Problem 1830: Minimum Number of Operations to Make String Sorted

Problem Information

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

Problem Description

You are given a string

s

(

0-indexed

)■■■■■■. You are asked to perform the following operation on

s

until you get a sorted string:

Find

the largest index

i

such that

1 <= i < s.length

and

s[i] < s[i - 1]

.

Find

the largest index

j

such that

i <= j < s.length

and

s[k] < s[i - 1]

for all the possible values of

k

in the range

[i, j]

inclusive.

Swap the two characters at indices

i - 1

and

j

.

Reverse the suffix starting at index

i

.

Return

the number of operations needed to make the string sorted.

Since the answer can be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

s = "cba"

Output:

5

Explanation:

The simulation goes as follows: Operation 1: i=2, j=2. Swap s[1] and s[2] to get s="cab", then reverse the suffix starting at 2. Now, s="cab". Operation 2: i=1, j=2. Swap s[0] and s[2] to get s="bac", then reverse the suffix starting at 1. Now, s="bca". Operation 3: i=2, j=2. Swap s[1] and s[2] to get s="bac", then reverse the suffix starting at 2. Now, s="bac". Operation 4: i=1, j=1. Swap s[0] and s[1] to get s="abc", then reverse the suffix starting at 1. Now, s="acb".

Operation 5: i=2, j=2. Swap s[1] and s[2] to get s="abc", then reverse the suffix starting at 2. Now, s="abc".

Example 2:

Input:

s = "aabaa"

Output:

2

Explanation:

The simulation goes as follows: Operation 1: i=3, j=4. Swap s[2] and s[4] to get s="aaaab", then reverse the substring starting at 3. Now, s="aaaba". Operation 2: i=4, j=4. Swap s[3] and s[4] to get s="aaaab", then reverse the substring starting at 4. Now, s="aaaab".

Constraints:

1 <= s.length <= 3000

s

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int makeStringSorted(string s) {

}
};
```

Java:

```
class Solution {
public int makeStringSorted(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function makeStringSorted(s: string): number {

};
```

C#:

```
public class Solution {
public int MakeStringSorted(string s) {

}
}
```

C:

```
int makeStringSorted(char* s) {

}
```

Go:

```
func makeStringSorted(s string) int {

}
```

Kotlin:

```
class Solution {
fun makeStringSorted(s: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn make_string_sorted(s: String) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int makeStringSorted(String s) {

}
}
```

Scala:

```
object Solution {
def makeStringSorted(s: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec make_string_sorted(S :: unicode:unicode_binary()) -> integer().
make_string_sorted(S) ->
.
```

Racket:

```
(define/contract (make-string-sorted s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Operations to Make String Sorted
* Difficulty: Hard
* Tags: string, tree, math, sort
*
* 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:
int makeStringSorted(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Operations to Make String Sorted
* Difficulty: Hard
* Tags: string, tree, math, sort
*
* 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 int makeStringSorted(String s) {
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Operations to Make String Sorted
Difficulty: Hard
Tags: string, tree, math, sort

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 makeStringSorted(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make String Sorted
* Difficulty: Hard
* Tags: string, tree, math, sort
*
* 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 {number}
*/
var makeStringSorted = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make String Sorted
* Difficulty: Hard
* Tags: string, tree, math, sort
*
* 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 makeStringSorted(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Operations to Make String Sorted
* Difficulty: Hard
* Tags: string, tree, math, sort
*
* 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 int MakeStringSorted(string s) {

}

}
```

C Solution:

```
/*
* Problem: Minimum Number of Operations to Make String Sorted
* Difficulty: Hard
* Tags: string, tree, math, sort
*
* 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
*/

int makeStringSorted(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Number of Operations to Make String Sorted
// Difficulty: Hard
// Tags: string, tree, math, sort
//
// 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 makeStringSorted(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun makeStringSorted(s: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Operations to Make String Sorted
// Difficulty: Hard
// Tags: string, tree, math, sort
//
// 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 make_string_sorted(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int makeStringSorted(String s) {

}
}
```

Scala Solution:

```
object Solution {
def makeStringSorted(s: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec make_string_sorted(S :: unicode:unicode_binary()) -> integer().
make_string_sorted(S) ->
.
```

Racket Solution:

```
(define/contract (make-string-sorted s)
(-> string? exact-integer?)
)
```