Problem 2114: Maximum Number of Words Found in Sentences

Problem Information

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

Problem Description

A

sentence

is a list of

words

that are separated by a single space with no leading or trailing spaces.

You are given an array of strings

sentences

, where each

sentences[i]

represents a single

sentence

.

Return

the

maximum number of words

that appear in a single sentence

.

Example 1:

Input:

sentences = ["alice and bob love leetcode", "i think so too",

"this is great thanks very much"

]

Output:

6

Explanation:

- The first sentence, "alice and bob love leetcode", has 5 words in total. - The second sentence, "i think so too", has 4 words in total. - The third sentence, "this is great thanks very much", has 6 words in total. Thus, the maximum number of words in a single sentence comes from the third sentence, which has 6 words.

Example 2:

Input:

sentences = ["please wait",

"continue to fight"

,

"continue to win"

]

Output:

3

Explanation:

It is possible that multiple sentences contain the same number of words. In this example, the second and third sentences (underlined) have the same number of words.

Constraints:

1 <= sentences.length <= 100

1 <= sentences[i].length <= 100

sentences[i]

consists only of lowercase English letters and

' '

only.

sentences[i]

does not have leading or trailing spaces.

All the words in

sentences[i]

are separated by a single space.

Code Snippets

C++:

```
class Solution {
public:
int mostWordsFound(vector<string>& sentences) {

}
};
```

Java:

```
class Solution {
public int mostWordsFound(String[] sentences) {

}
}
```

Python3:

```
class Solution:
def mostWordsFound(self, sentences: List[str]) -> int:
```

Python:

```
class Solution(object):
def mostWordsFound(self, sentences):
"""
:type sentences: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} sentences
* @return {number}
*/
var mostWordsFound = function(sentences) {

};
```

}

TypeScript:

```
function mostWordsFound(sentences: string[]): number {

};
```

C#:

```
public class Solution {
public int MostWordsFound(string[] sentences) {

}
}
```

C:

```
int mostWordsFound(char** sentences, int sentencesSize) {

}
```

Go:

```
func mostWordsFound(sentences []string) int {

}
```

Kotlin:

```
class Solution {
fun mostWordsFound(sentences: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func mostWordsFound(_ sentences: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn most_words_found(sentences: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} sentences
# @return {Integer}
def most_words_found(sentences)

end
```

PHP:

```
class Solution {

/**
* @param String[] $sentences
* @return Integer
*/
function mostWordsFound($sentences) {

}
}
```

Dart:

```
class Solution {
int mostWordsFound(List<String> sentences) {

}
}
```

Scala:

```
object Solution {
def mostWordsFound(sentences: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_words_found(sentences :: [String.t]) :: integer
def most_words_found(sentences) do

end
end
```

Erlang:

```
-spec most_words_found(Sentences :: [unicode:unicode_binary()]) -> integer().
most_words_found(Sentences) ->
.
```

Racket:

```
(define/contract (most-words-found sentences)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Words Found in Sentences
* Difficulty: Easy
* 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:
int mostWordsFound(vector<string>& sentences) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Words Found in Sentences
* Difficulty: Easy
* 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 int mostWordsFound(String[] sentences) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Words Found in Sentences
Difficulty: Easy
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 mostWordsFound(self, sentences: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mostWordsFound(self, sentences):
"""
:type sentences: List[str]
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Words Found in Sentences
* Difficulty: Easy
* 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[]} sentences
* @return {number}
*/
var mostWordsFound = function(sentences) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Words Found in Sentences
* Difficulty: Easy
* 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 mostWordsFound(sentences: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Words Found in Sentences
* Difficulty: Easy
* 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 int MostWordsFound(string[] sentences) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Words Found in Sentences
* Difficulty: Easy
* 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
*/

int mostWordsFound(char** sentences, int sentencesSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Words Found in Sentences
// Difficulty: Easy
// 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 mostWordsFound(sentences []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun mostWordsFound(sentences: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func mostWordsFound(_ sentences: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Words Found in Sentences
// Difficulty: Easy
// 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 most_words_found(sentences: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} sentences
# @return {Integer}
def most_words_found(sentences)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $sentences
* @return Integer
*/
function mostWordsFound($sentences) {

}
}
```

Dart Solution:

```
class Solution {
int mostWordsFound(List<String> sentences) {

}
}
```

Scala Solution:

```
object Solution {
def mostWordsFound(sentences: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_words_found(sentences :: [String.t]) :: integer
def most_words_found(sentences) do

end
end
```

Erlang Solution:

```
-spec most_words_found(Sentences :: [unicode:unicode_binary()]) -> integer().
most_words_found(Sentences) ->
.
```

Racket Solution:

```
(define/contract (most-words-found sentences)
(-> (listof string?) exact-integer?)
)
```