Problem 2423: Remove Letter To Equalize Frequency

Problem Information

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

Problem Description

You are given a

0-indexed

string

word

, consisting of lowercase English letters. You need to select

one

index and

remove

the letter at that index from

word

so that the

frequency

of every letter present in

word

is equal.

Return

true

if it is possible to remove one letter so that the frequency of all letters in

word

are equal, and

false

otherwise

.

Note:

The

frequency

of a letter

x

is the number of times it occurs in the string.

You

must

remove exactly one letter and cannot choose to do nothing.

Example 1:

Input:

word = "abcc"

Output:

true

Explanation:

Select index 3 and delete it: word becomes "abc" and each character has a frequency of 1.

Example 2:

Input:

word = "aazz"

Output:

false

Explanation:

We must delete a character, so either the frequency of "a" is 1 and the frequency of "z" is 2, or vice versa. It is impossible to make all present letters have equal frequency.

Constraints:

2 <= word.length <= 100

word

consists of lowercase English letters only.

Code Snippets

C++:

```
class Solution {
public:
bool equalFrequency(string word) {

}
};
```

Java:

```
class Solution {
public boolean equalFrequency(String word) {

}
}
```

Python3:

```
class Solution:
def equalFrequency(self, word: str) -> bool:
```

Python:

```
class Solution(object):
def equalFrequency(self, word):
"""
:type word: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} word
* @return {boolean}
*/
var equalFrequency = function(word) {

};
```

TypeScript:

```
function equalFrequency(word: string): boolean {

};
```

C#:

```
public class Solution {
public bool EqualFrequency(string word) {

}
}
```

C:

```
bool equalFrequency(char* word) {

}
```

Go:

```
func equalFrequency(word string) bool {

}
```

Kotlin:

```
class Solution {
fun equalFrequency(word: String): Boolean {

}
}
```

Swift:

```
class Solution {
func equalFrequency(_ word: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn equal_frequency(word: String) -> bool {

}
}
```

Ruby:

```
# @param {String} word
# @return {Boolean}
def equal_frequency(word)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @return Boolean
*/
function equalFrequency($word) {

}
}
```

Dart:

```
class Solution {
bool equalFrequency(String word) {

}
}
```

Scala:

```
object Solution {
def equalFrequency(word: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec equal_frequency(word :: String.t) :: boolean
def equal_frequency(word) do

end
end
```

Erlang:

```
-spec equal_frequency(Word :: unicode:unicode_binary()) -> boolean().
equal_frequency(Word) ->
.
```

Racket:

```
(define/contract (equal-frequency word)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Letter To Equalize Frequency
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool equalFrequency(string word) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Letter To Equalize Frequency
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean equalFrequency(String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Letter To Equalize Frequency
Difficulty: Easy
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def equalFrequency(self, word: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def equalFrequency(self, word):
"""
:type word: str
:rtype: bool
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Remove Letter To Equalize Frequency
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} word
* @return {boolean}
*/
var equalFrequency = function(word) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Letter To Equalize Frequency
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function equalFrequency(word: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Remove Letter To Equalize Frequency
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool EqualFrequency(string word) {

}
}
```

C Solution:

```
/*
* Problem: Remove Letter To Equalize Frequency
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool equalFrequency(char* word) {

}
```

Go Solution:

```
// Problem: Remove Letter To Equalize Frequency
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
func equalFrequency(word string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun equalFrequency(word: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func equalFrequency(_ word: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Remove Letter To Equalize Frequency
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn equal_frequency(word: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} word
# @return {Boolean}
def equal_frequency(word)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @return Boolean
*/
function equalFrequency($word) {

}
}
```

Dart Solution:

```
class Solution {
bool equalFrequency(String word) {

}
}
```

Scala Solution:

```
object Solution {
def equalFrequency(word: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec equal_frequency(word :: String.t) :: boolean
def equal_frequency(word) do

end
end
```

Erlang Solution:

```
-spec equal_frequency(Word :: unicode:unicode_binary()) -> boolean().
equal_frequency(Word) ->
.
```

Racket Solution:

```
(define/contract (equal-frequency word)
(-> string? boolean?)
)
```