Problem 1592: Rearrange Spaces Between Words

Problem Information

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

Problem Description

You are given a string

text

of words that are placed among some number of spaces. Each word consists of one or more lowercase English letters and are separated by at least one space. It's guaranteed that

text

contains at least one word

.

Rearrange the spaces so that there is an

equal

number of spaces between every pair of adjacent words and that number is

maximized

. If you cannot redistribute all the spaces equally, place the

extra spaces at the end

, meaning the returned string should be the same length as

text

.

Return

the string after rearranging the spaces

.

Example 1:

Input:

text = " this is a sentence "

Output:

"this is a sentence"

Explanation:

There are a total of 9 spaces and 4 words. We can evenly divide the 9 spaces between the words: 9 / (4-1) = 3 spaces.

Example 2:

Input:

text = " practice makes perfect"

Output:

"practice makes perfect "

Explanation:

There are a total of 7 spaces and 3 words. 7 / (3-1) = 3 spaces plus 1 extra space. We place this extra space at the end of the string.

Constraints:

1 <= text.length <= 100

text

consists of lowercase English letters and

' '

.

text

contains at least one word.

Code Snippets

C++:

```
class Solution {
public:
string reorderSpaces(string text) {

}
};
```

Java:

```
class Solution {
public String reorderSpaces(String text) {

}
}
```

Python3:

```
class Solution:
def reorderSpaces(self, text: str) -> str:
```

Python:

```
class Solution(object):
def reorderSpaces(self, text):
"""
:type text: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} text
* @return {string}
*/
var reorderSpaces = function(text) {

};
```

TypeScript:

```
function reorderSpaces(text: string): string {

};
```

C#:

```
public class Solution {
public string ReorderSpaces(string text) {

}
}
```

C:

```
char* reorderSpaces(char* text) {

}
```

Go:

```
func reorderSpaces(text string) string {

}
```

Kotlin:

```
class Solution {
fun reorderSpaces(text: String): String {

}
}
```

Swift:

```
class Solution {
func reorderSpaces(_ text: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn reorder_spaces(text: String) -> String {

}
}
```

Ruby:

```
# @param {String} text
# @return {String}
def reorder_spaces(text)

end
```

PHP:

```
class Solution {

/**
* @param String $text
* @return String
```

```
*/
function reorderSpaces($text) {

}
}
```

Dart:

```
class Solution {
String reorderSpaces(String text) {

}
}
```

Scala:

```
object Solution {
def reorderSpaces(text: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec reorder_spaces(text :: String.t) :: String.t
def reorder_spaces(text) do

end
end
```

Erlang:

```
-spec reorder_spaces(Text :: unicode:unicode_binary()) ->
unicode:unicode_binary().
reorder_spaces(Text) ->
.
```

Racket:

```
(define/contract (reorder-spaces text)
(-> string? string?)
```

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rearrange Spaces Between Words
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string reorderSpaces(string text) {

}
};
```

Java Solution:

```
/**
* Problem: Rearrange Spaces Between Words
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String reorderSpaces(String text) {

}
}
```

Python3 Solution:

```
"""
Problem: Rearrange Spaces Between Words
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

```
class Solution(object):
def reorderSpaces(self, text):
"""
:type text: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Rearrange Spaces Between Words
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} text
* @return {string}
*/
```

```
var reorderSpaces = function(text) {

};
```

TypeScript Solution:

```
/**
* Problem: Rearrange Spaces Between Words
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function reorderSpaces(text: string): string {

};
```

C# Solution:

```
/*
* Problem: Rearrange Spaces Between Words
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string ReorderSpaces(string text) {

}
}
```

C Solution:

```c
/*
* Problem: Rearrange Spaces Between Words
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* reorderSpaces(char* text) {

}
```

Go Solution:

```go
// Problem: Rearrange Spaces Between Words
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func reorderSpaces(text string) string {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun reorderSpaces(text: String): String {

}
}
```

Swift Solution:

```swift
class Solution {
func reorderSpaces(_ text: String) -> String {

}
```

```
}
```

Rust Solution:

```
// Problem: Rearrange Spaces Between Words
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn reorder_spaces(text: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} text
# @return {String}
def reorder_spaces(text)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $text
* @return String
*/
function reorderSpaces($text) {

}
}
```

Dart Solution:

```
class Solution {
String reorderSpaces(String text) {

}
}
```

Scala Solution:

```
object Solution {
def reorderSpaces(text: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec reorder_spaces(text :: String.t) :: String.t
def reorder_spaces(text) do

end
end
```

Erlang Solution:

```
-spec reorder_spaces(Text :: unicode:unicode_binary()) ->
unicode:unicode_binary().
reorder_spaces(Text) ->
.
```

Racket Solution:

```
(define/contract (reorder-spaces text)
(-> string? string?)
)
```