Problem 1957: Delete Characters to Make Fancy String

Problem Information

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

Problem Description

A

fancy string

is a string where no

three

consecutive

characters are equal.

Given a string

s

, delete the

minimum

possible number of characters from

s

to make it

fancy

.

Return

the final string after the deletion

. It can be shown that the answer will always be

unique

.

Example 1:

Input:

s = "le

e

etcode"

Output:

"leetcode"

Explanation:

Remove an 'e' from the first group of 'e's to create "leetcode". No three consecutive characters are equal, so return "leetcode".

Example 2:

Input:

s = "

a

aab

aa

aa"

Output:

"aabaa"

Explanation:

Remove an 'a' from the first group of 'a's to create "aabaaaa". Remove two 'a's from the second group of 'a's to create "aabaa". No three consecutive characters are equal, so return "aabaa".

Example 3:

Input:

s = "aab"

Output:

"aab"

Explanation:

No three consecutive characters are equal, so return "aab".

Constraints:

1 <= s.length <= 10

5

s

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
```

```
var makeFancyString = function(s) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* makeFancyString(char* s) {

}
```

Go:

```
func makeFancyString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (make-fancy-string s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Delete Characters to Make Fancy String
* 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 makeFancyString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Delete Characters to Make Fancy String
* 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 makeFancyString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Delete Characters to Make Fancy String
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 makeFancyString(self, s: str) -> str:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Delete Characters to Make Fancy String
* 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 makeFancyString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Delete Characters to Make Fancy String
* 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 makeFancyString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Delete Characters to Make Fancy String
* 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 MakeFancyString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Delete Characters to Make Fancy String
* 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* makeFancyString(char* s) {

}
```

Go Solution:

```
// Problem: Delete Characters to Make Fancy String
// 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 makeFancyString(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Delete Characters to Make Fancy String
// 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 make_fancy_string(s: String) -> String {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (make-fancy-string s)
(-> string? string?)
)
```