Documentation of Number of Segments in a String

Table of Contents

1. Problem Statement

Given a string s, return the number of segments in the string.

A segment is defined as a contiguous sequence of non-space characters.

Constraints

- 0≤s.length≤300
- s consists of:
    - Lowercase and uppercase English letters
    - Digits
    - Special characters: !@#$%^&*()_+-=',.:
    - The only space character in s is ' '

Examples

Example 1

Input: s = "Hello, my name is John"

Output:5

Explanation:The five segments are ["Hello,", "my", "name", "is", "John"].

Example 2

Input: s = "Hello"

Output: 1

2. Intuition

- The problem requires counting contiguous sequences of non-space characters.
- The split() function in Python efficiently breaks a string into words based on whitespace and removes extra spaces automatically.
- The number of segments can be directly obtained by counting the resulting list elements.

3. Key Observations

- Consecutive spaces should not be counted as segments.
- An empty string ("") should return 0.
- A string consisting only of spaces should also return 0.
- Single words and punctuations are considered valid segments.

4. Approach

1. Use Python's built-in .split() function, which:
    - Automatically splits a string by whitespace.
    - Removes extra spaces between words.
2. Compute the number of segments by getting the length of the resulting list.

Formula: len(s.split())

5. **Edge Cases**

Case	Input	Expected Output
Empty String	""	0
Single Word	"Hello"	1
Multiple Spaces	" "	0
Trailing Spaces	"Hello "	1
Multiple Words	"Hello, my name is John"	5
Extra Spaces Between	"Hello, World!"	2

6. **Complexity Analysis**

Time Complexity

- Splitting the String: O(n)O(n), where nn is the length of s.
- Computing Length: O(1)O(1).
- Overall Complexity: O(n)O(n).

Space Complexity

- In the worst case, if all characters are non-space, the split list stores O(n)O(n) words.
- Overall Complexity: O(n)O(n).

7. **Alternative Approaches**

Approach 1: Manual Iteration

Instead of using split(), we can manually traverse the string, counting segments when transitioning from a space to a non-space character.

Pros:

✅ Space-efficient (O(1)O(1) space complexity).

Cons:

❌ More complex to implement.

❌ Requires handling multiple edge cases.

8. **Test Cases**

```
sol = Solution()

# Test Case 1: General case with multiple words
assert sol.countSegments("Hello, my name is John") == 5

# Test Case 2: Single word
assert sol.countSegments("Hello") == 1

# Test Case 3: Empty string
assert sol.countSegments("") == 0

# Test Case 4: String with only spaces
assert sol.countSegments("   ") == 0

# Test Case 5: String with multiple spaces between words
assert sol.countSegments("Hello,    World!") == 2

# Test Case 6: String with trailing spaces
assert sol.countSegments("Hello   ") == 1

print("All test cases passed!")
```

9. **Final Thoughts**

- Using split() is the simplest and most efficient way to count segments in a string.
- The approach is robust and automatically handles edge cases.
- Alternative approaches (manual iteration) may save space but increase complexity.

✅ **Recommended Approach:** Use split().