Problem 1662: Check If Two String Arrays are Equivalent

Problem Information

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

Problem Description

Given two string arrays

word1

and

word2

, return

true

if the two arrays

represent

the same string, and

false

otherwise.

A string is

represented

by an array if the array elements concatenated

in order

forms the string.

Example 1:

Input:

word1 = ["ab", "c"], word2 = ["a", "bc"]

Output:

true

Explanation:

word1 represents string "ab" + "c" -> "abc" word2 represents string "a" + "bc" -> "abc" The strings are the same, so return true.

Example 2:

Input:

word1 = ["a", "cb"], word2 = ["ab", "c"]

Output:

false

Example 3:

Input:

word1 = ["abc", "d", "defg"], word2 = ["abcddefg"]

Output:

true

Constraints:

1 <= word1.length, word2.length <= 10

3

1 <= word1[i].length, word2[i].length <= 10

3

1 <= sum(word1[i].length), sum(word2[i].length) <= 10

3

word1[i]

and

word2[i]

consist of lowercase letters.

Code Snippets

C++:

```
class Solution {
public:
bool arrayStringsAreEqual(vector<string>& word1, vector<string>& word2) {

}
};
```

Java:

```
class Solution {
public boolean arrayStringsAreEqual(String[] word1, String[] word2) {
```

```
}
}
```

Python3:

```
class Solution:
def arrayStringsAreEqual(self, word1: List[str], word2: List[str]) -> bool:
```

Python:

```
class Solution(object):
def arrayStringsAreEqual(self, word1, word2):
"""
:type word1: List[str]
:type word2: List[str]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[]} word1
* @param {string[]} word2
* @return {boolean}
*/
var arrayStringsAreEqual = function(word1, word2) {

};
```

TypeScript:

```
function arrayStringsAreEqual(word1: string[], word2: string[]): boolean {

};
```

C#:

```
public class Solution {
public bool ArrayStringsAreEqual(string[] word1, string[] word2) {

}
```

```
}
```

C:

```
bool arrayStringsAreEqual(char** word1, int word1Size, char** word2, int
word2Size) {

}
```

Go:

```
func arrayStringsAreEqual(word1 []string, word2 []string) bool {

}
```

Kotlin:

```
class Solution {
fun arrayStringsAreEqual(word1: Array<String>, word2: Array<String>): Boolean
{

}
}
```

Swift:

```
class Solution {
func arrayStringsAreEqual(_ word1: [String], _ word2: [String]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn array_strings_are_equal(word1: Vec<String>, word2: Vec<String>) ->
bool {

}
}
```

Ruby:

```
# @param {String[]} word1
# @param {String[]} word2
# @return {Boolean}
def array_strings_are_equal(word1, word2)

end
```

PHP:

```
class Solution {

/**
* @param String[] $word1
* @param String[] $word2
* @return Boolean
*/
function arrayStringsAreEqual($word1, $word2) {

}
}
```

Dart:

```
class Solution {
bool arrayStringsAreEqual(List<String> word1, List<String> word2) {

}
}
```

Scala:

```
object Solution {
def arrayStringsAreEqual(word1: Array[String], word2: Array[String]): Boolean
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec array_strings_are_equal(word1 :: [String.t], word2 :: [String.t]) ::
boolean
```

```
def array_strings_are_equal(word1, word2) do

end
end
```

Erlang:

```
-spec array_strings_are_equal(Word1 :: [unicode:unicode_binary()], Word2 ::
[unicode:unicode_binary()]) -> boolean().
array_strings_are_equal(Word1, Word2) ->
.
```

Racket:

```
(define/contract (array-strings-are-equal word1 word2)
(-> (listof string?) (listof string?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check If Two String Arrays are Equivalent
* 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:
bool arrayStringsAreEqual(vector<string>& word1, vector<string>& word2) {

}
};
```

Java Solution:

```
/**
* Problem: Check If Two String Arrays are Equivalent
* 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 boolean arrayStringsAreEqual(String[] word1, String[] word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Check If Two String Arrays are Equivalent
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 arrayStringsAreEqual(self, word1: List[str], word2: List[str]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def arrayStringsAreEqual(self, word1, word2):
"""
:type word1: List[str]
:type word2: List[str]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check If Two String Arrays are Equivalent
* 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[]} word1
* @param {string[]} word2
* @return {boolean}
*/
var arrayStringsAreEqual = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Check If Two String Arrays are Equivalent
* 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 arrayStringsAreEqual(word1: string[], word2: string[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check If Two String Arrays are Equivalent
* 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 bool ArrayStringsAreEqual(string[] word1, string[] word2) {

}
}
```

C Solution:

```
/*
* Problem: Check If Two String Arrays are Equivalent
* 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
*/

bool arrayStringsAreEqual(char** word1, int word1Size, char** word2, int
word2Size) {

}
```

Go Solution:

```
// Problem: Check If Two String Arrays are Equivalent
// 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 arrayStringsAreEqual(word1 []string, word2 []string) bool {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun arrayStringsAreEqual(word1: Array<String>, word2: Array<String>): Boolean
{

}
}
```

Swift Solution:

```
class Solution {
func arrayStringsAreEqual(_ word1: [String], _ word2: [String]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check If Two String Arrays are Equivalent
// 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 array_strings_are_equal(word1: Vec<String>, word2: Vec<String>) ->
bool {

}
}
```

Ruby Solution:

```
# @param {String[]} word1
# @param {String[]} word2
```

```
# @return {Boolean}
def array_strings_are_equal(word1, word2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $word1
* @param String[] $word2
* @return Boolean
*/
function arrayStringsAreEqual($word1, $word2) {

}
}
```

Dart Solution:

```
class Solution {
bool arrayStringsAreEqual(List<String> word1, List<String> word2) {

}
}
```

Scala Solution:

```
object Solution {
def arrayStringsAreEqual(word1: Array[String], word2: Array[String]): Boolean
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec array_strings_are_equal(word1 :: [String.t], word2 :: [String.t]) ::
boolean
```

```
def array_strings_are_equal(word1, word2) do

end
end
```

Erlang Solution:

```
-spec array_strings_are_equal(Word1 :: [unicode:unicode_binary()], Word2 ::
[unicode:unicode_binary()]) -> boolean().
array_strings_are_equal(Word1, Word2) ->
.
```

Racket Solution:

```
(define/contract (array-strings-are-equal word1 word2)
(-> (listof string?) (listof string?) boolean?)
)
```