Problem 1745: Palindrome Partitioning IV

Problem Information

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

Problem Description

Given a string

s

, return

true

if it is possible to split the string

s

into three

non-empty

palindromic substrings. Otherwise, return

false

.

A string is said to be palindrome if it the same string when reversed.

Example 1:

Input:

s = "abcbdd"

Output:

true

Explanation:

"abcbdd" = "a" + "bcb" + "dd", and all three substrings are palindromes.

Example 2:

Input:

s = "bcbddxy"

Output:

false

Explanation:

s cannot be split into 3 palindromes.

Constraints:

3 <= s.length <= 2000

s

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function checkPartitioning(s: string): boolean {
};
```

C#:

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

}
}
```

C:

```
bool checkPartitioning(char* s) {

}
```

Go:

```
func checkPartitioning(s string) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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



}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-partitioning s)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Palindrome Partitioning IV
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool checkPartitioning(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Palindrome Partitioning IV
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean checkPartitioning(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Palindrome Partitioning IV
Difficulty: Hard
Tags: string, tree, dp

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

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Palindrome Partitioning IV
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Palindrome Partitioning IV
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function checkPartitioning(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Palindrome Partitioning IV
* Difficulty: Hard
* Tags: string, tree, dp
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool CheckPartitioning(string s) {

}
}
```

C Solution:

```
/*
* Problem: Palindrome Partitioning IV
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool checkPartitioning(char* s) {

}
```

Go Solution:

```
// Problem: Palindrome Partitioning IV
// Difficulty: Hard
// Tags: string, tree, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func checkPartitioning(s string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Palindrome Partitioning IV
// Difficulty: Hard
// Tags: string, tree, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-partitioning s)
(-> string? boolean?)
)
```