Problem 1541: Minimum Insertions to Balance a Parentheses String

Problem Information

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

Problem Description

Given a parentheses string

s

containing only the characters

'('

and

')'

. A parentheses string is

balanced

if:

Any left parenthesis

'('

must have a corresponding two consecutive right parenthesis

'))'

.

Left parenthesis

'('

must go before the corresponding two consecutive right parenthesis

'))'

.

In other words, we treat

'('

as an opening parenthesis and

'))'

as a closing parenthesis.

For example,

"())"

,

"())(()))"

and

"(())()))"

are balanced,

")()"

,

"()))"

and

"(()))"

are not balanced.

You can insert the characters

'('

and

')'

at any position of the string to balance it if needed.

Return

the minimum number of insertions

needed to make

s

balanced.

Example 1:

Input:

s = "(()))"

Output:

1

Explanation:

The second '(' has two matching '))', but the first '(' has only ')' matching. We need to add one more ')' at the end of the string to be "(())))" which is balanced.

Example 2:

Input:

s = "())"

Output:

0

Explanation:

The string is already balanced.

Example 3:

Input:

s = "))())("

Output:

3

Explanation:

Add '(' to match the first '))', Add '))' to match the last '('.

Constraints:

1 <= s.length <= 10

s

consists of

'('

and

')'

only.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minInsertions(self, s):
```

```
"""
:type s: str
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minInsertions(char* s) {

}
```

Go:

```
func minInsertions(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-insertions s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Insertions to Balance a Parentheses String
* Difficulty: Medium
* Tags: string, greedy, 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:
int minInsertions(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Insertions to Balance a Parentheses String
* Difficulty: Medium
* Tags: string, greedy, 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 int minInsertions(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Insertions to Balance a Parentheses String
Difficulty: Medium
Tags: string, greedy, 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 minInsertions(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Insertions to Balance a Parentheses String
* Difficulty: Medium
* Tags: string, greedy, 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 {number}
*/
var minInsertions = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Insertions to Balance a Parentheses String
* Difficulty: Medium
* Tags: string, greedy, 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 minInsertions(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Insertions to Balance a Parentheses String
* Difficulty: Medium
* Tags: string, greedy, 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 int MinInsertions(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Insertions to Balance a Parentheses String
* Difficulty: Medium
* Tags: string, greedy, 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
```

```
*/

int minInsertions(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Insertions to Balance a Parentheses String
// Difficulty: Medium
// Tags: string, greedy, 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 minInsertions(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Insertions to Balance a Parentheses String
// Difficulty: Medium
// Tags: string, greedy, 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 min_insertions(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-insertions s)
(-> string? exact-integer?)
)
```