434. Number of Segments in a String

Easy String

Problem Description

The problem provides you with a string `s` and asks you to find the number of segments in this string. Here, a segment is considered to be a continuous sequence of characters that does not include any spaces. This means that words separated by one or more spaces are counted as distinct segments. The task is to count these segments and return the number.

For example, in the string `"Hello, my name is John"`, there are five segments: "Hello,", "my", "name", "is", "John".

Intuition

The intuition for solving this problem is based on the definition of a segment. Since a segment is defined as a contiguous sequence of non-space characters, we can look for transitions from a space to a non-space character to identify the start of a segment.

The following points form the basis of the intuition:

- If the current character is not a space, it might be a part of a segment.
- To confirm if it's the start of a new segment, check the character that comes before it. If the preceding character is a space (or if there is no preceding character, as when the current character is the first in the string), then it's the start of a new segment.
- Increment the segment count each time we spot the start of a new segment.

Solution Approach

The solution to this problem is implemented in a very straightforward manner, without the use of complex algorithms or data structures. It uses a simple loop and conditionals to evaluate the characters of the string one by one.

Here's a breakdown of the implementation:

- Initialize a counter variable `ans` with a value of 0.
- Loop through the string `s` using the `enumerate` function, which gives us both the index `i` and the character `c` for each iteration.
- Inside the loop, check if the current character `c` is not a space. Since we are only interested in non-space characters, we ignore spaces.
- Then, check if it's the first character in the string (`i == 0`) or if the preceding character is a space (`s[i - 1] == ' '`).
- If either condition is true, it signifies the start of a new segment, and the counter `ans` should be incremented by 1.
- Continue this process until you have examined all the characters in the string.
- After the loop concludes, return the value of `ans` as it now contains the total count of segments in the string `s`.

The algorithm effectively uses a finite state machine pattern where you're only changing the state (incrementing the counter) when certain conditions (state transitions) are met, that is—from space to non-space. It operates in linear time complexity (O(n)), where n is the length of the string `s`, because it examines each character exactly once.

Example Walkthrough

Let's illustrate the solution approach with a small example string `s = " Algo Expert "`. We want to count the number of segments (i.e., words) in this string. Remember, a segment is defined as a contiguous sequence of non-space characters.

Following the solution approach:

1. Initialize `ans` to 0 because we haven't counted any segments yet.
2. Start looping through the string using `enumerate` to get both index `i` and character `c`.
3. Iteration 1: `i = 0`, `c = ' '`. Since `c` is a space, we skip it.
4. Iteration 2: `i = 1`, `c = ' '`. Another space, we skip it again.
5. Iteration 3: `i = 2`, `c = 'A'`. This character is not a space, and since the preceding character is a space, it signifies the start of a new segment. We increment `ans` to 1.
6. Iterations 4-10: These iterations deal with characters `l`, `g`, `o`, , `E`, `x`, `p`. Except these characters, none starts a new segment since they're either part of the current segment or they're space followed by another space.
7. Iteration 11: `i = 10`, `c = 'E'`. This isn't a space. The preceding character is a space, so we've found a new segment. Increment `ans` to 2.
8. Continue iterating through the rest of the string. We find no more starting points for a segment since all remaining characters are either part of an ongoing segment or are trailing spaces.
9. By the end, we've identified that `ans = 2`, which means there are two segments in our example string `s`.

This walk-through demonstrates how the algorithm intelligently counts segments in a string by spotting those specific transitions from space to non-space characters (and also accounts for the first character if it's not a space). The count is then returned as the output.

Solution Implementation

Python

```python
class Solution:
    def count_segments(self, s: str) -> int:
        # Initialize a counter for the number of segments
        segment_count = 0

        # Iterate through the string character by character along with their index
        for index, char in enumerate(s):
            # Check if the character is not a space, and it's the start of a segment
            # A new segment starts either at the beginning of the string (index == 0)
            # or just after a space character (s[index - 1] == ' ')
            if char != ' ' and (index == 0 or s[index - 1] == ' '):
                segment_count += 1  # Increment the segment count

        # Return the total number of segments found
        return segment_count
```

Java

```java
class Solution {
    public int countSegments(String s) {
        // Initialize a variable to hold the count of segments
        int segmentCount = 0;

        // Iterate over each character in the string
        for (int i = 0; i < s.length(); ++i) {
            // Check if the current character is not a space and if it is either the first character
            // or the character before it was a space (indicating the start of a new segment)
            if (s.charAt(i) != ' ' && (i == 0 || s.charAt(i - 1) == ' ')) {
                // Increment the segment count
                ++segmentCount;
            }
        }

        // Return the total count of segments in the string
        return segmentCount;
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to count the number of segments in a string,
    // where a segment is defined as a contiguous sequence of non-space characters.
    int countSegments(string s) {
        int segmentCount = 0; // Initialize a count of segments to 0

        // Loop through each character of the string
        for (int i = 0; i < s.size(); ++i) {
            // Check if the current character is not a space character
            // and it is either the first character or preceded by a space
            if (s[i] != ' ' && (i == 0 || s[i - 1] == ' ')) {
                ++segmentCount;  // If true, increment the segment count
            }
        }

        // Return the total number of segments found in the string
        return segmentCount;
    }
};
```

TypeScript

```typescript
// Function to count the number of segments in a string,
// where a segment is defined as a contiguous sequence of non-space characters.
function countSegments(s: string): number {
    let segmentCount = 0; // Initialize a count of segments to 0

    // Loop through each character of the string
    for (let i = 0; i < s.length; i++) {
        // Check if the current character is not a space character
        // and it is either the first character or preceded by a space
        if (s[i] !== ' ' && (i === 0 || s[i - 1] === ' ')) {
            segmentCount++;  // If true, increment the segment count
        }
    }

    // Return the total number of segments found in the string
    return segmentCount;
}


class Solution:
    def count_segments(self, s: str) -> int:
        # Initialize a counter for the number of segments
        segment_count = 0

        # Iterate through the string character by character along with their index
        for index, char in enumerate(s):
            # Check if the character is not a space, and it's the start of a segment
            # A new segment starts either at the beginning of the string (index == 0)
            # or just after a space character (s[index - 1] == ' ')
            if char != ' ' and (index == 0 or s[index - 1] == ' '):
                segment_count += 1  # Increment the segment count

        # Return the total number of segments found
        return segment_count
```

Time and Space Complexity

The time complexity of the provided code snippet is `O(n)`, where `n` is the length of the string `s`. This is because the code iterates once over all the characters in the string to count the number of segments. The conditional checks inside the loop are all `O(1)` operations, and they do not change the overall linear time complexity.

As for the space complexity, the provided code snippet uses `O(1)` extra space. The variable `ans` is the only additional space used that holds the count of segments. The space used does not grow with the size of the input string `s`, which means it is constant space complexity.