Problem 1849: Splitting a String Into Descending Consecutive Values

Problem Information

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

Problem Description

You are given a string

s

that consists of only digits.

Check if we can split

s

into

two or more non-empty substrings

such that the

numerical values

of the substrings are in

descending order

and the

difference

between numerical values of every two

adjacent

substrings

is equal to

1

.

For example, the string

s = "0090089"

can be split into

["0090", "089"]

with numerical values

[90,89]

. The values are in descending order and adjacent values differ by

1

, so this way is valid.

Another example, the string

s = "001"

can be split into

["0", "01"]

,

["00", "1"]

, or

["0", "0", "1"]

. However all the ways are invalid because they have numerical values

[0,1]

,

[0,1]

, and

[0,0,1]

respectively, all of which are not in descending order.

Return

true

if it is possible to split

s

as described above

, or

false

otherwise.

A

substring

is a contiguous sequence of characters in a string.

Example 1:

Input:

s = "1234"

Output:

false

Explanation:

There is no valid way to split s.

Example 2:

Input:

s = "050043"

Output:

true

Explanation:

s can be split into ["05", "004", "3"] with numerical values [5,4,3]. The values are in descending order with adjacent values differing by 1.

Example 3:

Input:

s = "9080701"

Output:

false

Explanation:

There is no valid way to split s.

Constraints:

1 <= s.length <= 20

s

only consists of digits.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
bool splitString(char* s) {

}
```

Go:

```
func splitString(s string) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Boolean
```

```
*/
function splitString($s) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Splitting a String Into Descending Consecutive Values
* Difficulty: Medium
* 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 splitString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Splitting a String Into Descending Consecutive Values
* Difficulty: Medium
* 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 splitString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Splitting a String Into Descending Consecutive Values
Difficulty: Medium
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 splitString(self, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Splitting a String Into Descending Consecutive Values
* Difficulty: Medium
* 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 splitString = function(s) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Splitting a String Into Descending Consecutive Values
* Difficulty: Medium
* 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 splitString(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Splitting a String Into Descending Consecutive Values
* Difficulty: Medium
* 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 SplitString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Splitting a String Into Descending Consecutive Values
* Difficulty: Medium
```

```
* 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 splitString(char* s) {

}
```

Go Solution:

```
// Problem: Splitting a String Into Descending Consecutive Values
// Difficulty: Medium
// 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 splitString(s string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Splitting a String Into Descending Consecutive Values
// Difficulty: Medium
// 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 split_string(s: String) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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