557. Reverse Words in a String III

Easy Two Pointers String

Problem Description

The problem presents a task where we are given a string `s` that represents a sentence. The goal is to reverse the sequence of characters in each individual word contained within the sentence, while maintaining the original order of the words and preserving the spaces between them. We are not required to reverse the entire sentence, but only individual words within it.

Intuition

The intuition behind the solution is straightforward. We break down the problem into smaller, logical steps. These steps are:

1. Split the given string `s` into individual words. A word is defined as a sequence of characters separated by spaces. We use the `split()` method in Python, which splits a string by the specified delimiter, in this case, a space (`' '`).
2. Reverse each word individually. In Python, this is easily accomplished using slicing with `[::-1]`. This syntax creates a reversed copy of the sequence.
3. Join the reversed words back together with a space (`' '`) as a separator to form a new string that represents the sentence with each word's characters in reverse order. We do this using the `join()` method, which combines all the elements of an iterable (like a list of words) into a single string, separated by the specified string (a space in our case).

The provided solution encapsulates this intuition into a one-liner by combining all the steps using a list comprehension and the `join()` method.

Solution Approach

The implementation of the solution utilizes a combination of string methods and list comprehensions, which are powerful features of Python's standard library. The step-by-step approach is as follows:

1. **Splitting the String**: The first step in the algorithm is to take the input string `s` which represents the sentence and split it into a list of words. This is accomplished with `s.split(' ')`. The split method takes a delimiter, which in this case is a space (`' '`), and returns a list of substrings.
2. **Reversing Each Word**: Next, we need to reverse each word within the list. Python's slicing syntax can reverse a sequence when used with the step parameter set to -1 (`[::-1]`). We use a list comprehension to apply this reversal to each word in the list, resulting in `[t[::-1] for t in s.split(' ')]`. This piece of code iterates over each word `t` in the list of words created from the original sentence and creates a new list with each word reversed.
3. **Joining the Words**: Finally, we need to reassemble the sentence with each word's characters reversed but maintaining the original word order. The `join()` method is used to concatenate the elements of the list into a single string. `' '.join(...)` takes each element of the iterable provided (the list of reversed words) and joins them into a single string separated by spaces. This restores the structure of the original sentence with spaces intact except that each word is now reversed.

The combination of `split()`, slicing, list comprehension, and `join()` methods provide a concise and efficient solution to the problem. Data structures like lists assist with managing collections of words, and the built-in string operations in Python handle the manipulation required without the need for more complex algorithms or patterns.

Example Walkthrough

Let's take a simple sentence: `"Hello World"`. Our task is to reverse the characters in each word. Following the solution approach, we'd perform the following steps:

1. **Splitting the String**: We start by splitting the string into individual words using `s.split(' ')`.

   Input Sentence: `"Hello World"`

   After splitting: `["Hello", "World"]`

2. **Reversing Each Word**: Using Python's slicing feature, we reverse each word in the list.

   Reversed words: `["olleH", "dlroW"]`

   Mechanism: For string `"Hello"`, `"Hello"[::-1]` becomes `"olleH"`; for `"World"`, `"World"[::-1]` becomes `"dlroW"`.

3. **Joining the Words**: With all words reversed, we join them back together with spaces between them to form a coherent sentence.

   Final Output: `"olleH dlroW"`

   Method used: `' '.join(["olleH", "dlroW"])`

The complete Python expression for this would be `' '.join([word[::-1] for word in "Hello World".split(' '))`, which would evaluate to the expected output: `"olleH dlroW"`.

