Problem 680: Valid Palindrome II

Problem Information

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

Problem Description

Given a string

s

, return

true

if the

s

can be palindrome after deleting

at most one

character from it

.

Example 1:

Input:

s = "aba"

Output:

true

Example 2:

Input:

s = "abca"

Output:

true

Explanation:

You could delete the character 'c'.

Example 3:

Input:

s = "abc"

Output:

false

Constraints:

1 <= s.length <= 10

5

s

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

}

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
bool validPalindrome(char* s) {

}
```

Go:

```
func validPalindrome(s string) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Valid Palindrome II
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 validPalindrome(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Palindrome II
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 validPalindrome(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Palindrome II
Difficulty: Easy
Tags: array, string, greedy

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 validPalindrome(self, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Valid Palindrome II
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 validPalindrome = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Palindrome II
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 validPalindrome(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Palindrome II
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 ValidPalindrome(string s) {

}
}
```

C Solution:

```
/*
* Problem: Valid Palindrome II
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 validPalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Valid Palindrome II
// Difficulty: Easy
// Tags: array, string, greedy
//
// 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 validPalindrome(s string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Valid Palindrome II
// Difficulty: Easy
// Tags: array, string, greedy
//
// 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 valid_palindrome(s: String) -> bool {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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