Problem 1047: Remove All Adjacent Duplicates In String

Problem Information

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

Problem Description

You are given a string

s

consisting of lowercase English letters. A

duplicate removal

consists of choosing two

adjacent

and

equal

letters and removing them.

We repeatedly make

duplicate removals

on

s

until we no longer can.

Return

the final string after all such duplicate removals have been made

. It can be proven that the answer is

unique

.

Example 1:

Input:

s = "abbaca"

Output:

"ca"

Explanation:

For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

Example 2:

Input:

s = "azxxzy"

Output:

"ay"

Constraints:

1 <= s.length <= 10

5

s

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* removeDuplicates(char* s) {

}
```

Go:

```
func removeDuplicates(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (remove-duplicates s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove All Adjacent Duplicates In String
* Difficulty: Easy
* Tags: string, stack
```

```
*
* 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 removeDuplicates(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Remove All Adjacent Duplicates In String
* Difficulty: Easy
* Tags: string, stack
*
* 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 removeDuplicates(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove All Adjacent Duplicates In String
Difficulty: Easy
Tags: string, stack

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove All Adjacent Duplicates In String
* Difficulty: Easy
* Tags: string, stack
*
* 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 removeDuplicates = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove All Adjacent Duplicates In String
* Difficulty: Easy
```

```
* Tags: string, stack
*
* 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 removeDuplicates(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Remove All Adjacent Duplicates In String
* Difficulty: Easy
* Tags: string, stack
*
* 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 RemoveDuplicates(string s) {

}
}
```

C Solution:

```
/*
* Problem: Remove All Adjacent Duplicates In String
* Difficulty: Easy
* Tags: string, stack
*
* 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* removeDuplicates(char* s) {

}
```

Go Solution:

```
// Problem: Remove All Adjacent Duplicates In String
// Difficulty: Easy
// Tags: string, stack
//
// 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 removeDuplicates(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove All Adjacent Duplicates In String
// Difficulty: Easy
// Tags: string, stack
//
// 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 remove_duplicates(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (remove-duplicates s)
(-> string? string?)
)
```