Solution Implementation

Python

```python
class Solution:
    def reverseWords(self, s: str) -> str:
        # Split the input string into words using a space as the delimiter.
        words = s.split(' ')

        # Reverse each word in the list of words.
        reversed_words = [word[::-1] for word in words]

        # Join the reversed words back into a string with spaces between them.
        reversed_sentence = ' '.join(reversed_words)

        # Return the reversed sentence.
        return reversed_sentence
```

Java

```java
class Solution {
    public String reverseWords(String s) {
        // StringBuilder to accumulate the final result
        StringBuilder result = new StringBuilder();

        // Splitting the input string s into words separated by spaces
        String[] words = s.split(" ");

        // Iterating through each word in the array
        for (String word : words) {
            // For each word, reverse it by iterating from the end to the start
            for (int i = word.length() - 1; i >= 0; --i) {
                // Append each character to result
                result.append(word.charAt(i));
            }
            // After reversing, add a space to separate the words
            result.append(" ");
        }

        // Return the string representation of result
        // We exclude the last space using substring to match the required output
        return result.substring(0, result.length() - 1);
    }
}
```

C++

```cpp
#include <algorithm> // Include algorithm library for std::reverse
#include <string>    // Include string library for std::string

class Solution {
public:
    // Function to reverse each word in a string.
    string reverseWords(string s) {
        // Iterate over the entire string.
        for (int start = 0, len = s.size(); start < len; ++start) {
            if (s[start] == ' ') continue; // Skip spaces.

            // Find the end of the current word.
            int end = start;
            while (end < len && s[end] != ' ')
                ++end;

            // Reverse the current word.
            std::reverse(s.begin() + start, s.begin() + end);

            // Move the start to the end of the current word.
            start = end;
        }

        return s; // Return the modified string.
    }
};
```

TypeScript

```typescript
// Function to reverse each word in a given string s.
function reverseWords(s: string): string {
    // Split the string by one or more whitespace characters.
    return s
        .split(/\s+/)
        // Map each word in the array to its reversed form.
        .map((word: string) => {
            // Initialize an empty string to store the reversed word.
            let reversedWord = '';
            // Iterate over each character in the word.
            for (const char of word) {
                // Prepend the character to the reversedWord to reverse the word.
                reversedWord = char + reversedWord;
            }
            // Return the reversed word.
            return reversedWord;
        })
        // Join the reversed words back into a single string with spaces.
        .join(' ');
}
```

```python
class Solution:
    def reverseWords(self, s: str) -> str:
        # Split the input string into words using a space as the delimiter.
        words = s.split(' ')

        # Reverse each word in the list of words.
        reversed_words = [word[::-1] for word in words]

        # Join the reversed words back into a string with spaces between them.
        reversed_sentence = ' '.join(reversed_words)

        # Return the reversed sentence.
        return reversed_sentence
```

Time and Space Complexity

Time Complexity

The time complexity of the code mainly consists of two parts: splitting the string into words and reversing individual words.

1. Splitting the string into words using `s.split(' ')` has a time complexity of `O(n)` where `n` is the length of the string `s`. This is because we have to go through the entire string to find the spaces to split it into words.
2. Reversing each word and joining them happens in the list comprehension. Reversing a word of length `k` takes `O(k)` time, and we do this for each word. Assuming `m` words result from the split, and the average length of a word is `k`, the total time for this operation is `O(m*k)`.

Assuming `k` is roughly equal to `n/m` (i.e., the average length of a word is proportional to the total length of the string divided by the number of words), the overall time complexity would be `O(n + m*(n/m))` which simplifies to `O(n + n)` and finally to `O(n)`.

Space Complexity

The space complexity of the code can be broken down as follows:

1. The `s.split(' ')` operation creates a list of words, which takes `O(m)` space, where `m` is the number of words.
2. The list comprehension `[t[::-1] for t in s.split(' ')]` generates a new list that contains the reversed words. This list also takes `O(m)` space.
3. When these words are joined to form the final string, this string would have the same length as the input string, `O(n)` space.

Hence, if we consider the space required for the input `s`, output string, and the intermediate list of words, the space complexity is `O(n + m + m)`. Since we know that `2m` is less than `n` (as `m` is the number of words, and there must be at least one character per word), we can approximate the space complexity to be `O(n)`.

The final space complexity is thus `O(n)`.