Reverse Each Word in a String GFG

You are given a string containing words separated by periods. Your task is to reverse the order of the words in the string, keeping the periods in their respective places.

Example: Input: "This.program.is.to...reverse..words.of.the.given.string"

Output: "sihT.margorp.si.ot...esrever..sdrow.fo.eht.nevig.gnirts"

Approach 1: Reverses the words in the input string using a vector

- Iterate through each character in the string and build each word character by character.
- When a period is encountered, add the word and a period to the **wordsArray**.
- Finally, construct the reversed string by appending the words in **wordsArray** in reverse order.
- **Time Complexity: O(n), where n is the length of the input string.**
- **Space Complexity: O(n), where n is the length of the input string.**

Approach 2: Reverses the words in the input string using a stack

- Iterate through each character in the string and build each word character by character.
- When a period is encountered, push the word and a period onto the **wordStack**.
- Finally, construct the reversed string by popping words from the **wordStack** and appending them to the **reversedString**.
- **Time Complexity: O(n), where n is the length of the input string.**
- **Space Complexity: O(n), where n is the length of the input string.**

Approach 3: Reverses the words in the input string using another string

- Iterate through each character in the string and build each word character by character.
- When a period is encountered, add periods to the **word** string.
- Append the reversed word to the **reversedString** and clear the **word** for the next iteration.
- **Time Complexity: O(n), where n is the length of the input string.**

- **Space Complexity: O(n), where n is the length of the input string.**

Which approach to use:

- Overall, all three approaches have similar time and space complexity. The choice depends on personal preference and the specific requirements of the problem.
- If memory usage is a concern, Approach 2 (using a stack) and Approach 3 (using another string) have slightly better space complexity, as they only store the words without periods. However, the difference in space complexity is negligible unless the input string is extremely large.
- In terms of code efficiency, Approach 2 and Approach 3 avoid unnecessary operations and have some optimizations compared to Approach 1. Among the three, Approach 2 using a stack is generally considered more efficient due to its simplicity and effectiveness in reversing the words.