

Problem 2531: Make Number of Distinct Characters Equal

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two

0-indexed

strings

word1

and

word2

.

A

move

consists of choosing two indices

i

and

j

such that

$0 \leq i < \text{word1.length}$

and

$0 \leq j < \text{word2.length}$

and swapping

`word1[i]`

with

`word2[j]`

.

Return

true

if it is possible to get the number of distinct characters in

`word1`

and

`word2`

to be equal with

exactly one

move.

Return

false

otherwise

.

Example 1:

Input:

word1 = "ac", word2 = "b"

Output:

false

Explanation:

Any pair of swaps would yield two distinct characters in the first string, and one in the second string.

Example 2:

Input:

word1 = "abcc", word2 = "aab"

Output:

true

Explanation:

We swap index 2 of the first string with index 0 of the second string. The resulting strings are word1 = "abac" and word2 = "cab", which both have 3 distinct characters.

Example 3:

Input:

```
word1 = "abcde", word2 = "fghij"
```

Output:

true

Explanation:

Both resulting strings will have 5 distinct characters, regardless of which indices we swap.

Constraints:

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

5

word1

and

word2

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
    bool isItPossible(string word1, string word2) {

    }
};
```

Java:

```
class Solution {
    public boolean isItPossible(String word1, String word2) {
```

```
}  
}
```

Python3:

```
class Solution:  
    def isItPossible(self, word1: str, word2: str) -> bool:
```

Python:

```
class Solution(object):  
    def isItPossible(self, word1, word2):  
        """  
        :type word1: str  
        :type word2: str  
        :rtype: bool  
        """
```

JavaScript:

```
/**  
 * @param {string} word1  
 * @param {string} word2  
 * @return {boolean}  
 */  
var isItPossible = function(word1, word2) {  
  
};
```

TypeScript:

```
function isItPossible(word1: string, word2: string): boolean {  
  
};
```

C#:

```
public class Solution {  
    public bool IsItPossible(string word1, string word2) {  
  
    }  
}
```

```
}
```

C:

```
bool isItPossible(char* word1, char* word2) {  
  
}
```

Go:

```
func isItPossible(word1 string, word2 string) bool {  
  
}
```

Kotlin:

```
class Solution {  
    fun isItPossible(word1: String, word2: String): Boolean {  
  
    }  
}
```

Swift:

```
class Solution {  
    func isItPossible(_ word1: String, _ word2: String) -> Bool {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn is_it_possible(word1: String, word2: String) -> bool {  
  
    }  
}
```

Ruby:

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

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

end
```

PHP:

```
class Solution {

    /**
     * @param String $word1
     * @param String $word2
     * @return Boolean
     */
    function isItPossible($word1, $word2) {

    }

}
```

Dart:

```
class Solution {
  bool isItPossible(String word1, String word2) {

  }
}
```

Scala:

```
object Solution {
  def isItPossible(word1: String, word2: String): Boolean = {

  }
}
```

Elixir:

```
defmodule Solution do
  @spec is_it_possible(word1 :: String.t, word2 :: String.t) :: boolean
  def is_it_possible(word1, word2) do

  end
end
```

```
end
```

Erlang:

```
-spec is_it_possible(Word1 :: unicode:unicode_binary(), Word2 ::  
unicode:unicode_binary()) -> boolean().  
is_it_possible(Word1, Word2) ->  
.
```

Racket:

```
(define/contract (is-it-possible word1 word2)  
  (-> string? string? boolean?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Make Number of Distinct Characters Equal  
 * Difficulty: Medium  
 * 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 isItPossible(string word1, string word2) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Make Number of Distinct Characters Equal
```

```

* Difficulty: Medium
* 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 isItPossible(String word1, String word2) {

}
}

```

Python3 Solution:

```

"""
Problem: Make Number of Distinct Characters Equal
Difficulty: Medium
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 isItPossible(self, word1: str, word2: str) -> bool:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def isItPossible(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: bool
"""

```

JavaScript Solution:

```
/**
 * Problem: Make Number of Distinct Characters Equal
 * Difficulty: Medium
 * 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} word1
 * @param {string} word2
 * @return {boolean}
 */
var isItPossible = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
 * Problem: Make Number of Distinct Characters Equal
 * Difficulty: Medium
 * 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 isItPossible(word1: string, word2: string): boolean {

};
```

C# Solution:

```
/*
 * Problem: Make Number of Distinct Characters Equal
 * Difficulty: Medium
 * 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 IsItPossible(string word1, string word2) {

}
}

```

C Solution:

```

/*
* Problem: Make Number of Distinct Characters Equal
* Difficulty: Medium
* 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 isItPossible(char* word1, char* word2) {

}

```

Go Solution:

```

// Problem: Make Number of Distinct Characters Equal
// Difficulty: Medium
// 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 isItPossible(word1 string, word2 string) bool {

}

```

Kotlin Solution:

```
class Solution {  
    fun isItPossible(word1: String, word2: String): Boolean {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func isItPossible(_ word1: String, _ word2: String) -> Bool {  
  
    }  
}
```

Rust Solution:

```
// Problem: Make Number of Distinct Characters Equal  
// Difficulty: Medium  
// 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 is_it_possible(word1: String, word2: String) -> bool {  
  
    }  
}
```

Ruby Solution:

```
# @param {String} word1  
# @param {String} word2  
# @return {Boolean}  
def is_it_possible(word1, word2)  
  
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param String $word1  
     * @param String $word2  
     * @return Boolean  
     */  
    function isItPossible($word1, $word2) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    bool isItPossible(String word1, String word2) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def isItPossible(word1: String, word2: String): Boolean = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec is_it_possible(word1 :: String.t, word2 :: String.t) :: boolean  
    def is_it_possible(word1, word2) do  
  
    end  
end
```

Erlang Solution:

```
-spec is_it_possible(Word1 :: unicode:unicode_binary(), Word2 ::  
unicode:unicode_binary()) -> boolean().  
is_it_possible(Word1, Word2) ->  
.
```

Racket Solution:

```
(define/contract (is-it-possible word1 word2)  
  (-> string? string? boolean?)  
)
```