Problem 1209: Remove All Adjacent Duplicates in String II

Problem Information

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

Problem Description

You are given a string

s

and an integer

k

, a

k

duplicate removal

consists of choosing

k

adjacent and equal letters from

s

and removing them, causing the left and the right side of the deleted substring to concatenate together.

We repeatedly make

k

duplicate removals

on

s

until we no longer can.

Return

the final string after all such duplicate removals have been made

. It is guaranteed that the answer is

unique

.

Example 1:

Input:

s = "abcd", k = 2

Output:

"abcd"

Explanation:

There's nothing to delete.

Example 2:

Input:

s = "deeedbbcccbdaa", k = 3

Output:

"aa"

Explanation:

First delete "eee" and "ccc", get "ddbbbdaa" Then delete "bbb", get "dddaa" Finally delete "ddd", get "aa"

Example 3:

Input:

s = "pbbcggttciiippooaais", k = 2

Output:

"ps"

Constraints:

1 <= s.length <= 10

5

2 <= k <= 10

4

s

only contains lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string removeDuplicates(string s, int k) {

}
};
```

Java:

```
class Solution {
public String removeDuplicates(String s, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function removeDuplicates(s: string, k: number): string {

};
```

C#:

```
public class Solution {
public string RemoveDuplicates(string s, int k) {

}
}
```

C:

```
char* removeDuplicates(char* s, int k) {

}
```

Go:

```
func removeDuplicates(s string, k int) string {

}
```

Kotlin:

```
class Solution {
fun removeDuplicates(s: String, k: Int): String {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn remove_duplicates(s: String, k: i32) -> String {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String removeDuplicates(String s, int k) {

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Remove All Adjacent Duplicates in String II
* Difficulty: Medium
* Tags: string, tree, stack
*
* 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:
```

```
string removeDuplicates(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Remove All Adjacent Duplicates in String II
* Difficulty: Medium
* Tags: string, tree, stack
*
* 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 String removeDuplicates(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove All Adjacent Duplicates in String II
Difficulty: Medium
Tags: string, tree, stack

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove All Adjacent Duplicates in String II
* Difficulty: Medium
* Tags: string, tree, stack
*
* 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
* @param {number} k
* @return {string}
*/
var removeDuplicates = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove All Adjacent Duplicates in String II
* Difficulty: Medium
* Tags: string, tree, stack
*
* 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 removeDuplicates(s: string, k: number): string {
```

```
};
```

C# Solution:

```
/*
* Problem: Remove All Adjacent Duplicates in String II
* Difficulty: Medium
* Tags: string, tree, stack
*
* 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 string RemoveDuplicates(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Remove All Adjacent Duplicates in String II
* Difficulty: Medium
* Tags: string, tree, stack
*
* 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
*/

char* removeDuplicates(char* s, int k) {

}
```

Go Solution:

```
// Problem: Remove All Adjacent Duplicates in String II
// Difficulty: Medium
```

```
// Tags: string, tree, stack
//
// 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 removeDuplicates(s string, k int) string {

}
```

Kotlin Solution:

```
class Solution {
fun removeDuplicates(s: String, k: Int): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove All Adjacent Duplicates in String II
// Difficulty: Medium
// Tags: string, tree, stack
//
// 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 remove_duplicates(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
String removeDuplicates(String s, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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