Problem 1754: Largest Merge Of Two Strings

Problem Information

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

Problem Description

You are given two strings

word1

and

word2

. You want to construct a string

merge

in the following way: while either

word1

or

word2

are non-empty, choose

one

of the following options:

If

word1

is non-empty, append the

first

character in

word1

to

merge

and delete it from

word1

.

For example, if

word1 = "abc"

and

merge = "dv"

, then after choosing this operation,

word1 = "bc"

and

merge = "dva"

.

If

word2

is non-empty, append the

first

character in

word2

to

merge

and delete it from

word2

.

For example, if

word2 = "abc"

and

merge = ""

, then after choosing this operation,

word2 = "bc"

and

merge = "a"

.

Return

the lexicographically

largest

merge

you can construct

.

A string

a

is lexicographically larger than a string

b

(of the same length) if in the first position where

a

and

b

differ,

a

has a character strictly larger than the corresponding character in

b

. For example,

"abcd"

is lexicographically larger than

"abcc"

because the first position they differ is at the fourth character, and

d

is greater than

c

.

Example 1:

Input:

word1 = "cabaa", word2 = "bcaaa"

Output:

"cbcabaaaaa"

Explanation:

One way to get the lexicographically largest merge is: - Take from word1: merge = "c", word1 = "abaa", word2 = "bcaaa" - Take from word2: merge = "cb", word1 = "abaa", word2 = "caaa" - Take from word2: merge = "cbc", word1 = "abaa", word2 = "aaa" - Take from word1: merge = "cbca", word1 = "baa", word2 = "aaa" - Take from word1: merge = "cbcab", word1 = "aa", word2 = "aaa" - Append the remaining 5 a's from word1 and word2 at the end of merge.

Example 2:

Input:

word1 = "abcabc", word2 = "abdcaba"

Output:

"abdcabcabcaba"

Constraints:

1 <= word1.length, word2.length <= 3000

word1

and

word2

consist only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string largestMerge(string word1, string word2) {

}
};
```

Java:

```
class Solution {
public String largestMerge(String word1, String word2) {

}
}
```

Python3:

```
class Solution:
def largestMerge(self, word1: str, word2: str) -> str:
```

Python:

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

JavaScript:

```
/**
* @param {string} word1
* @param {string} word2
* @return {string}
*/
var largestMerge = function(word1, word2) {

};
```

TypeScript:

```
function largestMerge(word1: string, word2: string): string {

};
```

C#:

```
public class Solution {
public string LargestMerge(string word1, string word2) {

}
}
```

C:

```
char* largestMerge(char* word1, char* word2) {

}
```

Go:

```
func largestMerge(word1 string, word2 string) string {

}
```

Kotlin:

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

}
}
```

Swift:

```
class Solution {
func largestMerge(_ word1: String, _ word2: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn largest_merge(word1: String, word2: String) -> String {

}
}
```

Ruby:

```
# @param {String} word1
# @param {String} word2
# @return {String}
def largest_merge(word1, word2)

end
```

PHP:

```
class Solution {

/**
* @param String $word1
```

```
* @param String $word2
* @return String
*/
function largestMerge($word1, $word2) {

}
}
```

Dart:

```
class Solution {
String largestMerge(String word1, String word2) {

}
}
```

Scala:

```
object Solution {
def largestMerge(word1: String, word2: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec largest_merge(word1 :: String.t, word2 :: String.t) :: String.t
def largest_merge(word1, word2) do

end
end
```

Erlang:

```
-spec largest_merge(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
largest_merge(Word1, Word2) ->
.
```

Racket:

```
(define/contract (largest-merge word1 word2)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Merge Of Two Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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:
string largestMerge(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Merge Of Two Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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 String largestMerge(String word1, String word2) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Largest Merge Of Two Strings
Difficulty: Medium
Tags: array, string, graph, greedy

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 largestMerge(self, word1: str, word2: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Merge Of Two Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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} word1
* @param {string} word2
* @return {string}
*/
var largestMerge = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Merge Of Two Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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 largestMerge(word1: string, word2: string): string {

};
```

C# Solution:

```
/*
* Problem: Largest Merge Of Two Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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 string LargestMerge(string word1, string word2) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Largest Merge Of Two Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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
*/

char* largestMerge(char* word1, char* word2) {

}
```

Go Solution:

```
// Problem: Largest Merge Of Two Strings
// Difficulty: Medium
// Tags: array, string, graph, greedy
//
// 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 largestMerge(word1 string, word2 string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

```
class Solution {
func largestMerge(_ word1: String, _ word2: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Largest Merge Of Two Strings
// Difficulty: Medium
// Tags: array, string, graph, greedy
//
// 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 largest_merge(word1: String, word2: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} word1
# @param {String} word2
# @return {String}
def largest_merge(word1, word2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return String
*/
function largestMerge($word1, $word2) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String largestMerge(String word1, String word2) {

}
}
```

Scala Solution:

```
object Solution {
def largestMerge(word1: String, word2: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec largest_merge(word1 :: String.t, word2 :: String.t) :: String.t
def largest_merge(word1, word2) do

end
end
```

Erlang Solution:

```
-spec largest_merge(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
largest_merge(Word1, Word2) ->
.
```

Racket Solution:

```
(define/contract (largest-merge word1 word2)
(-> string? string? string?)
)
```