58. Length of Last Word

Easy String

Problem Description

The problem requires us to find the length of the last word in a given string `s`. The string consists of English letters and spaces (`' '`). Words can be considered as sequences of characters separated by one or more spaces. Importantly, we're only interested in the length of the last word, not the word itself. This means that we need to traverse the string to separate the last word from possible trailing spaces and then measure its length.

Intuition

To solve this problem, we realize that the last word in the string could possibly be followed by spaces. If we start scanning from the end of the string, the first non-space character we encounter marks the end of the last word. We record this position as `i`. We then continue scanning backwards till we find the first space character or reach the beginning of the string; this indicates the position right before the start of the last word, which we denote as `j`. So the length of the last word is simply the difference between `i` and `j`.

Now, let's go step-by-step to reach this solution:

1. Start scanning from the end of the string to bypass any trailing spaces if they exist.
2. Once a non-space character is found, this is the last character of the last word; mark this position `i`.
3. Continue scanning backwards until we find a space character or the beginning of the string; this marks the position before the start of the last word, `j`.
4. Calculate the length of the last word as `i - j`.

This approach works because we need to find only the last word, which allows us to use a reverse scan, thus eliminating the need to process the entire string or the other words. As a result, we obtain a solution that is efficient in both time and space.

Solution Approach

The implementation of the solution is direct and utilize a simple algorithm without needing any additional data structures. We directly process the string itself using two pointers.

Let's describe the steps that correspond to the provided Python code:

1. Initialize `i` to point to the last character of the string `s`. This is done by setting `i = len(s) - 1`.
2. Use a `while` loop to decrement `i` until we find a non-space character. This loop skips any trailing spaces that might be at the end of the string `s`. The condition `i >= 0` ensures we don't go out of bounds if the entire string is composed of spaces.
3. Once the loop finds a non-space character, `i` points to the last character of the last word. We then use `j = i` to mark the end of the last word.
4. Another `while` loop is used to continue moving `j` backwards (`j -= 1`) until it finds a space character or until we reach the start of the string. This gives us the position just before the start of the last word.
5. Finally, we calculate the length of the last word using `i - j`. Since `j` is the position before the first character of the last word, `i - j` correctly gives the length of the last word.

No additional data structures are necessary because the solution modifies the existing input only by keeping count of the current index with the pointers `i` and `j`, and does not modify the string itself.

The pattern used in this solution is known as the **two pointers technique**, which is often used in array and string manipulation problems to track or compare elements from the start/end or both.

Considering the complexity:

- The time complexity is `O(n)` because we potentially have to traverse the entire string if the last word is at the beginning or if there's only one word without leading or trailing spaces.
- Space complexity is `O(1)`, meaning that the space used by the algorithm does not grow with the input size. We only use a finite amount of additional memory for the pointers and condition checking.

Here is the approach with the code blocks:

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        # Initialize i to the last index of the string
        i = len(s) - 1

        # Skip any trailing spaces
        while i >= 0 and s[i] == ' ':
            i -= 1

        # i now points to the last character of the last word
        j = i

        # Find the space before the start of the last word
        while j >= 0 and s[j] != ' ':
            j -= 1

        # The length of the last word is the difference between i and j
        return i - j
```

Using these pointers, the solution efficiently calculates the length of the last word in the input string.

Example Walkthrough

Let's consider a string `s` with the value "Hello LeetCode ".

1. First, we initialize `i` to the index of the last character in `s`. Thus, `i = len("Hello LeetCode ") - 1`, which means `i = 14`.
2. Starting from the end of the string, we run a `while` loop that decreases `i` until we find a non-space character. Since the last character is a space, we decrement `i`. It continues until `i = 12` which is right before the space (pointing to 'e').
3. At this stage `s[i]` is not a space, so we move out of the first `while` loop. We set `j = i` to mark the end of the last word, which in this case is `j = 12`.
4. The second `while` loop starts decreasing `j` until we find a space character or reach the beginning of the string. Traversing backward, the space at index 5 is found, so `j` becomes 5. We've now identified the position right before the start of the last word.
5. Finally, we calculate the length of the last word as the difference between `i` and `j`, plus one to account for the indexing (since `j` is right before the first character of the last word). Therefore, the length is `12 - 5`, resulting in `7`.

Following this approach, we can say the last word, "LeetCode", has a length of 7 characters without using additional data structures or traversing unnecessary parts of the string.

Solution Implementation

Python

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        # Start from the end of the string and find the index of the first non-space character
        end_index = len(s) - 1
        while end_index >= 0 and s[end_index] == ' ':
            end_index -= 1

        # Find the beginning of the word (the index of the space before the word or -1 if the word is at the start)
        start_index = end_index
        while start_index >= 0 and s[start_index] != ' ':
            start_index -= 1

        # Return the length of the last word, which is the difference between the end and start indices
        # Add 1 because end_index is the last character of the last word and
        # start_index is the space before the last word (or -1 if the word is at the start)
        return end_index - start_index

# Example usage:
# sol = Solution()
# length = sol.lengthOfLastWord("Hello World  ")
# print(length)  # Should print 5, which is the length of "World"
```

