

Problem 1614: Maximum Nesting Depth of the Parentheses

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a

valid parentheses string

s

, return the

nesting depth

of

s

. The nesting depth is the

maximum

number of nested parentheses.

Example 1:

Input:

```
s = "(1+(2*3)+((8)/4))+1"
```

Output:

3

Explanation:

Digit 8 is inside of 3 nested parentheses in the string.

Example 2:

Input:

```
s = "(1)+((2))+(((3)))"
```

Output:

3

Explanation:

Digit 3 is inside of 3 nested parentheses in the string.

Example 3:

Input:

```
s = "()()((((())))"
```

Output:

3

Constraints:

$1 \leq s.length \leq 100$

s

consists of digits

0-9

and characters

'+'

,

'_'

,

'*'!

,

'/'

,

'('

, and

')'

.

It is guaranteed that parentheses expression

s

is a VPS.

Code Snippets

C++:

```
class Solution {  
public:  
    int maxDepth(string s) {  
  
    }  
};
```

Java:

```
class Solution {  
public int maxDepth(String s) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function maxDepth(s: string): number {  
}  
};
```

C#:

```
public class Solution {  
    public int MaxDepth(string s) {  
  
    }  
}
```

C:

```
int maxDepth(char* s) {  
  
}
```

Go:

```
func maxDepth(s string) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxDepth(s: String): Int {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return Integer  
     */  
    function maxDepth($s) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int maxDepth(String s) {  
  
    }  
}
```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (max-depth s)
  (-> string? exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum Nesting Depth of the Parentheses
 * Difficulty: Easy
 * 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:
  int maxDepth(string s) {

  }
};
```

Java Solution:

```
/**  
 * Problem: Maximum Nesting Depth of the Parentheses  
 * Difficulty: Easy  
 * 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 int maxDepth(String s) {  
  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Maximum Nesting Depth of the Parentheses  
Difficulty: Easy  
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 maxDepth(self, s: str) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**  
 * Problem: Maximum Nesting Depth of the Parentheses  
 * Difficulty: Easy  
 * 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 {number}  
 */  
var maxDepth = function(s) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Maximum Nesting Depth of the Parentheses  
 * Difficulty: Easy  
 * 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 maxDepth(s: string): number {  
  
};
```

C# Solution:

```

/*
 * Problem: Maximum Nesting Depth of the Parentheses
 * Difficulty: Easy
 * 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 int MaxDepth(string s) {
        return 0;
    }
}

```

C Solution:

```

/*
 * Problem: Maximum Nesting Depth of the Parentheses
 * Difficulty: Easy
 * 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
 */

int maxDepth(char* s) {
    return 0;
}

```

Go Solution:

```

// Problem: Maximum Nesting Depth of the Parentheses
// Difficulty: Easy
// 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 maxDepth(s string) int {  
    }  
}
```

Kotlin Solution:

```
class Solution {  
    fun maxDepth(s: String): Int {  
        }  
    }  
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Maximum Nesting Depth of the Parentheses  
// Difficulty: Easy  
// 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 max_depth(s: String) -> i32 {  
        }  
    }  
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return Integer  
     */  
    function maxDepth($s) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
int maxDepth(String s) {  
  
}  
}
```

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-depth s)  
(-> string? exact-integer?)  
) 
```