Problem 1933: Check if String Is Decomposable Into Value-Equal Substrings

Problem Information

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

Problem Description

A

value-equal

string is a string where

all

characters are the same.

For example,

"1111"

and

"33"

are value-equal strings.

In contrast,

"123"

is not a value-equal string.

Given a digit string

s

, decompose the string into some number of

consecutive value-equal

substrings where

exactly one

substring has a

length of

2

and the remaining substrings have a

length of

3

.

Return

true

if you can decompose

s

according to the above rules. Otherwise, return

false

.

A

substring

is a contiguous sequence of characters in a string.

Example 1:

Input:

s = "000111000"

Output:

false

Explanation:

s cannot be decomposed according to the rules because ["000", "111", "000"] does not have a substring of length 2.

Example 2:

Input:

s = "00011111222"

Output:

true

Explanation:

s can be decomposed into ["000", "111", "11", "222"].

Example 3:

Input:

s = "011100022233"

Output:

false

Explanation:

s cannot be decomposed according to the rules because of the first '0'.

Constraints:

1 <= s.length <= 1000

s

consists of only digits

'0'

through

'9'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
bool isDecomposable(char* s) {

}
```

Go:

```
func isDecomposable(s string) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-decomposable s)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if String Is Decomposable Into Value-Equal Substrings
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Check if String Is Decomposable Into Value-Equal Substrings
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public boolean isDecomposable(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if String Is Decomposable Into Value-Equal Substrings
Difficulty: Easy
Tags: string, tree

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if String Is Decomposable Into Value-Equal Substrings
* Difficulty: Easy
```

```
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Check if String Is Decomposable Into Value-Equal Substrings
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function isDecomposable(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if String Is Decomposable Into Value-Equal Substrings
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

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

}
}
```

C Solution:

```
/*
* Problem: Check if String Is Decomposable Into Value-Equal Substrings
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

bool isDecomposable(char* s) {

}
```

Go Solution:

```
// Problem: Check if String Is Decomposable Into Value-Equal Substrings
// Difficulty: Easy
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func isDecomposable(s string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check if String Is Decomposable Into Value-Equal Substrings
// Difficulty: Easy
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-decomposable s)
(-> string? boolean?)
)
```