

TypeScript

```typescript
/**
 * Removes the first occurrence of a given digit that is followed by a larger digit,
 * or removes the last occurrence of the digit if no such condition is met.
 *
 * @param {string} number - The original number represented as a string.
 * @param {string} digit - The digit to be removed from the number.
 * @returns {string} - The modified number as a string after removing the specified digit.
 */
function removeDigit(number: string, digit: string): string {
    // Determine the length of the number string.
    const numberLength: number = number.length;

    // Initialize an index to store the position of the digit to be removed.
    let lastOccurrenceIndex: number = -1;

    // Iterate through each character in the number string.
    for (let i = 0; i < numberLength; ++i) {
        // Check if the current character is the digit we want to remove.
        if (number[i] === digit) {
            // Update the last occurrence index of the digit to the current index.
            lastOccurrenceIndex = i;

            // Check if the current digit is followed by a larger digit.
            if (i + 1 < numberLength && number[i] < number[i + 1]) {
                // If so, break the loop as we've found the optimal digit to remove.
                break;
            }
        }
    }

    // Combine the parts of the string before and after the digit to be removed,
    // effectively removing the digit from the number.
    return number.substring(0, lastOccurrenceIndex) + number.substring(lastOccurrenceIndex + 1);
}
```

```python
class Solution:
    def removeDigit(self, number: str, digit_to_remove: str) -> str:
        # Initialize variable to keep track of the position where
        # the last occurrence of the digit to remove is found
        last_occurrence_index = -1
        # Calculate the length of the input number string
        number_length = len(number)
        # Loop through each character in the number string by index and value
        for index, digit in enumerate(number):
            # If the current digit is the one we want to remove, update the last occurrence index
            if digit == digit_to_remove:
                last_occurrence_index = index
                # Check if there's a next digit and if it's greater than the current digit,
                # in which case, we break out of the loop to remove this particular occurrence
                if index + 1 < number_length and digit < number[index + 1]:
                    break
        # Return the number string with the digit removed at the last occurrence index
        # This concatenation skips the digit to remove
        return number[:last_occurrence_index] + number[last_occurrence_index + 1:]
```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the string `number`. This is because the code involves a single for-loop over the string `number`, where each iteration performs a constant amount of work.

The space complexity of the given code is `O(n)`. This is due to the string slicing operation `number[:last] + number[last + 1 :]`, which creates a new string that can be of length up to `n`. Even though Python strings are immutable and slicing creates a new string, it is essential to note that slicing may leverage copy-on-write under the hood, where the actual complexity can depend on the Python implementation. However, for complexity analysis, it is safe to state that in the worst case the space complexity is linear with respect to the length of the input string.