Problem 392: Is Subsequence

Problem Information

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

Problem Description

Given two strings

s

and

t

, return

true

if

s

is a

subsequence

of

t

, or

false

otherwise

.

A

subsequence

of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e.,

"ace"

is a subsequence of

"

a

b

c

d

e

"

while

"aec"

is not).

Example 1:

Input:

s = "abc", t = "ahbgdc"

Output:

true

Example 2:

Input:

s = "axc", t = "ahbgdc"

Output:

false

Constraints:

0 <= s.length <= 100

0 <= t.length <= 10

4

s

and

t

consist only of lowercase English letters.

Follow up:

Suppose there are lots of incoming

s

, say

s

1

, s

2

, ..., s

k

where

k >= 10

9

, and you want to check one by one to see if

t

has its subsequence. In this scenario, how would you change your code?

Code Snippets

C++:

```
class Solution {
public:
bool isSubsequence(string s, string t) {

}
};
```

Java:

```
class Solution {
public boolean isSubsequence(String s, String t) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function isSubsequence(s: string, t: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsSubsequence(string s, string t) {

}
}
```

C:

```
bool isSubsequence(char* s, char* t) {

}
```

Go:

```
func isSubsequence(s string, t string) bool {

}
```

Kotlin:

```
class Solution {
fun isSubsequence(s: String, t: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isSubsequence(_ s: String, _ t: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_subsequence(s: String, t: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @return {Boolean}
def is_subsequence(s, t)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Boolean
*/
function isSubsequence($s, $t) {

}
}
```

Dart:

```
class Solution {
bool isSubsequence(String s, String t) {

}
}
```

Scala:

```
object Solution {
def isSubsequence(s: String, t: String): Boolean = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

```
-spec is_subsequence(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> boolean().
is_subsequence(S, T) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Is Subsequence
* Difficulty: Easy
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool isSubsequence(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Is Subsequence
* Difficulty: Easy
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean isSubsequence(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Is Subsequence
Difficulty: Easy
Tags: array, string, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Is Subsequence
* Difficulty: Easy
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @param {string} t
* @return {boolean}
*/
var isSubsequence = function(s, t) {

};
```

TypeScript Solution:

```
/**
* Problem: Is Subsequence
* Difficulty: Easy
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function isSubsequence(s: string, t: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Is Subsequence
* Difficulty: Easy
```

```
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* 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 IsSubsequence(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Is Subsequence
* Difficulty: Easy
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool isSubsequence(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Is Subsequence
// Difficulty: Easy
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func isSubsequence(s string, t string) bool {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun isSubsequence(s: String, t: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isSubsequence(_ s: String, _ t: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Is Subsequence
// Difficulty: Easy
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn is_subsequence(s: String, t: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @return {Boolean}
def is_subsequence(s, t)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Boolean
*/
function isSubsequence($s, $t) {

}
}
```

Dart Solution:

```
class Solution {
bool isSubsequence(String s, String t) {

}
}
```

Scala Solution:

```
object Solution {
def isSubsequence(s: String, t: String): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec is_subsequence(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> boolean().
is_subsequence(S, T) ->
.
```

Racket Solution:

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