Problem 316: Remove Duplicate Letters

Problem Information

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

Problem Description

Given a string

s

, remove duplicate letters so that every letter appears once and only once. You must make sure your result is

the smallest in lexicographical order

among all possible results.

Example 1:

Input:

s = "bcabc"

Output:

"abc"

Example 2:

Input:

s = "cbacdcbc"

Output:

"acdb"

Constraints:

1 <= s.length <= 10

4

s

consists of lowercase English letters.

Note:

This question is the same as 1081:

https://leetcode.com/problems/smallest-subsequence-of-distinct-characters/

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* removeDuplicateLetters(char* s) {

}
```

Go:

```
func removeDuplicateLetters(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (remove-duplicate-letters s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Duplicate Letters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Remove Duplicate Letters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String removeDuplicateLetters(String s) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Remove Duplicate Letters
Difficulty: Medium
Tags: string, graph, greedy, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove Duplicate Letters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Remove Duplicate Letters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function removeDuplicateLetters(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Remove Duplicate Letters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Remove Duplicate Letters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* removeDuplicateLetters(char* s) {

}
```

Go Solution:

```
// Problem: Remove Duplicate Letters
// Difficulty: Medium
// Tags: string, graph, greedy, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func removeDuplicateLetters(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Remove Duplicate Letters
// Difficulty: Medium
// Tags: string, graph, greedy, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (remove-duplicate-letters s)
(-> string? string?)
)
```