Codility_

Candidate Report: Anonymous

Check out Codility training tasks

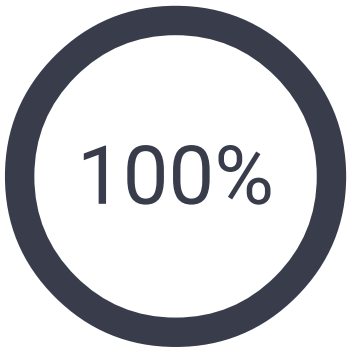
Test Name:

Summary Timeline Feedback

Tasks summary

Task	Time spent	Score
Nesting Swift 4	9 min	100%

Total score



Tasks Details

Easy

1. Nesting

Determine whether a given string of parentheses (single type) is properly nested.

Task Score	Correctness	Performance
100%	100%	100%

Task description

A string S consisting of N characters is called *properly nested* if:

- S is empty;
- S has the form "(U)" where U is a properly nested string;
- S has the form "VW" where V and W are properly nested strings.

For example, string "(()(())())" is properly nested but string "())" isn't.

Write a function:

```
public func solution(_ S : inout String) -> Int
```

that, given a string S consisting of N characters, returns 1 if string S is properly nested and 0 otherwise.

For example, given S = "(()(())())", the function should return 1 and given S = "())", the function should return 0, as explained above.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [0..1,000,000];
- string S consists only of the characters "(" and/or ")".

Solution

Programming language used:	Swift 4
Total time used:	9 minutes
Effective time used:	9 minutes
Notes:	*not defined yet*

Task timeline

10:52:52 11:01:34

Code: 11:01:34 UTC, swift4, final, score: **100** show code in pop-up

```
1  import Foundation
2  import Glibc
3
4  public struct Stack<T> {
5      fileprivate var array = [T]()
6
7      public var isEmpty: Bool {
```

```
8           return array.isEmpty
9       }
10
11      public var count: Int {
12          return array.count
13      }
14
15      public mutating func push(_ element: T) {
16          array.append(element)
17      }
18
19      public mutating func pop() -> T? {
20          return array.popLast()
21      }
22
23      public var top: T? {
24          return array.last
25      }
26  }
27
28  extension Stack: Sequence {
29      public func makeIterator() -> AnyIterator<T> {
30          var curr = self
31          return AnyIterator {
32              return curr.pop()
33          }
34      }
35  }
36
37
38  public func solution(_ S : inout String) -> Int {
39
40      var stack = Stack<Character>()
41      for char in S {
42          if let top = stack.top {
43              if top == "(" && char == ")" {
44                  stack.pop()
45              }else{
46                  stack.push(char)
47              }
48          }else{
49              stack.push(char)
50          }
51      }
52
53      return stack.count == 0 ? 1 : 0
54  }
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

expand all	Example tests	
▶	example1 example test	✔ OK
▶	example2 example test2	✔ OK
expand all	Correctness tests	
▶	negative_match invalid structure, but the number of parentheses matches	✔ OK
▶	empty empty string	✔ OK
▶	simple_grouped simple grouped positive and negative test, length=22	✔ OK

▶	small_random	✔ OK

expand all Performance tests

▶	large1 simple large positive and negative test, 10K or 10K+1 ('s followed by 10K)'s	✔ OK
▶	large_full_ternary_tree tree of the form T=(TTT) and depth 11, length=177K+	✔ OK
▶	multiple_full_binary_trees sequence of full trees of the form T=(TT), depths [1..10..1], with/without unmatched ')' at the end, length=49K+	✔ OK
▶	broad_tree_with_deep_paths string of the form (TTT...T) of 300 T's, each T being '((((...))))' nested 200-fold, length=1 million	✔ OK