Problem 1190: Reverse Substrings Between Each Pair of Parentheses

Problem Information

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

Problem Description

You are given a string

s

that consists of lower case English letters and brackets.

Reverse the strings in each pair of matching parentheses, starting from the innermost one.

Your result should

not

contain any brackets.

Example 1:

Input:

s = "(abcd)"

Output:

"dcba"

Example 2:

Input:

s = "(u(love)i)"

Output:

"iloveu"

Explanation:

The substring "love" is reversed first, then the whole string is reversed.

Example 3:

Input:

s = "(ed(et(oc))el)"

Output:

"leetcode"

Explanation:

First, we reverse the substring "oc", then "etco", and finally, the whole string.

Constraints:

1 <= s.length <= 2000

s

only contains lower case English characters and parentheses.

It is guaranteed that all parentheses are balanced.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* reverseParentheses(char* s) {

}
```

Go:

```
func reverseParentheses(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (reverse-parentheses s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Substrings Between Each Pair of Parentheses
* 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 reverseParentheses(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Substrings Between Each Pair of Parentheses
* 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 reverseParentheses(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse Substrings Between Each Pair of Parentheses
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 reverseParentheses(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Reverse Substrings Between Each Pair of Parentheses
* 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
* @return {string}
*/
var reverseParentheses = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Substrings Between Each Pair of Parentheses
* 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 reverseParentheses(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Reverse Substrings Between Each Pair of Parentheses
* 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 ReverseParentheses(string s) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Substrings Between Each Pair of Parentheses
* 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* reverseParentheses(char* s) {

}
```

Go Solution:

```
// Problem: Reverse Substrings Between Each Pair of Parentheses
// 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 reverseParentheses(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reverse Substrings Between Each Pair of Parentheses
// 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 reverse_parentheses(s: String) -> String {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (reverse-parentheses s)
(-> string? string?)
)
```