Problem 2330: Valid Palindrome IV

Problem Information

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

Problem Description

You are given a

0-indexed

string

s

consisting of only lowercase English letters. In one operation, you can change

any

character of

s

to any

other

character.

Return

true

if you can make

s

a palindrome after performing

exactly

one or two operations, or return

false

otherwise.

Example 1:

Input:

s = "abcdba"

Output:

true

Explanation:

One way to make s a palindrome using 1 operation is: - Change s[2] to 'd'. Now, s = "abddba". One operation could be performed to make s a palindrome so return true.

Example 2:

Input:

s = "aa"

Output:

true

Explanation:

One way to make s a palindrome using 2 operations is: - Change s[0] to 'b'. Now, s = "ba". - Change s[1] to 'b'. Now, s = "bb". Two operations could be performed to make s a palindrome so return true.

Example 3:

Input:

s = "abcdef"

Output:

false

Explanation:

It is not possible to make s a palindrome using one or two operations so return false.

Constraints:

1 <= s.length <= 10

5

s

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool makePalindrome(string s) {

}
};
```

Java:

```
class Solution {
public boolean makePalindrome(String s) {

}
}
```

Python3:

```
class Solution:
def makePalindrome(self, s: str) -> bool:
```

Python:

```
class Solution(object):
def makePalindrome(self, s):
"""
:type s: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {boolean}
*/
var makePalindrome = function(s) {

};
```

TypeScript:

```
function makePalindrome(s: string): boolean {

};
```

C#:

```
}
```

```
public class Solution {
public bool MakePalindrome(string s) {

}
}
```

C:

```
bool makePalindrome(char* s) {

}
```

Go:

```
func makePalindrome(s string) bool {

}
```

Kotlin:

```
class Solution {
fun makePalindrome(s: String): Boolean {

}
}
```

Swift:

```
class Solution {
func makePalindrome(_ s: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn make_palindrome(s: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @return {Boolean}
def make_palindrome(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function makePalindrome($s) {

}
}
```

Dart:

```
class Solution {
bool makePalindrome(String s) {

}
}
```

Scala:

```
object Solution {
def makePalindrome(s: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec make_palindrome(s :: String.t) :: boolean
def make_palindrome(s) do

end
end
```

Erlang:

```
-spec make_palindrome(S :: unicode:unicode_binary()) -> boolean().
make_palindrome(S) ->
.
```

Racket:

```
(define/contract (make-palindrome s)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Palindrome IV
* Difficulty: Medium
* 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 makePalindrome(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Palindrome IV
* Difficulty: Medium
* 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 makePalindrome(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Palindrome IV
Difficulty: Medium
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 makePalindrome(self, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def makePalindrome(self, s):
"""
:type s: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Valid Palindrome IV
* Difficulty: Medium
* 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} s
* @return {boolean}
*/
var makePalindrome = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Palindrome IV
* Difficulty: Medium
* 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 makePalindrome(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Palindrome IV
* Difficulty: Medium
* 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 MakePalindrome(string s) {

}
}
```

C Solution:

```
/*
* Problem: Valid Palindrome IV
* Difficulty: Medium
* 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 makePalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Valid Palindrome IV
// Difficulty: Medium
// 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 makePalindrome(s string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun makePalindrome(s: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func makePalindrome(_ s: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Valid Palindrome IV
// Difficulty: Medium
// 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 make_palindrome(s: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Boolean}
def make_palindrome(s)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $s
* @return Boolean
*/
function makePalindrome($s) {

}
}
```

Dart Solution:

```
class Solution {
bool makePalindrome(String s) {

}
}
```

Scala Solution:

```
object Solution {
def makePalindrome(s: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec make_palindrome(s :: String.t) :: boolean
def make_palindrome(s) do

end
end
```

Erlang Solution:

```
-spec make_palindrome(S :: unicode:unicode_binary()) -> boolean().
make_palindrome(S) ->
.
```

Racket Solution:

```
(define/contract (make-palindrome s)
(-> string? boolean?)
)
```