Problem 2825: Make String a Subsequence Using Cyclic Increments

Problem Information

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

Problem Description

You are given two

0-indexed

strings

str1

and

str2

.

In an operation, you select a

set

of indices in

str1

, and for each index

i

in the set, increment

str1[i]

to the next character

cyclically

. That is

'a'

becomes

'b'

,

'b'

becomes

'c'

, and so on, and

'z'

becomes

'a'

.

Return

true

if it is possible to make

str2

a subsequence of

str1

by performing the operation

at most once

,

and

false

otherwise

.

Note:

A subsequence of a string is a new string that is formed from the original string by deleting some (possibly none) of the characters without disturbing the relative positions of the remaining characters.

Example 1:

Input:

str1 = "abc", str2 = "ad"

Output:

true

Explanation:

Select index 2 in str1. Increment str1[2] to become 'd'. Hence, str1 becomes "abd" and str2 is now a subsequence. Therefore, true is returned.

Example 2:

Input:

str1 = "zc", str2 = "ad"

Output:

true

Explanation:

Select indices 0 and 1 in str1. Increment str1[0] to become 'a'. Increment str1[1] to become 'd'. Hence, str1 becomes "ad" and str2 is now a subsequence. Therefore, true is returned.

Example 3:

Input:

str1 = "ab", str2 = "d"

Output:

false

Explanation:

In this example, it can be shown that it is impossible to make str2 a subsequence of str1 using the operation at most once. Therefore, false is returned.

Constraints:

1 <= str1.length <= 10

5

1 <= str2.length <= 10

5

str1

and

str2

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool canMakeSubsequence(string str1, string str2) {

}
};
```

Java:

```
class Solution {
public boolean canMakeSubsequence(String str1, String str2) {

}
}
```

Python3:

```
class Solution:
def canMakeSubsequence(self, str1: str, str2: str) -> bool:
```

Python:

```
class Solution(object):
def canMakeSubsequence(self, str1, str2):
```

```
"""
:type str1: str
:type str2: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} str1
* @param {string} str2
* @return {boolean}
*/
var canMakeSubsequence = function(str1, str2) {

};
```

TypeScript:

```
function canMakeSubsequence(str1: string, str2: string): boolean {

};
```

C#:

```
public class Solution {
public bool CanMakeSubsequence(string str1, string str2) {

}
}
```

C:

```
bool canMakeSubsequence(char* str1, char* str2) {

}
```

Go:

```
func canMakeSubsequence(str1 string, str2 string) bool {

}
```

Kotlin:

```
class Solution {
fun canMakeSubsequence(str1: String, str2: String): Boolean {

}
}
```

Swift:

```
class Solution {
func canMakeSubsequence(_ str1: String, _ str2: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_make_subsequence(str1: String, str2: String) -> bool {

}
}
```

Ruby:

```
# @param {String} str1
# @param {String} str2
# @return {Boolean}
def can_make_subsequence(str1, str2)

end
```

PHP:

```
class Solution {

/**
* @param String $str1
* @param String $str2
* @return Boolean
*/
function canMakeSubsequence($str1, $str2) {
}
}
```

Dart:

```
class Solution {
bool canMakeSubsequence(String str1, String str2) {

}
}
```

Scala:

```
object Solution {
def canMakeSubsequence(str1: String, str2: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_make_subsequence(str1 :: String.t, str2 :: String.t) :: boolean
def can_make_subsequence(str1, str2) do

end
end
```

Erlang:

```
-spec can_make_subsequence(Str1 :: unicode:unicode_binary(), Str2 ::
unicode:unicode_binary()) -> boolean().
can_make_subsequence(Str1, Str2) ->
.
```

Racket:

```
(define/contract (can-make-subsequence str1 str2)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make String a Subsequence Using Cyclic Increments
* 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 canMakeSubsequence(string str1, string str2) {

}
};
```

Java Solution:

```
/**
* Problem: Make String a Subsequence Using Cyclic Increments
* 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 canMakeSubsequence(String str1, String str2) {

}
}
```

Python3 Solution:

```
"""
Problem: Make String a Subsequence Using Cyclic Increments
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 canMakeSubsequence(self, str1: str, str2: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canMakeSubsequence(self, str1, str2):
"""
:type str1: str
:type str2: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Make String a Subsequence Using Cyclic Increments
* 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} str1
* @param {string} str2
* @return {boolean}
*/
```

```
var canMakeSubsequence = function(str1, str2) {

};
```

TypeScript Solution:

```
/**
* Problem: Make String a Subsequence Using Cyclic Increments
* 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 canMakeSubsequence(str1: string, str2: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Make String a Subsequence Using Cyclic Increments
* 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 CanMakeSubsequence(string str1, string str2) {

}
}
```

C Solution:

```
/*
* Problem: Make String a Subsequence Using Cyclic Increments
* 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 canMakeSubsequence(char* str1, char* str2) {

}
```

Go Solution:

```
// Problem: Make String a Subsequence Using Cyclic Increments
// 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 canMakeSubsequence(str1 string, str2 string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canMakeSubsequence(str1: String, str2: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canMakeSubsequence(_ str1: String, _ str2: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Make String a Subsequence Using Cyclic Increments
// 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 can_make_subsequence(str1: String, str2: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} str1
# @param {String} str2
# @return {Boolean}
def can_make_subsequence(str1, str2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $str1
* @param String $str2
* @return Boolean
*/
function canMakeSubsequence($str1, $str2) {

}
}
```

Dart Solution:

```
class Solution {
bool canMakeSubsequence(String str1, String str2) {

}
}
```

Scala Solution:

```
object Solution {
def canMakeSubsequence(str1: String, str2: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_make_subsequence(str1 :: String.t, str2 :: String.t) :: boolean
def can_make_subsequence(str1, str2) do

end
end
```

Erlang Solution:

```
-spec can_make_subsequence(Str1 :: unicode:unicode_binary(), Str2 ::
unicode:unicode_binary()) -> boolean().
can_make_subsequence(Str1, Str2) ->
.
```

Racket Solution:

```
(define/contract (can-make-subsequence str1 str2)
(-> string? string? boolean?)
)
```