Problem 2734: Lexicographically Smallest String After Substring Operation

Problem Information

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

Problem Description

Given a string

s

consisting of lowercase English letters. Perform the following operation:

Select any non-empty

substring

then replace every letter of the substring with the preceding letter of the English alphabet. For example, 'b' is converted to 'a', and 'a' is converted to 'z'.

Return the

lexicographically smallest

string

after performing the operation

.

Example 1:

Input:

s = "cbabc"

Output:

"baabc"

Explanation:

Perform the operation on the substring starting at index 0, and ending at index 1 inclusive.

Example 2:

Input:

s = "aa"

Output:

"az"

Explanation:

Perform the operation on the last letter.

Example 3:

Input:

s = "acbbc"

Output:

"abaab"

Explanation:

Perform the operation on the substring starting at index 1, and ending at index 4 inclusive.

Example 4:

Input:

s = "leetcode"

Output:

"kddsbncd"

Explanation:

Perform the operation on the entire string.

Constraints:

1 <= s.length <= 3 * 10

5

s

consists of lowercase English letters

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* smallestString(char* s) {

}
```

Go:

```
func smallestString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

```
(define/contract (smallest-string s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest String After Substring Operation
* Difficulty: Medium
* Tags: string, tree, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest String After Substring Operation
* Difficulty: Medium
* Tags: string, tree, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest String After Substring Operation
Difficulty: Medium
Tags: string, tree, graph, greedy

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest String After Substring Operation
* Difficulty: Medium
* Tags: string, tree, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest String After Substring Operation
* Difficulty: Medium
* Tags: string, tree, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function smallestString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest String After Substring Operation
* Difficulty: Medium
* Tags: string, tree, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
```

```
public string SmallestString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest String After Substring Operation
* Difficulty: Medium
* Tags: string, tree, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* smallestString(char* s) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest String After Substring Operation
// Difficulty: Medium
// Tags: string, tree, graph, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func smallestString(s string) string {

}
```

Kotlin Solution:

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

}
```

```
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest String After Substring Operation
// Difficulty: Medium
// Tags: string, tree, graph, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
*/
function smallestString($s) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (smallest-string s)
(-> string? string?)
)
```