Problem 32: Longest Valid Parentheses

Problem Information

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

Problem Description

Given a string containing just the characters

'('

and

')'

, return

the length of the longest valid (well-formed) parentheses

substring

.

Example 1:

Input:

s = "(()"

Output:

2

Explanation:

The longest valid parentheses substring is "()".

Example 2:

Input:

s = ")()())"

Output:

4

Explanation:

The longest valid parentheses substring is "()()".

Example 3:

Input:

s = ""

Output:

0

Constraints:

0 <= s.length <= 3 * 10

4

s[i]

is

'('

, or

')'

.

Code Snippets

C++:

```
class Solution {
public:
int longestValidParentheses(string s) {

}
};
```

Java:

```
class Solution {
public int longestValidParentheses(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function longestValidParentheses(s: string): number {

};
```

C#:

```
public class Solution {
public int LongestValidParentheses(string s) {

}
}
```

C:

```
int longestValidParentheses(char* s) {

}
```

Go:

```
func longestValidParentheses(s string) int {

}
```

Kotlin:

```
class Solution {
fun longestValidParentheses(s: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn longest_valid_parentheses(s: String) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int longestValidParentheses(String s) {

}
}
```

Scala:

```
object Solution {
def longestValidParentheses(s: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec longest_valid_parentheses(S :: unicode:unicode_binary()) -> integer().
longest_valid_parentheses(S) ->
.
```

Racket:

```
(define/contract (longest-valid-parentheses s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Valid Parentheses
* Difficulty: Hard
* Tags: string, tree, dp, stack
*
* 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 longestValidParentheses(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Valid Parentheses
* Difficulty: Hard
* Tags: string, tree, dp, stack
*
* 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 longestValidParentheses(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Valid Parentheses
Difficulty: Hard
Tags: string, tree, dp, stack

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Valid Parentheses
* Difficulty: Hard
* Tags: string, tree, dp, stack
*
* 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} s
* @return {number}
*/
var longestValidParentheses = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Valid Parentheses
* Difficulty: Hard
* Tags: string, tree, dp, stack
*
* 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 longestValidParentheses(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Valid Parentheses
* Difficulty: Hard
* Tags: string, tree, dp, stack
*
* 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 LongestValidParentheses(string s) {

}
}
```

C Solution:

```
/*
* Problem: Longest Valid Parentheses
* Difficulty: Hard
* Tags: string, tree, dp, stack
*
* 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 longestValidParentheses(char* s) {

}
```

Go Solution:

```
// Problem: Longest Valid Parentheses
// Difficulty: Hard
// Tags: string, tree, dp, stack
//
// 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 longestValidParentheses(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestValidParentheses(s: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Valid Parentheses
// Difficulty: Hard
// Tags: string, tree, dp, stack
//
// 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 longest_valid_parentheses(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int longestValidParentheses(String s) {

}
}
```

Scala Solution:

```
object Solution {
def longestValidParentheses(s: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec longest_valid_parentheses(S :: unicode:unicode_binary()) -> integer().
longest_valid_parentheses(S) ->
.
```

Racket Solution:

```
(define/contract (longest-valid-parentheses s)
(-> string? exact-integer?)
)
```