Problem 1616: Split Two Strings to Make Palindrome

Problem Information

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

Problem Description

You are given two strings

a

and

b

of the same length. Choose an index and split both strings

at the same index

, splitting

a

into two strings:

a

prefix

and

a

suffix

where

a = a

prefix

+ a

suffix

, and splitting

b

into two strings:

b

prefix

and

b

suffix

where

b = b

prefix

+ b

suffix

. Check if

a

prefix

+ b

suffix

or

b

prefix

+ a

suffix

forms a palindrome.

When you split a string

s

into

s

prefix

and

s

suffix

, either

s

suffix

or

s

prefix

is allowed to be empty. For example, if

s = "abc"

, then

"" + "abc"

,

"a" + "bc"

,

"ab" + "c"

, and

"abc" + ""

are valid splits.

Return

true

if it is possible to form

a palindrome string, otherwise return

false

.

Notice

that

x + y

denotes the concatenation of strings

x

and

y

.

Example 1:

Input:

a = "x", b = "y"

Output:

true

Explaination:

If either a or b are palindromes the answer is true since you can split in the following way: a

prefix

= "", a

suffix

= "x" b

prefix

= "", b

suffix

= "y" Then, a

prefix

+ b

suffix

= "" + "y" = "y", which is a palindrome.

Example 2:

Input:

a = "xbdef", b = "xecab"

Output:

false

Example 3:

Input:

a = "ulacfd", b = "jizalu"

Output:

true

Explaination:

Split them at index 3: a

prefix

= "ula", a

suffix

= "cfd" b

prefix

= "jiz", b

suffix

= "alu" Then, a

prefix

+ b

suffix

= "ula" + "alu" = "ulaalu", which is a palindrome.

Constraints:

1 <= a.length, b.length <= 10

5

a.length == b.length

a

and

b

consist of lowercase English letters

Code Snippets

C++:

```
class Solution {
public:
bool checkPalindromeFormation(string a, string b) {

}
};
```

Java:

```
class Solution {
public boolean checkPalindromeFormation(String a, String b) {

}
}
```

Python3:

```
class Solution:
def checkPalindromeFormation(self, a: str, b: str) -> bool:
```

Python:

```
class Solution(object):
def checkPalindromeFormation(self, a, b):
"""
:type a: str
:type b: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} a
* @param {string} b
* @return {boolean}
*/
var checkPalindromeFormation = function(a, b) {

};
```

TypeScript:

```
function checkPalindromeFormation(a: string, b: string): boolean {

};
```

C#:

```
public class Solution {
public bool CheckPalindromeFormation(string a, string b) {

}
}
```

C:

```
bool checkPalindromeFormation(char* a, char* b) {

}
```

Go:

```
func checkPalindromeFormation(a string, b string) bool {

}
```

Kotlin:

```
class Solution {
fun checkPalindromeFormation(a: String, b: String): Boolean {

}
```

```
}
```

Swift:

```
class Solution {
func checkPalindromeFormation(_ a: String, _ b: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn check_palindrome_formation(a: String, b: String) -> bool {

}
}
```

Ruby:

```
# @param {String} a
# @param {String} b
# @return {Boolean}
def check_palindrome_formation(a, b)

end
```

PHP:

```
class Solution {

/**
* @param String $a
* @param String $b
* @return Boolean
*/
function checkPalindromeFormation($a, $b) {

}
}
```

Dart:

```
class Solution {
bool checkPalindromeFormation(String a, String b) {

}
}
```

Scala:

```
object Solution {
def checkPalindromeFormation(a: String, b: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_palindrome_formation(a :: String.t, b :: String.t) :: boolean
def check_palindrome_formation(a, b) do

end
end
```

Erlang:

```
-spec check_palindrome_formation(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary()) -> boolean().
check_palindrome_formation(A, B) ->
.
```

Racket:

```
(define/contract (check-palindrome-formation a b)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split Two Strings to Make Palindrome
* 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 checkPalindromeFormation(string a, string b) {

}
};
```

Java Solution:

```
/**
* Problem: Split Two Strings to Make Palindrome
* 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 checkPalindromeFormation(String a, String b) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Two Strings to Make Palindrome
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 checkPalindromeFormation(self, a: str, b: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def checkPalindromeFormation(self, a, b):
"""
:type a: str
:type b: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Split Two Strings to Make Palindrome
* 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} a
* @param {string} b
* @return {boolean}
*/
var checkPalindromeFormation = function(a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Split Two Strings to Make Palindrome
* 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 checkPalindromeFormation(a: string, b: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Split Two Strings to Make Palindrome
* 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 CheckPalindromeFormation(string a, string b) {

}
}
```

C Solution:

```
/*
* Problem: Split Two Strings to Make Palindrome
* 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 checkPalindromeFormation(char* a, char* b) {

}
```

Go Solution:

```
// Problem: Split Two Strings to Make Palindrome
// 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 checkPalindromeFormation(a string, b string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkPalindromeFormation(a: String, b: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func checkPalindromeFormation(_ a: String, _ b: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Split Two Strings to Make Palindrome
// 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 check_palindrome_formation(a: String, b: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} a
# @param {String} b
# @return {Boolean}
def check_palindrome_formation(a, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $a
* @param String $b
* @return Boolean
*/
function checkPalindromeFormation($a, $b) {

}
}
```

Dart Solution:

```
class Solution {
bool checkPalindromeFormation(String a, String b) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def checkPalindromeFormation(a: String, b: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_palindrome_formation(a :: String.t, b :: String.t) :: boolean
def check_palindrome_formation(a, b) do

end
end
```

Erlang Solution:

```
-spec check_palindrome_formation(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary()) -> boolean().
check_palindrome_formation(A, B) ->
.
```

Racket Solution:

```
(define/contract (check-palindrome-formation a b)
(-> string? string? boolean?)
)
```