Problem 2301: Match Substring After Replacement

Problem Information

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

Problem Description

You are given two strings

s

and

sub

. You are also given a 2D character array

mappings

where

mappings[i] = [old

i

, new

i

]

indicates that you may perform the following operation any

number of times:

Replace

a character

old

i

of

sub

with

new

i

.

Each character in

sub

cannot

be replaced more than once.

Return

true

if it is possible to make

sub

a substring of

s

by replacing zero or more characters according to

mappings

. Otherwise, return

false

.

A

substring

is a contiguous non-empty sequence of characters within a string.

Example 1:

Input:

s = "fool3e7bar", sub = "leet", mappings = [["e","3"],["t","7"],["t","8"]]

Output:

true

Explanation:

Replace the first 'e' in sub with '3' and 't' in sub with '7'. Now sub = "l3e7" is a substring of s, so we return true.

Example 2:

Input:

s = "fooleetbar", sub = "f00l", mappings = [["o","0"]]

Output:

false

Explanation:

The string "f00l" is not a substring of s and no replacements can be made. Note that we cannot replace '0' with 'o'.

Example 3:

Input:

s = "Fool33tbaR", sub = "leetd", mappings = [["e","3"],["t","7"],["t","8"],["d","b"],["p","b"]]

Output:

true

Explanation:

Replace the first and second 'e' in sub with '3' and 'd' in sub with 'b'. Now sub = "l33tb" is a substring of s, so we return true.

Constraints:

1 <= sub.length <= s.length <= 5000

0 <= mappings.length <= 1000

mappings[i].length == 2

old

i

!= new

i

s

and

sub

consist of uppercase and lowercase English letters and digits.

old

i

and

new

i

are either uppercase or lowercase English letters or digits.

Code Snippets

C++:

```
class Solution {
public:
bool matchReplacement(string s, string sub, vector<vector<char>>& mappings) {

}
};
```

Java:

```
class Solution {
public boolean matchReplacement(String s, String sub, char[][] mappings) {
}
}
```

Python3:

```
class Solution:
def matchReplacement(self, s: str, sub: str, mappings: List[List[str]]) ->
bool:
```

Python:

```
class Solution(object):
def matchReplacement(self, s, sub, mappings):
"""
:type s: str
:type sub: str
:type mappings: List[List[str]]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} sub
* @param {character[][]} mappings
* @return {boolean}
*/
var matchReplacement = function(s, sub, mappings) {

};
```

TypeScript:

```
function matchReplacement(s: string, sub: string, mappings: string[][]):
boolean {

};
```

C#:

```
public class Solution {
public bool MatchReplacement(string s, string sub, char[][] mappings) {

}
}
```

C:

```
bool matchReplacement(char* s, char* sub, char** mappings, int mappingsSize,
int* mappingsColSize) {

}
```

Go:

```
func matchReplacement(s string, sub string, mappings [][]byte) bool {

}
```

Kotlin:

```
class Solution {
fun matchReplacement(s: String, sub: String, mappings: Array<CharArray>):
Boolean {

}
}
```

Swift:

```
class Solution {
func matchReplacement(_ s: String, _ sub: String, _ mappings: [[Character]])
-> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn match_replacement(s: String, sub: String, mappings: Vec<Vec<char>>) ->
bool {
}
}
```

Ruby:

```
# @param {String} s
# @param {String} sub
# @param {Character[][]} mappings
# @return {Boolean}
def match_replacement(s, sub, mappings)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $sub
* @param String[][] $mappings
* @return Boolean
*/
function matchReplacement($s, $sub, $mappings) {

}
}
```

Dart:

```
class Solution {
bool matchReplacement(String s, String sub, List<List<String>> mappings) {

}
}
```

Scala:

```
object Solution {
def matchReplacement(s: String, sub: String, mappings: Array[Array[Char]]):
Boolean = {
}
}
```

Elixir:

```
defmodule Solution do
@spec match_replacement(s :: String.t, sub :: String.t, mappings :: [[char]])
:: boolean
def match_replacement(s, sub, mappings) do

end
end
```

Erlang:

```
-spec match_replacement(S :: unicode:unicode_binary(), Sub ::
unicode:unicode_binary(), Mappings :: [[char()]]) -> boolean().
match_replacement(S, Sub, Mappings) ->
.
```

Racket:

```
(define/contract (match-replacement s sub mappings)
(-> string? string? (listof (listof char?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Match Substring After Replacement
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class Solution {
public:
bool matchReplacement(string s, string sub, vector<vector<char>>& mappings) {

}
};
```

Java Solution:

```
/**
* Problem: Match Substring After Replacement
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public boolean matchReplacement(String s, String sub, char[][] mappings) {

}
}
```

Python3 Solution:

```
"""
Problem: Match Substring After Replacement
Difficulty: Hard
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def matchReplacement(self, s: str, sub: str, mappings: List[List[str]]) ->
bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def matchReplacement(self, s, sub, mappings):
"""
:type s: str
:type sub: str
:type mappings: List[List[str]]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Match Substring After Replacement
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} s
* @param {string} sub
* @param {character[][]} mappings
* @return {boolean}
*/
var matchReplacement = function(s, sub, mappings) {

};
```

TypeScript Solution:

```
/**
* Problem: Match Substring After Replacement
* Difficulty: Hard
* Tags: array, string, tree, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function matchReplacement(s: string, sub: string, mappings: string[][]):
boolean {

};
```

C# Solution:

```
/*
* Problem: Match Substring After Replacement
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* 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 MatchReplacement(string s, string sub, char[][] mappings) {

}
}
```

C Solution:

```
/*
* Problem: Match Substring After Replacement
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
bool matchReplacement(char* s, char* sub, char** mappings, int mappingsSize,
int* mappingsColSize) {

}
```

Go Solution:

```
// Problem: Match Substring After Replacement
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func matchReplacement(s string, sub string, mappings [][]byte) bool {

}
```

Kotlin Solution:

```
class Solution {
fun matchReplacement(s: String, sub: String, mappings: Array<CharArray>):
Boolean {

}
}
```

Swift Solution:

```
class Solution {
func matchReplacement(_ s: String, _ sub: String, _ mappings: [[Character]])
-> Bool {

}
}
```

Rust Solution:

```
// Problem: Match Substring After Replacement
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn match_replacement(s: String, sub: String, mappings: Vec<Vec<char>>) ->
bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} sub
# @param {Character[][]} mappings
# @return {Boolean}
def match_replacement(s, sub, mappings)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $sub
* @param String[][] $mappings
* @return Boolean
*/
function matchReplacement($s, $sub, $mappings) {

}
}
```

Dart Solution:

```
class Solution {
bool matchReplacement(String s, String sub, List<List<String>> mappings) {

}
}
```

Scala Solution:

```
object Solution {
def matchReplacement(s: String, sub: String, mappings: Array[Array[Char]]):
Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec match_replacement(s :: String.t, sub :: String.t, mappings :: [[char]])
:: boolean
def match_replacement(s, sub, mappings) do

end
end
```

Erlang Solution:

```
-spec match_replacement(S :: unicode:unicode_binary(), Sub ::
unicode:unicode_binary(), Mappings :: [[char()]]) -> boolean().
match_replacement(S, Sub, Mappings) ->
.
```

Racket Solution:

```
(define/contract (match-replacement s sub mappings)
(-> string? string? (listof (listof char?)) boolean?)
)
```