Problem 151: Reverse Words in a String

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given an input string

s

, reverse the order of the

words

.

A

word

is defined as a sequence of non-space characters. The

words

in

s

will be separated by at least one space.

Return

a string of the words in reverse order concatenated by a single space.

Note

that

s

may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Example 1:

Input:

s = "the sky is blue"

Output:

"blue is sky the"

Example 2:

Input:

s = " hello world "

Output:

"world hello"

Explanation:

Your reversed string should not contain leading or trailing spaces.

Example 3:

Input:

s = "a good example"

Output:

"example good a"

Explanation:

You need to reduce multiple spaces between two words to a single space in the reversed string.

Constraints:

1 <= s.length <= 10

4

s

contains English letters (upper-case and lower-case), digits, and spaces

' '

.

There is

at least one

word in

s

.

Follow-up:

If the string data type is mutable in your language, can you solve it

in-place

with

O(1)

extra space?

Code Snippets

C++:

```
class Solution {
public:
string reverseWords(string s) {

}
};
```

Java:

```
class Solution {
public String reverseWords(String s) {

}
}
```

Python3:

```
class Solution:
def reverseWords(self, s: str) -> str:
```

Python:

```
class Solution(object):
def reverseWords(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
var reverseWords = function(s) {

};
```

TypeScript:

```
function reverseWords(s: string): string {

};
```

C#:

```
public class Solution {
public string ReverseWords(string s) {

}
}
```

C:

```
char* reverseWords(char* s) {

}
```

Go:

```
func reverseWords(s string) string {

}
```

Kotlin:

```
class Solution {
fun reverseWords(s: String): String {

}
}
```

Swift:

```
class Solution {
func reverseWords(_ s: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn reverse_words(s: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @return {String}
def reverse_words(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
*/
function reverseWords($s) {

}
}
```

Dart:

```
class Solution {
String reverseWords(String s) {

}
```

```
}
```

Scala:

```
object Solution {
def reverseWords(s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec reverse_words(s :: String.t) :: String.t
def reverse_words(s) do

end
end
```

Erlang:

```
-spec reverse_words(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
reverse_words(S) ->
.
```

Racket:

```
(define/contract (reverse-words s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Words in a String
* Difficulty: Medium
* Tags: array, string
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string reverseWords(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Words in a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String reverseWords(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse Words in a String
Difficulty: Medium
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
```

```
"""

class Solution:
def reverseWords(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def reverseWords(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Reverse Words in a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @return {string}
*/
var reverseWords = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Words in a String
* Difficulty: Medium
```

```
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function reverseWords(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Reverse Words in a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string ReverseWords(string s) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Words in a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
char* reverseWords(char* s) {

}
```

Go Solution:

```
// Problem: Reverse Words in a String
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func reverseWords(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun reverseWords(s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func reverseWords(_ s: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Reverse Words in a String
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
```

```
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn reverse_words(s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String}
def reverse_words(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function reverseWords($s) {

}
}
```

Dart Solution:

```
class Solution {
String reverseWords(String s) {

}
}
```

Scala Solution:

```
object Solution {
def reverseWords(s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec reverse_words(s :: String.t) :: String.t
def reverse_words(s) do

end
end
```

Erlang Solution:

```
-spec reverse_words(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
reverse_words(S) ->
.
```

Racket Solution:

```
(define/contract (reverse-words s)
(-> string? string?)
)
```