Problem 1324: Print Words Vertically

Problem Information

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

Problem Description

Given a string

s

. Return all the words vertically in the same order in which they appear in

s

.

Words are returned as a list of strings, complete with spaces when is necessary. (Trailing spaces are not allowed).

Each word would be put on only one column and that in one column there will be only one word.

Example 1:

Input:

s = "HOW ARE YOU"

Output:

["HAY","ORO","WEU"]

Explanation:

Each word is printed vertically. "HAY" "ORO" "WEU"

Example 2:

Input:

s = "TO BE OR NOT TO BE"

Output:

["TBONTB","OEROOE"," T"]

Explanation:

Trailing spaces is not allowed. "TBONTB" "OEROOE" " T"

Example 3:

Input:

s = "CONTEST IS COMING"

Output:

["CIC","OSO","N M","T I","E N","S G","T"]

Constraints:

1 <= s.length <= 200

s

contains only upper case English letters.

It's guaranteed that there is only one space between 2 words.

Code Snippets

C++:

```
class Solution {
public:
vector<string> printVertically(string s) {

}
};
```

Java:

```
class Solution {
public List<String> printVertically(String s) {

}
}
```

Python3:

```
class Solution:
def printVertically(self, s: str) -> List[str]:
```

Python:

```
class Solution(object):
def printVertically(self, s):
"""
:type s: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string[]}
*/
var printVertically = function(s) {

};
```

TypeScript:

```
function printVertically(s: string): string[] {

};
```

C#:

```
public class Solution {
public IList<string> PrintVertically(string s) {

}
}
```

C:

```


/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char ** printVertically(char * s, int* returnSize){

}
```

Go:

```
func printVertically(s string) []string {

}
```

Kotlin:

```
class Solution {
fun printVertically(s: String): List<String> {

}
}
```

Swift:

```
class Solution {
func printVertically(_ s: String) -> [String] {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn print_vertically(s: String) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} s
# @return {String[]}
def print_vertically(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String[]
*/
function printVertically($s) {

}
}
```

Scala:

```
object Solution {
def printVertically(s: String): List[String] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Print Words Vertically
* 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:
vector<string> printVertically(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Print Words Vertically
* 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 List<String> printVertically(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Print Words Vertically
```

```
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 printVertically(self, s: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def printVertically(self, s):
"""
:type s: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Print Words Vertically
* 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 printVertically = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Print Words Vertically
* 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 printVertically(s: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Print Words Vertically
* 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 IList<string> PrintVertically(string s) {

}
}
```

C Solution:

```
/*
* Problem: Print Words Vertically
* 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
*/



/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char ** printVertically(char * s, int* returnSize){

}
```

Go Solution:

```
// Problem: Print Words Vertically
// 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 printVertically(s string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun printVertically(s: String): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func printVertically(_ s: String) -> [String] {

}
```

```
}
```

Rust Solution:

```
// Problem: Print Words Vertically
// 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 print_vertically(s: String) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String[]}
def print_vertically(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String[]
*/
function printVertically($s) {

}
}
```

Scala Solution:

```
object Solution {
def printVertically(s: String): List[String] = {

}
}
```