Java

```
class Solution {
    // Method to find the length of the last word in a string.
    public int lengthOfLastWord(String s) {
        // Initialize index 'endIndex' to point to the end of the string.
        int endIndex = s.length() - 1;

        // Skip all the trailing spaces if any.
        while (endIndex >= 0 && s.charAt(endIndex) == ' ') {
            endIndex--;
        }

        // Initialize index 'startIndex' to keep track of the start of the last word.
        int startIndex = endIndex;

        // Move 'startIndex' backwards until we find a space or reach the beginning of the string.
        while (startIndex >= 0 && s.charAt(startIndex) != ' ') {
            startIndex--;
        }

        // The length of the last word is the difference between 'endIndex' and 'startIndex'.
        // We add 1 because 'startIndex' is either pointing to a space or one position off the string.
        return endIndex - startIndex;
    }
}
```

C++

```
#include <string>  // Include the string library to use the std::string class

class Solution {
public:
    // Function to calculate the length of the last word in a string
    int lengthOfLastWord(std::string s) {
        // Initialize 'index' to the last character of the string
        int index = s.size() - 1;

        // Skip the trailing spaces if there are any
        while (index >= 0 && s[index] == ' ') {
            --index; // Move backwards in the string
        }

        // Memorize the position of the end of the last word (after skipping trailing spaces)
        int endOfWordIndex = index;

        // Find the beginning of the last word
        while (index >= 0 && s[index] != ' ') {
            --index; // Continue moving backwards until we find a space or reach the beginning of the string
        }

        // Calculate the length of the last word
        // The length is the difference between the position of the end of the word and
        // the position of the beginning of the word (or -1 if the word starts at the beginning of the string)
        int lengthOfLastWord = endOfWordIndex - index;

        // Return the length of the last word
        return lengthOfLastWord;
    }
};
```

TypeScript

```
function lengthOfLastWord(s: string): number {
    // Initialize the index to the last character of the string
    let endIndex = s.length - 1;

    // Move the index backwards to skip any trailing spaces
    while (endIndex >= 0 && s[endIndex] === ' ') {
        endIndex--;
    }

    // If the entire string was spaces, return 0
    if (endIndex < 0) {
        return 0;
    }

    // Initialize the start index to the position of the endIndex
    let startIndex = endIndex;

    // Move the startIndex backwards until a space is encountered or the start of the string
    while (startIndex >= 0 && s[startIndex] !== ' ') {
        startIndex--;
    }

    // The length of the last word is the difference between the endIndex and startIndex
    return endIndex - startIndex;
}

// The function can be tested with the following examples:
console.log(lengthOfLastWord("Hello World")); // Output: 5
console.log(lengthOfLastWord(" ")); // Output: 0
console.log(lengthOfLastWord("a ")); // Output: 1
```

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        # Start from the end of the string and find the index of the first non-space character
        end_index = len(s) - 1
        while end_index >= 0 and s[end_index] == ' ':
            end_index -= 1

        # Find the beginning of the word (the index of the space before the word or -1 if the word is at the start)
        start_index = end_index
        while start_index >= 0 and s[start_index] != ' ':
            start_index -= 1

        # Return the length of the last word, which is the difference between the end and start indices
        # Add 1 because end_index is the last character of the last word and
        # start_index is the space before the last word (or -1 if the word is at the start)
        return end_index - start_index

# Example usage:
# sol = Solution()
# length = sol.lengthOfLastWord("Hello World  ")
# print(length)  # Should print 5, which is the length of "World"
```

Time and Space Complexity

Time Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the string `s`. This is because the algorithm consists of a reverse traversal for both leading spaces and the length of the last word. Each traversal individually takes at worst case `O(n)` time (if the string is either only spaces or has no spaces). However, these two loops are traversed sequentially, not nested, so the overall time complexity remains `O(n)`.

Space Complexity

The space complexity of the code is `O(1)`. This is because no additional space that scales with input size is being used. The variables `i` and `j` are used for indexing and do not depend on the size of the input string, hence they use a constant amount of space.