Problem 2957: Remove Adjacent Almost-Equal Characters

Problem Information

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

Problem Description

You are given a

0-indexed

string

word

.

In one operation, you can pick any index

i

of

word

and change

word[i]

to any lowercase English letter.

Return

the

minimum

number of operations needed to remove all adjacent

almost-equal

characters from

word

.

Two characters

a

and

b

are

almost-equal

if

a == b

or

a

and

b

are adjacent in the alphabet.

Example 1:

Input:

word = "aaaaa"

Output:

2

Explanation:

We can change word into "a

c

a

c

a" which does not have any adjacent almost-equal characters. It can be shown that the minimum number of operations needed to remove all adjacent almost-equal characters from word is 2.

Example 2:

Input:

word = "abddez"

Output:

2

Explanation:

We can change word into "

y

bd

o

ez" which does not have any adjacent almost-equal characters. It can be shown that the minimum number of operations needed to remove all adjacent almost-equal characters from word is 2.

Example 3:

Input:

word = "zyxyxyz"

Output:

3

Explanation:

We can change word into "z

a

x

a

x

a

z" which does not have any adjacent almost-equal characters. It can be shown that the minimum number of operations needed to remove all adjacent almost-equal characters from word is 3.

Constraints:

1 <= word.length <= 100

word

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int removeAlmostEqualCharacters(string word) {

}
};
```

Java:

```
class Solution {
public int removeAlmostEqualCharacters(String word) {

}
}
```

Python3:

```
class Solution:
def removeAlmostEqualCharacters(self, word: str) -> int:
```

Python:

```
class Solution(object):
def removeAlmostEqualCharacters(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} word
* @return {number}
*/
var removeAlmostEqualCharacters = function(word) {

};
```

TypeScript:

```
function removeAlmostEqualCharacters(word: string): number {

};
```

C#:

```
public class Solution {
public int RemoveAlmostEqualCharacters(string word) {

}
}
```

C:

```
int removeAlmostEqualCharacters(char* word) {

}
```

Go:

```
func removeAlmostEqualCharacters(word string) int {

}
```

Kotlin:

```
class Solution {
fun removeAlmostEqualCharacters(word: String): Int {

}
}
```

Swift:

```
class Solution {
func removeAlmostEqualCharacters(_ word: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn remove_almost_equal_characters(word: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} word
# @return {Integer}
def remove_almost_equal_characters(word)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @return Integer
*/
function removeAlmostEqualCharacters($word) {

}
}
```

Dart:

```
class Solution {
int removeAlmostEqualCharacters(String word) {

}
}
```

Scala:

```
object Solution {
def removeAlmostEqualCharacters(word: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec remove_almost_equal_characters(word :: String.t) :: integer
def remove_almost_equal_characters(word) do

end
end
```

Erlang:

```
-spec remove_almost_equal_characters(Word :: unicode:unicode_binary()) ->
integer().
remove_almost_equal_characters(Word) ->
.
```

Racket:

```
(define/contract (remove-almost-equal-characters word)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Adjacent Almost-Equal Characters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public:
int removeAlmostEqualCharacters(string word) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Adjacent Almost-Equal Characters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int removeAlmostEqualCharacters(String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Adjacent Almost-Equal Characters
Difficulty: Medium
Tags: string, dp, greedy

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def removeAlmostEqualCharacters(self, word: str) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def removeAlmostEqualCharacters(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Remove Adjacent Almost-Equal Characters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} word
* @return {number}
*/
var removeAlmostEqualCharacters = function(word) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Adjacent Almost-Equal Characters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

function removeAlmostEqualCharacters(word: string): number {

};
```

C# Solution:

```
/*
* Problem: Remove Adjacent Almost-Equal Characters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int RemoveAlmostEqualCharacters(string word) {

}
}
```

C Solution:

```
/*
* Problem: Remove Adjacent Almost-Equal Characters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int removeAlmostEqualCharacters(char* word) {

}
```

Go Solution:

```
// Problem: Remove Adjacent Almost-Equal Characters
// Difficulty: Medium
// Tags: string, dp, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func removeAlmostEqualCharacters(word string) int {

}
```

Kotlin Solution:

```
class Solution {
fun removeAlmostEqualCharacters(word: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func removeAlmostEqualCharacters(_ word: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Remove Adjacent Almost-Equal Characters
// Difficulty: Medium
// Tags: string, dp, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn remove_almost_equal_characters(word: String) -> i32 {
```

```
}
}
```

Ruby Solution:

```
# @param {String} word
# @return {Integer}
def remove_almost_equal_characters(word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @return Integer
*/
function removeAlmostEqualCharacters($word) {

}
}
```

Dart Solution:

```
class Solution {
int removeAlmostEqualCharacters(String word) {

}
}
```

Scala Solution:

```
object Solution {
def removeAlmostEqualCharacters(word: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec remove_almost_equal_characters(word :: String.t) :: integer
def remove_almost_equal_characters(word) do

end
end
```

Erlang Solution:

```
-spec remove_almost_equal_characters(Word :: unicode:unicode_binary()) ->
integer().
remove_almost_equal_characters(Word) ->
.
```

Racket Solution:

```
(define/contract (remove-almost-equal-characters word)
(-> string? exact-integer?)
)
```