Problem 418: Sentence Screen Fitting

Problem Information

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

Problem Description

Given a

rows x cols

screen and a

sentence

represented as a list of strings, return

the number of times the given sentence can be fitted on the screen

.

The order of words in the sentence must remain unchanged, and a word cannot be split into two lines. A single space must separate two consecutive words in a line.

Example 1:

Input:

sentence = ["hello","world"], rows = 2, cols = 8

Output:

1

Explanation:

hello--- world--- The character '-' signifies an empty space on the screen.

Example 2:

Input:

sentence = ["a", "bcd", "e"], rows = 3, cols = 6

Output:

2

Explanation:

a-bcd- e-a--- bcd-e- The character '-' signifies an empty space on the screen.

Example 3:

Input:

sentence = ["i","had","apple","pie"], rows = 4, cols = 5

Output:

1

Explanation:

i-had apple pie-i had-- The character '-' signifies an empty space on the screen.

Constraints:

1 <= sentence.length <= 100

1 <= sentence[i].length <= 10

sentence[i]

consists of lowercase English letters.

1 <= rows, cols <= 2 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int wordsTyping(vector<string>& sentence, int rows, int cols) {

}
};
```

Java:

```
class Solution {
public int wordsTyping(String[] sentence, int rows, int cols) {

}
}
```

Python3:

```
class Solution:
def wordsTyping(self, sentence: List[str], rows: int, cols: int) -> int:
```

Python:

```
class Solution(object):
def wordsTyping(self, sentence, rows, cols):
"""
:type sentence: List[str]
:type rows: int
:type cols: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {string[]} sentence
* @param {number} rows
* @param {number} cols
* @return {number}
*/
var wordsTyping = function(sentence, rows, cols) {

};
```

TypeScript:

```
function wordsTyping(sentence: string[], rows: number, cols: number): number
{

};
```

C#:

```
public class Solution {
public int WordsTyping(string[] sentence, int rows, int cols) {

}
}
```

C:

```
int wordsTyping(char** sentence, int sentenceSize, int rows, int cols) {

}
```

Go:

```
func wordsTyping(sentence []string, rows int, cols int) int {

}
```

Kotlin:

```
class Solution {
fun wordsTyping(sentence: Array<String>, rows: Int, cols: Int): Int {

}
}
```

Swift:

```
class Solution {
func wordsTyping(_ sentence: [String], _ rows: Int, _ cols: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn words_typing(sentence: Vec<String>, rows: i32, cols: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} sentence
# @param {Integer} rows
# @param {Integer} cols
# @return {Integer}
def words_typing(sentence, rows, cols)

end
```

PHP:

```
class Solution {

/**
* @param String[] $sentence
* @param Integer $rows
* @param Integer $cols
* @return Integer
*/
function wordsTyping($sentence, $rows, $cols) {
```

```
}
}
```

Dart:

```
class Solution {
int wordsTyping(List<String> sentence, int rows, int cols) {

}
}
```

Scala:

```
object Solution {
def wordsTyping(sentence: Array[String], rows: Int, cols: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec words_typing(sentence :: [String.t], rows :: integer, cols :: integer)
:: integer
def words_typing(sentence, rows, cols) do

end
end
```

Erlang:

```
-spec words_typing(Sentence :: [unicode:unicode_binary()], Rows :: integer(),
Cols :: integer()) -> integer().
words_typing(Sentence, Rows, Cols) ->
.
```

Racket:

```
(define/contract (words-typing sentence rows cols)
(-> (listof string?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sentence Screen Fitting
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int wordsTyping(vector<string>& sentence, int rows, int cols) {

}
};
```

Java Solution:

```
/**
* Problem: Sentence Screen Fitting
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int wordsTyping(String[] sentence, int rows, int cols) {

}
}
```

Python3 Solution:

```
"""
Problem: Sentence Screen Fitting
Difficulty: Medium
Tags: array, string, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def wordsTyping(self, sentence: List[str], rows: int, cols: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def wordsTyping(self, sentence, rows, cols):
"""
:type sentence: List[str]
:type rows: int
:type cols: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Sentence Screen Fitting
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string[]} sentence
* @param {number} rows
* @param {number} cols
```

```
* @return {number}
*/
var wordsTyping = function(sentence, rows, cols) {

};
```

TypeScript Solution:

```
/**
* Problem: Sentence Screen Fitting
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function wordsTyping(sentence: string[], rows: number, cols: number): number
{

};
```

C# Solution:

```
/*
* Problem: Sentence Screen Fitting
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int WordsTyping(string[] sentence, int rows, int cols) {

}
}
```

C Solution:

```
/*
* Problem: Sentence Screen Fitting
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int wordsTyping(char** sentence, int sentenceSize, int rows, int cols) {

}
```

Go Solution:

```
// Problem: Sentence Screen Fitting
// Difficulty: Medium
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func wordsTyping(sentence []string, rows int, cols int) int {

}
```

Kotlin Solution:

```
class Solution {
fun wordsTyping(sentence: Array<String>, rows: Int, cols: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func wordsTyping(_ sentence: [String], _ rows: Int, _ cols: Int) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Sentence Screen Fitting
// Difficulty: Medium
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn words_typing(sentence: Vec<String>, rows: i32, cols: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} sentence
# @param {Integer} rows
# @param {Integer} cols
# @return {Integer}
def words_typing(sentence, rows, cols)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $sentence
* @param Integer $rows
* @param Integer $cols
* @return Integer
*/
function wordsTyping($sentence, $rows, $cols) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int wordsTyping(List<String> sentence, int rows, int cols) {

}
}
```

Scala Solution:

```
object Solution {
def wordsTyping(sentence: Array[String], rows: Int, cols: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec words_typing(sentence :: [String.t], rows :: integer, cols :: integer)
:: integer
def words_typing(sentence, rows, cols) do

end
end
```

Erlang Solution:

```
-spec words_typing(Sentence :: [unicode:unicode_binary()], Rows :: integer(),
Cols :: integer()) -> integer().
words_typing(Sentence, Rows, Cols) ->
.
```

Racket Solution:

```
(define/contract (words-typing sentence rows cols)
(-> (listof string?) exact-integer? exact-integer? exact-integer?)
```

```
)
```