2710. Remove Trailing Zeros From a String

Easy String

Problem Description

The LeetCode problem requires writing a function to remove any trailing zeros from a string representation of a positive integer. Essentially, given a string that contains numeric characters ending with one or more zeros, the task is to return a new string where these trailing zeros are stripped off.

For example, if the input string is `num = "1234000"`, after removing the trailing zeros, the output should be `"1234"`.

It is important to note that only the zeros at the end of the string (trailing) need to be removed. Any zeros that appear before other numeric characters must remain. For instance, in the case of `num = "102030"`, the output should still be `"102030"` as there are no trailing zeros.

Intuition

The process of getting to the solution for removing trailing zeros from a string representing a positive integer is quite straightforward. In Python, strings are immutable sequences of characters that come with a built-in method called `rstrip()`. The `rstrip()` method is used to remove any trailing characters (characters at the end of a string) that match the specified set of characters.

Therefore, by passing the character `"0"` to the `rstrip()` method, we can effectively strip off all the trailing zeros from the string. This is done without affecting any other part of the string, which is crucial for maintaining the integer's representation without leading and middle zeros being affected.

Solution Approach

The implementation of the solution is extremely simple due to Python's built-in string methods. The solution utilizes the `rstrip()` method which is inherently designed for the exact purpose of removing trailing characters from a string.

Here's a step-by-step breakdown of the solution:

- The `rstrip()` method is called on the input string `num`.
- The argument `"0"` is passed to `rstrip()`, which tells the function to remove all occurrences of the character `"0"` from the end of the string `num`.
- The `rstrip()` method works its way from the end of the string towards the start, stopping as soon as it encounters a character that is not `"0"`. Thus, it will not affect any zeros that are not trailing zeros.
- The result of this operation is a new string with all the trailing zeros removed. This new string is then returned as the output of the function.

This approach doesn't require the use of additional data structures or complex algorithms. It simply leverages Python's ability to handle string manipulations efficiently.

In terms of complexity, the time complexity is O(n), where n is the length of the input string. This is because, in the worst-case scenario, the method has to check each character in the string once (when there are no trailing zeros to remove). The space complexity is O(1) because no additional storage is needed regardless of the size of the input.

The reference solution code looks like this:

```
class Solution:
    def removeTrailingZeros(self, num: str) -> str:
        return num.rstrip("0")
```

This concise implementation effectively solves the problem while being highly readable and leveraging Python's strengths for string manipulation.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach described above. Suppose we have as our input the following string representing a positive integer:

```
num = "1050000"
```

This string ends with trailing zeros, which we want to remove. According to our solution approach, we'll use the `rstrip()` method in Python, targeting the character `"0"`. Below are the steps:

1. We call `rstrip("0")` on the string `num`. The string is `num = "1050000"`.
2. `rstrip()` starts checking characters from the end of the string and removes the first `"0"`, updating the string to `num = "105000"`.
3. It continues to strip each trailing `"0"` until it encounters a character that is not a `"0"`, which in this case is the character `"5"`.
4. As it stops at the first non-zero character while scanning from right to left, the remaining string is now `num = "105"`.
5. The new string with the trailing zeros removed, `"105"`, is immediately ready to be returned from the function.

The reference solution code is simple and effective here:

```
class Solution:
    def removeTrailingZeros(self, num: str) -> str:
        return num.rstrip("0")
```

If we had to call the method using our example string, it would look like this:

```
solution = Solution()
output = solution.removeTrailingZeros("1050000")
print(output)  # Output: "105"
```

The output `"105"` is the string representation of the initial number with the trailing zeros removed, which is the expected result. This walkthrough highlights the ease and efficiency of the solution approach using Python's `rstrip()` method.

Solution Implementation

Python

```
class Solution:
    def remove_trailing_zeros(self, num: str) -> str:
        # Remove trailing zeros from a numeric string
        # 'rstrip' method removes all occurrences of given characters from the end of a string.
        # Here it removes all '0' characters from the right side of the num string.
        return num.rstrip('0')
```

Java

```
class Solution {
    public String removeTrailingZeros(String num) {
        // Start from the end of the string and move backwards
        int i = num.length() - 1;

        // Loop until a non-zero character is found
        while (i >= 0 && num.charAt(i) == '0') {
            i--; // Decrement the index to move one character left
        }

        // Return the substring from the beginning to the last non-zero character
        // If there were no zeros at the end, this returns the original string
        return num.substring(0, i + 1);
    }
}
```

C++

```
#include <string> // Includes the string library, which allows us to use the std::string type

class Solution {
public:
    // Function to remove trailing zeros from a string representing a number
    string removeTrailingZeros(string num) {
        // Loop that continues as long as the last character in the string is a '0'
        while (!num.empty() && num.back() == '0') {
            num.pop_back(); // Removes the last character from the string
        }
        return num; // Returns the modified string with trailing zeros removed
    }
};
```

TypeScript

```
/**
 * Removes trailing zeros from a numeric string.
 *
 * @param {string} num - The numeric string to be processed.
 * @returns {string} The numeric string with trailing zeros removed.
 */
function removeTrailingZeros(num: string): string {
    // Initialize an index variable starting from the end of the string
    let index = num.length - 1;

    // Loop backwards through the string until a non-zero character is found
    while (num[index] === '0') {
        --index; // Decrement the index to move to the previous character
    }

    // Return the substring from the start of the string up to the last non-zero character
    return num.substring(0, index + 1);
}
```

```
class Solution:
    def remove_trailing_zeros(self, num: str) -> str:
        # Remove trailing zeros from a numeric string
        # 'rstrip' method removes all occurrences of given characters from the end of a string.
        # Here it removes all '0' characters from the right side of the num string.
        return num.rstrip('0')
```

Time and Space Complexity

The time complexity of the function `removeTrailingZeros` is `O(n)`, where `n` is the length of the input string `num`. This is because `rstrip` iterates through the string from the end and stops when it encounters a non-zero character, making it linear with respect to the number of characters it needs to check.

The space complexity of the function is `O(1)`, as it does not allocate any additional space that is dependent on the input size. The `rstrip` function returns a new string, but the space used by this new string does not depend on the input size because it is at most the same size as the input.