Problem 1576: Replace All ?'s to Avoid Consecutive Repeating Characters

Problem Information

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

Problem Description

Given a string

s

containing only lowercase English letters and the

'?'

character, convert

all

the

'?'

characters into lowercase letters such that the final string does not contain any

consecutive repeating

characters. You

cannot

modify the non

'?'

characters.

It is

guaranteed

that there are no consecutive repeating characters in the given string

except

for

'?'

.

Return

the final string after all the conversions (possibly zero) have been made

. If there is more than one solution, return

any of them

. It can be shown that an answer is always possible with the given constraints.

Example 1:

Input:

s = "?zs"

Output:

"azs"

Explanation:

There are 25 solutions for this problem. From "azs" to "yzs", all are valid. Only "z" is an invalid modification as the string will consist of consecutive repeating characters in "zzs".

Example 2:

Input:

s = "ubv?w"

Output:

"ubvaw"

Explanation:

There are 24 solutions for this problem. Only "v" and "w" are invalid modifications as the strings will consist of consecutive repeating characters in "ubvvw" and "ubvww".

Constraints:

1 <= s.length <= 100

s

consist of lowercase English letters and

'?'

.

Code Snippets

C++:

```
class Solution {
public:
```

```
string modifyString(string s) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* modifyString(char* s) {

}
```

Go:

```
func modifyString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
* Difficulty: Easy
* Tags: string
*
* 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 modifyString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
* Difficulty: Easy
* Tags: string
*
* 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 modifyString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
Difficulty: Easy
Tags: string

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
* Difficulty: Easy
* Tags: string
*
* 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 modifyString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
* Difficulty: Easy
* Tags: string
*
* 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 modifyString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
* Difficulty: Easy
* Tags: string
*
```

```
* 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 ModifyString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
* Difficulty: Easy
* Tags: string
*
* 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* modifyString(char* s) {

}
```

Go Solution:

```
// Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
// Difficulty: Easy
// Tags: string
//
// 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 modifyString(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Replace All ?'s to Avoid Consecutive Repeating Characters
// Difficulty: Easy
// Tags: string
//
// 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 modify_string(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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