Problem 68: Text Justification

Problem Information

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

Problem Description

Given an array of strings

words

and a width

maxWidth

, format the text such that each line has exactly

maxWidth

characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces

' '

when necessary so that each line has exactly

maxWidth

characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left-justified, and no extra space is inserted between words.

Note:

A word is defined as a character sequence consisting of non-space characters only.

Each word's length is guaranteed to be greater than

0

and not exceed

maxWidth

.

The input array

words

contains at least one word.

Example 1:

Input:

words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16

Output:

["This is an", "example of text", "justification. "]

Example 2:

Input:

words = ["What","must","be","acknowledgment","shall","be"], maxWidth = 16

Output:

["What must be", "acknowledgment ", "shall be "]

Explanation:

Note that the last line is "shall be " instead of "shall be", because the last line must be left-justified instead of fully-justified. Note that the second line is also left-justified because it contains only one word.

Example 3:

Input:

words = ["Science","is","what","we","understand","well","enough","to","explain","to","a","computer.","Art","is","everything","else","we","do"], maxWidth = 20

Output:

["Science is what we", "understand well", "enough to explain to", "a computer. Art is", "everything else we", "do "]

Constraints:

1 <= words.length <= 300

1 <= words[i].length <= 20

words[i]

consists of only English letters and symbols.

1 <= maxWidth <= 100

words[i].length <= maxWidth

Code Snippets

C++:

```
class Solution {
public:
vector<string> fullJustify(vector<string>& words, int maxWidth) {

}
};
```

Java:

```
class Solution {
public List<String> fullJustify(String[] words, int maxWidth) {

}
}
```

Python3:

```
class Solution:
def fullJustify(self, words: List[str], maxWidth: int) -> List[str]:
```

Python:

```
class Solution(object):
def fullJustify(self, words, maxWidth):
"""
:type words: List[str]
:type maxWidth: int
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @param {number} maxWidth
* @return {string[]}
*/
```

```
var fullJustify = function(words, maxWidth) {

};
```

TypeScript:

```
function fullJustify(words: string[], maxWidth: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> FullJustify(string[] words, int maxWidth) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** fullJustify(char** words, int wordsSize, int maxWidth, int*
returnSize) {

}
```

Go:

```
func fullJustify(words []string, maxWidth int) []string {

}
```

Kotlin:

```
class Solution {
fun fullJustify(words: Array<String>, maxWidth: Int): List<String> {

}
}
```

Swift:

```
class Solution {
func fullJustify(_ words: [String], _ maxWidth: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn full_justify(words: Vec<String>, max_width: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {Integer} max_width
# @return {String[]}
def full_justify(words, max_width)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param Integer $maxWidth
* @return String[]
*/
function fullJustify($words, $maxWidth) {

}
}
```

Dart:

```
class Solution {
List<String> fullJustify(List<String> words, int maxWidth) {
```

```
}
}
```

Scala:

```
object Solution {
def fullJustify(words: Array[String], maxWidth: Int): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec full_justify(words :: [String.t], max_width :: integer) :: [String.t]
def full_justify(words, max_width) do

end
end
```

Erlang:

```
-spec full_justify(Words :: [unicode:unicode_binary()], MaxWidth ::
integer()) -> [unicode:unicode_binary()].
full_justify(Words, MaxWidth) ->
.
```

Racket:

```
(define/contract (full-justify words maxWidth)
(-> (listof string?) exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Text Justification
```

```
* Difficulty: Hard
* Tags: array, string, greedy
*
* 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> fullJustify(vector<string>& words, int maxWidth) {

}
};
```

Java Solution:

```
/**
* Problem: Text Justification
* Difficulty: Hard
* Tags: array, string, greedy
*
* 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> fullJustify(String[] words, int maxWidth) {

}
}
```

Python3 Solution:

```
"""
Problem: Text Justification
Difficulty: Hard
Tags: array, string, greedy

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 fullJustify(self, words: List[str], maxWidth: int) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def fullJustify(self, words, maxWidth):
"""
:type words: List[str]
:type maxWidth: int
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Text Justification
* Difficulty: Hard
* Tags: array, string, greedy
*
* 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[]} words
* @param {number} maxWidth
* @return {string[]}
*/
var fullJustify = function(words, maxWidth) {

};
```

TypeScript Solution:

```
/**
* Problem: Text Justification
* Difficulty: Hard
* Tags: array, string, greedy
*
* 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 fullJustify(words: string[], maxWidth: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Text Justification
* Difficulty: Hard
* Tags: array, string, greedy
*
* 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> FullJustify(string[] words, int maxWidth) {

}
}
```

C Solution:

```
/*
* Problem: Text Justification
* Difficulty: Hard
* Tags: array, string, greedy
*
* 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** fullJustify(char** words, int wordsSize, int maxWidth, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Text Justification
// Difficulty: Hard
// Tags: array, string, greedy
//
// 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 fullJustify(words []string, maxWidth int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun fullJustify(words: Array<String>, maxWidth: Int): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func fullJustify(_ words: [String], _ maxWidth: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Text Justification
// Difficulty: Hard
// Tags: array, string, greedy
//
// 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 full_justify(words: Vec<String>, max_width: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {Integer} max_width
# @return {String[]}
def full_justify(words, max_width)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param Integer $maxWidth
* @return String[]
*/
function fullJustify($words, $maxWidth) {

}
}
```

Dart Solution:

```
class Solution {
List<String> fullJustify(List<String> words, int maxWidth) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def fullJustify(words: Array[String], maxWidth: Int): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec full_justify(words :: [String.t], max_width :: integer) :: [String.t]
def full_justify(words, max_width) do

end
end
```

Erlang Solution:

```
-spec full_justify(Words :: [unicode:unicode_binary()], MaxWidth ::
integer()) -> [unicode:unicode_binary()].
full_justify(Words, MaxWidth) ->
.
```

Racket Solution:

```
(define/contract (full-justify words maxWidth)
(-> (listof string?) exact-integer? (listof string?))
)
```