Problem 791: Custom Sort String

Problem Information

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

Problem Description

You are given two strings

order

and

s

. All the characters of

order

are

unique

and were sorted in some custom order previously.

Permute the characters of

s

so that they match the order that

order

was sorted. More specifically, if a character

x

occurs before a character

y

in

order

, then

x

should occur before

y

in the permuted string.

Return

any permutation of

s

that satisfies this property

.

Example 1:

Input:

order = "cba", s = "abcd"

Output:

"cbad"

Explanation:

"a"

,

"b"

,

"c"

appear in order, so the order of

"a"

,

"b"

,

"c"

should be

"c"

,

"b"

, and

"a"

.

Since

"d"

does not appear in

order

, it can be at any position in the returned string.

"dcba"

,

"cdba"

,

"cbda"

are also valid outputs.

Example 2:

Input:

order = "bcafg", s = "abcd"

Output:

"bcad"

Explanation:

The characters

"b"

,

"c"

, and

"a"

from

order

dictate the order for the characters in

s

. The character

"d"

in

s

does not appear in

order

, so its position is flexible.

Following the order of appearance in

order

,

"b"

,

"c"

, and

"a"

from

s

should be arranged as

"b"

,

"c"

,

"a"

.

"d"

can be placed at any position since it's not in order. The output

"bcad"

correctly follows this rule. Other arrangements like

"dbca"

or

"bcda"

would also be valid, as long as

"b"

,

"c"

,

"a"

maintain their order.

Constraints:

1 <= order.length <= 26

1 <= s.length <= 200

order

and

s

consist of lowercase English letters.

All the characters of

order

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
string customSortString(string order, string s) {

}
};
```

Java:

```
class Solution {
public String customSortString(String order, String s) {

}
}
```

Python3:

```
class Solution:
def customSortString(self, order: str, s: str) -> str:
```

Python:

```
class Solution(object):
def customSortString(self, order, s):
"""
:type order: str
:type s: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} order
* @param {string} s
* @return {string}
*/
var customSortString = function(order, s) {
```

```
};
```

TypeScript:

```
function customSortString(order: string, s: string): string {

};
```

C#:

```
public class Solution {
public string CustomSortString(string order, string s) {

}
}
```

C:

```
char* customSortString(char* order, char* s) {

}
```

Go:

```
func customSortString(order string, s string) string {

}
```

Kotlin:

```
class Solution {
fun customSortString(order: String, s: String): String {

}
}
```

Swift:

```
class Solution {
func customSortString(_ order: String, _ s: String) -> String {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn custom_sort_string(order: String, s: String) -> String {

}
}
```

Ruby:

```
# @param {String} order
# @param {String} s
# @return {String}
def custom_sort_string(order, s)

end
```

PHP:

```
class Solution {

/**
* @param String $order
* @param String $s
* @return String
*/
function customSortString($order, $s) {

}
}
```

Dart:

```
class Solution {
String customSortString(String order, String s) {

}
}
```

Scala:

```
object Solution {
def customSortString(order: String, s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec custom_sort_string(order :: String.t, s :: String.t) :: String.t
def custom_sort_string(order, s) do

end
end
```

Erlang:

```
-spec custom_sort_string(Order :: unicode:unicode_binary(), S ::
unicode:unicode_binary()) -> unicode:unicode_binary().
custom_sort_string(Order, S) ->
.
```

Racket:

```
(define/contract (custom-sort-string order s)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Custom Sort String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public:
string customSortString(string order, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Custom Sort String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String customSortString(String order, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Custom Sort String
Difficulty: Medium
Tags: string, hash, sort

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

class Solution:
def customSortString(self, order: str, s: str) -> str:
```

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

Python Solution:

```
class Solution(object):
def customSortString(self, order, s):
"""
:type order: str
:type s: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Custom Sort String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} order
* @param {string} s
* @return {string}
*/
var customSortString = function(order, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Custom Sort String
* Difficulty: Medium
* Tags: string, hash, sort
*
```

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

function customSortString(order: string, s: string): string {

};
```

C# Solution:

```
/*
* Problem: Custom Sort String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string CustomSortString(string order, string s) {

}
}
```

C Solution:

```
/*
* Problem: Custom Sort String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* customSortString(char* order, char* s) {
```

```
}
```

Go Solution:

```
// Problem: Custom Sort String
// Difficulty: Medium
// Tags: string, hash, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func customSortString(order string, s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun customSortString(order: String, s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func customSortString(_ order: String, _ s: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Custom Sort String
// Difficulty: Medium
// Tags: string, hash, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
impl Solution {
pub fn custom_sort_string(order: String, s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} order
# @param {String} s
# @return {String}
def custom_sort_string(order, s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $order
* @param String $s
* @return String
*/
function customSortString($order, $s) {

}
}
```

Dart Solution:

```
class Solution {
String customSortString(String order, String s) {

}
}
```

Scala Solution:

```
object Solution {
def customSortString(order: String, s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec custom_sort_string(order :: String.t, s :: String.t) :: String.t
def custom_sort_string(order, s) do

end
end
```

Erlang Solution:

```
-spec custom_sort_string(Order :: unicode:unicode_binary(), S ::
unicode:unicode_binary()) -> unicode:unicode_binary().
custom_sort_string(Order, S) ->
.
```

Racket Solution:

```
(define/contract (custom-sort-string order s)
(-> string? string? string?)
)
```