Problem 3302: Find the Lexicographically Smallest Valid Sequence

Problem Information

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

Problem Description

You are given two strings

word1

and

word2

.

A string

x

is called

almost equal

to

y

if you can change

at most

one character in

x

to make it

identical

to

y

.

A sequence of indices

seq

is called

valid

if:

The indices are sorted in

ascending

order.

Concatenating

the characters at these indices in

word1

in

the same

order results in a string that is

almost equal

to

word2

.

Return an array of size

word2.length

representing the

lexicographically smallest

valid

sequence of indices. If no such sequence of indices exists, return an

empty

array.

Note

that the answer must represent the

lexicographically smallest array

,

not

the corresponding string formed by those indices.

Example 1:

Input:

word1 = "vbcca", word2 = "abc"

Output:

[0,1,2]

Explanation:

The lexicographically smallest valid sequence of indices is

[0, 1, 2]

:

Change

word1[0]

to

'a'

.

word1[1]

is already

'b'

.

word1[2]

is already

'c'

.

Example 2:

Input:

word1 = "bacdc", word2 = "abc"

Output:

[1,2,4]

Explanation:

The lexicographically smallest valid sequence of indices is

[1, 2, 4]

:

word1[1]

is already

'a'

.

Change

word1[2]

to

'b'

.

word1[4]

is already

'c'

.

Example 3:

Input:

word1 = "aaaaaa", word2 = "aaabc"

Output:

[]

Explanation:

There is no valid sequence of indices.

Example 4:

Input:

word1 = "abc", word2 = "ab"

Output:

[0,1]

Constraints:

1 <= word2.length < word1.length <= 3 * 10

5

word1

and

word2

consist only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<int> validSequence(string word1, string word2) {

}
};
```

Java:

```
class Solution {
public int[] validSequence(String word1, String word2) {

}
}
```

Python3:

```
class Solution:
def validSequence(self, word1: str, word2: str) -> List[int]:
```

Python:

```
class Solution(object):
def validSequence(self, word1, word2):
"""
:type word1: str
```

```
:type word2: str
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} word1
* @param {string} word2
* @return {number[]}
*/
var validSequence = function(word1, word2) {

};
```

TypeScript:

```
function validSequence(word1: string, word2: string): number[] {

};
```

C#:

```
public class Solution {
public int[] ValidSequence(string word1, string word2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* validSequence(char* word1, char* word2, int* returnSize) {

}
```

Go:

```
func validSequence(word1 string, word2 string) []int {
}
```

Kotlin:

```
class Solution {
fun validSequence(word1: String, word2: String): IntArray {

}
}
```

Swift:

```
class Solution {
func validSequence(_ word1: String, _ word2: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn valid_sequence(word1: String, word2: String) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String} word1
# @param {String} word2
# @return {Integer[]}
def valid_sequence(word1, word2)

end
```

PHP:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Integer[]
*/
function validSequence($word1, $word2) {

}
}
```

Dart:

```
class Solution {
List<int> validSequence(String word1, String word2) {

}
}
```

Scala:

```
object Solution {
def validSequence(word1: String, word2: String): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec valid_sequence(word1 :: String.t, word2 :: String.t) :: [integer]
def valid_sequence(word1, word2) do

end
end
```

Erlang:

```
-spec valid_sequence(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> [integer()].
valid_sequence(Word1, Word2) ->
.
```

Racket:

```
(define/contract (valid-sequence word1 word2)
(-> string? string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Lexicographically Smallest Valid Sequence
* Difficulty: Medium
* Tags: array, string, graph, dp, greedy, sort
*
* 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:
vector<int> validSequence(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Lexicographically Smallest Valid Sequence
* Difficulty: Medium
* Tags: array, string, graph, dp, greedy, sort
*
* 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 int[] validSequence(String word1, String word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Lexicographically Smallest Valid Sequence
Difficulty: Medium
Tags: array, string, graph, dp, greedy, sort

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 validSequence(self, word1: str, word2: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def validSequence(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Lexicographically Smallest Valid Sequence
* Difficulty: Medium
* Tags: array, string, graph, dp, greedy, sort
*
* 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} word1
* @param {string} word2
* @return {number[]}
*/
var validSequence = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Lexicographically Smallest Valid Sequence
* Difficulty: Medium
* Tags: array, string, graph, dp, greedy, sort
*
* 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 validSequence(word1: string, word2: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Lexicographically Smallest Valid Sequence
* Difficulty: Medium
* Tags: array, string, graph, dp, greedy, sort
*
* 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 int[] ValidSequence(string word1, string word2) {

}

}
```

C Solution:

```c
/*
* Problem: Find the Lexicographically Smallest Valid Sequence
* Difficulty: Medium
* Tags: array, string, graph, dp, greedy, sort
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* validSequence(char* word1, char* word2, int* returnSize) {

}
```

Go Solution:

```go
// Problem: Find the Lexicographically Smallest Valid Sequence
// Difficulty: Medium
// Tags: array, string, graph, dp, greedy, sort
//
// 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 validSequence(word1 string, word2 string) []int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun validSequence(word1: String, word2: String): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func validSequence(_ word1: String, _ word2: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find the Lexicographically Smallest Valid Sequence
// Difficulty: Medium
// Tags: array, string, graph, dp, greedy, sort
//
// 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 valid_sequence(word1: String, word2: String) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} word1
# @param {String} word2
# @return {Integer[]}
def valid_sequence(word1, word2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Integer[]
*/
function validSequence($word1, $word2) {

}
}
```

Dart Solution:

```
class Solution {
List<int> validSequence(String word1, String word2) {

}
}
```

Scala Solution:

```
object Solution {
def validSequence(word1: String, word2: String): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec valid_sequence(word1 :: String.t, word2 :: String.t) :: [integer]
def valid_sequence(word1, word2) do

end
end
```

Erlang Solution:

```
-spec valid_sequence(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> [integer()].
valid_sequence(Word1, Word2) ->
.
```

Racket Solution:

```
(define/contract (valid-sequence word1 word2)
(-> string? string? (listof exact-integer?))
)
```