Problem 3561: Resulting String After Adjacent Removals

Problem Information

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

Problem Description

You are given a string

s

consisting of lowercase English letters.

You

must

repeatedly perform the following operation while the string

s

has

at least

two

consecutive

characters:

Remove the

leftmost

pair of

adjacent

characters in the string that are

consecutive

in the alphabet, in either order (e.g.,

'a'

and

'b'

, or

'b'

and

'a'

).

Shift the remaining characters to the left to fill the gap.

Return the resulting string after no more operations can be performed.

Note:

Consider the alphabet as circular, thus

'a'

and

'z'

are consecutive.

Example 1:

Input:

s = "abc"

Output:

"c"

Explanation:

Remove

"ab"

from the string, leaving

"c"

as the remaining string.

No further operations are possible. Thus, the resulting string after all possible removals is

"c"

.

Example 2:

Input:

s = "adcb"

Output:

""

Explanation:

Remove

"dc"

from the string, leaving

"ab"

as the remaining string.

Remove

"ab"

from the string, leaving

""

as the remaining string.

No further operations are possible. Thus, the resulting string after all possible removals is

""

.

Example 3:

Input:

s = "zadb"

Output:

"db"

Explanation:

Remove

"za"

from the string, leaving

"db"

as the remaining string.

No further operations are possible. Thus, the resulting string after all possible removals is

"db"

.

Constraints:

1 <= s.length <= 10

5

s

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
```

```
string resultingString(string s) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* resultingString(char* s) {

}
```

Go:

```
func resultingString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Resulting String After Adjacent Removals
* Difficulty: Medium
* 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 resultingString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Resulting String After Adjacent Removals
* Difficulty: Medium
* 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 resultingString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Resulting String After Adjacent Removals
Difficulty: Medium
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 resultingString(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Resulting String After Adjacent Removals
* Difficulty: Medium
* 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 resultingString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Resulting String After Adjacent Removals
* Difficulty: Medium
* 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 resultingString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Resulting String After Adjacent Removals
* Difficulty: Medium
* 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 ResultingString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Resulting String After Adjacent Removals
* Difficulty: Medium
* 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* resultingString(char* s) {

}
```

Go Solution:

```
// Problem: Resulting String After Adjacent Removals
// Difficulty: Medium
// 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 resultingString(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Resulting String After Adjacent Removals
// Difficulty: Medium
// 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 resulting_string(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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