Problem 3039: Apply Operations to Make String Empty

Problem Information

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

Problem Description

You are given a string

s

.

Consider performing the following operation until

s

becomes

empty

:

For

every

alphabet character from

'a'

to

'z'

, remove the

first

occurrence of that character in

s

(if it exists).

For example, let initially

s = "aabcbbca"

. We do the following operations:

Remove the underlined characters

s = "

a

a

bc

bbca"

. The resulting string is

s = "abbca"

.

Remove the underlined characters

s = ''

ab

b

c

a''

. The resulting string is

s = "ba"

.

Remove the underlined characters

s = ''

ba

''

. The resulting string is

s = ""

.

Return

the value of the string

s

right

before

applying the

last

operation

. In the example above, answer is

"ba"

.

Example 1:

Input:

s = "aabcbbca"

Output:

"ba"

Explanation:

Explained in the statement.

Example 2:

Input:

s = "abcd"

Output:

"abcd"

Explanation:

We do the following operation: - Remove the underlined characters s = ''

abcd

''. The resulting string is s = ''''. The string just before the last operation is "abcd".

Constraints:

1 <= s.length <= 5 * 10

5

s

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string lastNonEmptyString(string s) {

}
};
```

Java:

```
class Solution {
public String lastNonEmptyString(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
var lastNonEmptyString = function(s) {

};
```

TypeScript:

```
function lastNonEmptyString(s: string): string {

};
```

C#:

```
public class Solution {
public string LastNonEmptyString(string s) {

}
}
```

C:

```
char* lastNonEmptyString(char* s) {

}
```

Go:

```
func lastNonEmptyString(s string) string {
```

```
}
```

Kotlin:

```
class Solution {
fun lastNonEmptyString(s: String): String {

}
}
```

Swift:

```
class Solution {
func lastNonEmptyString(_ s: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn last_non_empty_string(s: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @return {String}
def last_non_empty_string(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
*/
```

```
function lastNonEmptyString($s) {

}
}
```

Dart:

```
class Solution {
String lastNonEmptyString(String s) {

}
}
```

Scala:

```
object Solution {
def lastNonEmptyString(s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec last_non_empty_string(s :: String.t) :: String.t
def last_non_empty_string(s) do

end
end
```

Erlang:

```
-spec last_non_empty_string(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
last_non_empty_string(S) ->
.
```

Racket:

```
(define/contract (last-non-empty-string s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Apply Operations to Make String Empty
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string lastNonEmptyString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Apply Operations to Make String Empty
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String lastNonEmptyString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Apply Operations to Make String Empty
Difficulty: Medium
Tags: array, string, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Apply Operations to Make String Empty
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s
* @return {string}
*/
var lastNonEmptyString = function(s) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Apply Operations to Make String Empty
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function lastNonEmptyString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Apply Operations to Make String Empty
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string LastNonEmptyString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Apply Operations to Make String Empty
* Difficulty: Medium
```

```
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* lastNonEmptyString(char* s) {

}
```

Go Solution:

```
// Problem: Apply Operations to Make String Empty
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func lastNonEmptyString(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun lastNonEmptyString(s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func lastNonEmptyString(_ s: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Apply Operations to Make String Empty
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn last_non_empty_string(s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String}
def last_non_empty_string(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function lastNonEmptyString($s) {

}
}
```

Dart Solution:

```
class Solution {
String lastNonEmptyString(String s) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def lastNonEmptyString(s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec last_non_empty_string(s :: String.t) :: String.t
def last_non_empty_string(s) do

end
end
```

Erlang Solution:

```
-spec last_non_empty_string(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
last_non_empty_string(S) ->
.
```

Racket Solution:

```
(define/contract (last-non-empty-string s)
(-> string? string?)
)
```