Problem 1897: Redistribute Characters to Make All Strings Equal

Problem Information

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

Problem Description

You are given an array of strings

words

(

0-indexed

).

In one operation, pick two

distinct

indices

i

and

j

, where

words[i]

is a non-empty string, and move

any

character from

words[i]

to

any

position in

words[j]

.

Return

true

if you can make

every

string in

words

equal

using

any

number of operations

,

and

false

otherwise

.

Example 1:

Input:

words = ["abc","aabc","bc"]

Output:

true

Explanation:

Move the first 'a' in

words[1] to the front of words[2], to make

words[1]

= "abc" and words[2] = "abc". All the strings are now equal to "abc", so return

true

.

Example 2:

Input:

words = ["ab","a"]

Output:

false

Explanation:

It is impossible to make all the strings equal using the operation.

Constraints:

1 <= words.length <= 100

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

words[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool makeEqual(vector<string>& words) {

}
};
```

Java:

```
class Solution {
public boolean makeEqual(String[] words) {

}
}
```

Python3:

```
class Solution:
def makeEqual(self, words: List[str]) -> bool:
```

Python:

```
class Solution(object):
def makeEqual(self, words):
"""
:type words: List[str]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @return {boolean}
*/
var makeEqual = function(words) {

};
```

TypeScript:

```
function makeEqual(words: string[]): boolean {

};
```

C#:

```
public class Solution {
public bool MakeEqual(string[] words) {

}
}
```

C:

```
bool makeEqual(char** words, int wordsSize) {

}
```

Go:

```
func makeEqual(words []string) bool {

}
```

Kotlin:

```
class Solution {
fun makeEqual(words: Array<String>): Boolean {

}
}
```

Swift:

```
class Solution {
func makeEqual(_ words: [String]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn make_equal(words: Vec<String>) -> bool {

}
}
```

Ruby:

```
# @param {String[]} words
# @return {Boolean}
def make_equal(words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @return Boolean
*/
function makeEqual($words) {

}
}
```

Dart:

```
class Solution {
bool makeEqual(List<String> words) {

}
}
```

Scala:

```
object Solution {
def makeEqual(words: Array[String]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec make_equal(words :: [String.t]) :: boolean
def make_equal(words) do

end
end
```

Erlang:

```
-spec make_equal(Words :: [unicode:unicode_binary()]) -> boolean().
make_equal(Words) ->
.
```

Racket:

```
(define/contract (make-equal words)
(-> (listof string?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Redistribute Characters to Make All Strings Equal
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool makeEqual(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Redistribute Characters to Make All Strings Equal
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean makeEqual(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Redistribute Characters to Make All Strings Equal
Difficulty: Easy
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def makeEqual(self, words: List[str]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def makeEqual(self, words):
"""
:type words: List[str]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Redistribute Characters to Make All Strings Equal
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} words
* @return {boolean}
*/
var makeEqual = function(words) {
};
```

TypeScript Solution:

```
/**
* Problem: Redistribute Characters to Make All Strings Equal
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function makeEqual(words: string[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Redistribute Characters to Make All Strings Equal
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool MakeEqual(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Redistribute Characters to Make All Strings Equal
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool makeEqual(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Redistribute Characters to Make All Strings Equal
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func makeEqual(words []string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun makeEqual(words: Array<String>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func makeEqual(_ words: [String]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Redistribute Characters to Make All Strings Equal
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn make_equal(words: Vec<String>) -> bool {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {Boolean}
def make_equal(words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return Boolean
*/
function makeEqual($words) {

}
}
```

Dart Solution:

```
class Solution {
bool makeEqual(List<String> words) {
}
}
```

Scala Solution:

```
object Solution {
def makeEqual(words: Array[String]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec make_equal(words :: [String.t]) :: boolean
def make_equal(words) do

end
end
```

Erlang Solution:

```
-spec make_equal(Words :: [unicode:unicode_binary()]) -> boolean().
make_equal(Words) ->
.
```

Racket Solution:

```
(define/contract (make-equal words)
(-> (listof string?) boolean?)
)
```