Problem 1143: Longest Common Subsequence

Problem Information

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

Problem Description

Given two strings

text1

and

text2

, return

the length of their longest

common subsequence

.

If there is no

common subsequence

, return

0

.

A

subsequence

of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

For example,

"ace"

is a subsequence of

"abcde"

.

A

common subsequence

of two strings is a subsequence that is common to both strings.

Example 1:

Input:

text1 = "abcde", text2 = "ace"

Output:

3

Explanation:

The longest common subsequence is "ace" and its length is 3.

Example 2:

Input:

text1 = "abc", text2 = "abc"

Output:

3

Explanation:

The longest common subsequence is "abc" and its length is 3.

Example 3:

Input:

text1 = "abc", text2 = "def"

Output:

0

Explanation:

There is no such common subsequence, so the result is 0.

Constraints:

1 <= text1.length, text2.length <= 1000

text1

and

text2

consist of only lowercase English characters.

Code Snippets

C++:

```
class Solution {
public:
int longestCommonSubsequence(string text1, string text2) {

}
};
```

Java:

```
class Solution {
public int longestCommonSubsequence(String text1, String text2) {

}
}
```

Python3:

```
class Solution:
def longestCommonSubsequence(self, text1: str, text2: str) -> int:
```

Python:

```
class Solution(object):
def longestCommonSubsequence(self, text1, text2):
"""
:type text1: str
:type text2: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} text1
* @param {string} text2
* @return {number}
*/
var longestCommonSubsequence = function(text1, text2) {
```

```
};
```

TypeScript:

```
function longestCommonSubsequence(text1: string, text2: string): number {

};
```

C#:

```
public class Solution {
public int LongestCommonSubsequence(string text1, string text2) {

}
}
```

C:

```
int longestCommonSubsequence(char* text1, char* text2) {

}
```

Go:

```
func longestCommonSubsequence(text1 string, text2 string) int {

}
```

Kotlin:

```
class Solution {
fun longestCommonSubsequence(text1: String, text2: String): Int {

}
}
```

Swift:

```
class Solution {
func longestCommonSubsequence(_ text1: String, _ text2: String) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn longest_common_subsequence(text1: String, text2: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} text1
# @param {String} text2
# @return {Integer}
def longest_common_subsequence(text1, text2)

end
```

PHP:

```
class Solution {

/**
* @param String $text1
* @param String $text2
* @return Integer
*/
function longestCommonSubsequence($text1, $text2) {

}
}
```

Dart:

```
class Solution {
int longestCommonSubsequence(String text1, String text2) {

}
}
```

Scala:

```
object Solution {
def longestCommonSubsequence(text1: String, text2: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_common_subsequence(text1 :: String.t, text2 :: String.t) ::
integer
def longest_common_subsequence(text1, text2) do

end
end
```

Erlang:

```
-spec longest_common_subsequence(Text1 :: unicode:unicode_binary(), Text2 ::
unicode:unicode_binary()) -> integer().
longest_common_subsequence(Text1, Text2) ->
.
```

Racket:

```
(define/contract (longest-common-subsequence text1 text2)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Common Subsequence
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public:
int longestCommonSubsequence(string text1, string text2) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Common Subsequence
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int longestCommonSubsequence(String text1, String text2) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Common Subsequence
Difficulty: Medium
Tags: string, dp

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def longestCommonSubsequence(self, text1: str, text2: str) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestCommonSubsequence(self, text1, text2):
"""
:type text1: str
:type text2: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Common Subsequence
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} text1
* @param {string} text2
* @return {number}
*/
var longestCommonSubsequence = function(text1, text2) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Common Subsequence
* Difficulty: Medium
* Tags: string, dp
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function longestCommonSubsequence(text1: string, text2: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Common Subsequence
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int LongestCommonSubsequence(string text1, string text2) {

}
}
```

C Solution:

```
/*
* Problem: Longest Common Subsequence
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int longestCommonSubsequence(char* text1, char* text2) {
```

```
}
```

Go Solution:

```
// Problem: Longest Common Subsequence
// Difficulty: Medium
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func longestCommonSubsequence(text1 string, text2 string) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestCommonSubsequence(text1: String, text2: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestCommonSubsequence(_ text1: String, _ text2: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Common Subsequence
// Difficulty: Medium
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
impl Solution {
pub fn longest_common_subsequence(text1: String, text2: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} text1
# @param {String} text2
# @return {Integer}
def longest_common_subsequence(text1, text2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $text1
* @param String $text2
* @return Integer
*/
function longestCommonSubsequence($text1, $text2) {

}
}
```

Dart Solution:

```
class Solution {
int longestCommonSubsequence(String text1, String text2) {

}
}
```

Scala Solution:

```
object Solution {
def longestCommonSubsequence(text1: String, text2: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_common_subsequence(text1 :: String.t, text2 :: String.t) ::
integer
def longest_common_subsequence(text1, text2) do

end
end
```

Erlang Solution:

```
-spec longest_common_subsequence(Text1 :: unicode:unicode_binary(), Text2 ::
unicode:unicode_binary()) -> integer().
longest_common_subsequence(Text1, Text2) ->
.
```

Racket Solution:

```
(define/contract (longest-common-subsequence text1 text2)
(-> string? string? exact-integer?)
)
```