Codility_

## CodeCheck Report: trainingD3UDWP-CH4

Test Name:

Check out Codility training tasks

Summary Timeline

### Tasks summary

| Task | Time spent | Score |
|---|---|---|
| Brackets<br>Java 8 | 3 min | 100% |

### Total score


## Tasks Details

### 1. Brackets

Easy

Determine whether a given string of parentheses (multiple types) is properly nested.

| Task Score | Correctness | Performance |
|---|---|---|
| 100% | 100% | 100% |

### Task description

A string S consisting of N characters is considered to be *properly nested* if any of the following conditions is true:

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

For example, the string "`{[()()]}`" is properly nested but "`([)()]`" is not.

Write a function:

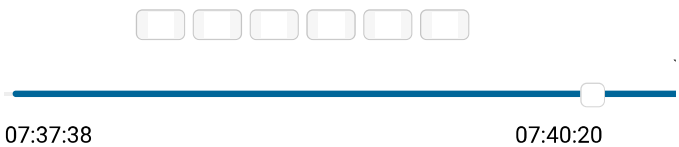
```
class Solution { public int solution(String S); }
```

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

### Solution

| | |
|---|---|
| Programming language used: | Java 8 |
| Total time used: | 3 minutes |
| Effective time used: | 3 minutes |
| Notes: | *not defined yet* |

### Task timeline



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..200,000];
- string S consists only of the following characters: "`(`", "`{`", "`[`", "`]`", "`}`" and/or "`)`".



Code: 07:40:19 UTC, java, final, score: **100** show code in pop-up

```
// you can also use imports, for example:
// import java.util.*;
import java.util.ArrayList;
// you can write to stdout for debugging purpo
// System.out.println("this is a debug message

class Solution {
    static ArrayList<String> bracketStack = ne

        public static void push(ArrayList<Stri
                stringList.add(s);
        }

        public static String pop(ArrayList<Str
                return (stringList.isEmpty()?"
        }
    public int solution(String S) {
        // write your code in Java SE 8

        int output = -1;
                boolean isNestedString = true;
                if(!S.isEmpty()) {
                        int index = 0;
                        int size = S.length();
                        boolean continueWhile
                        do {
                                //char current
                                String current

                                switch (curren
                                case ")":
                                        if (!p


                                        }
                                        break;
                                case "}":
                                        if (!p


                                        }
                                        break;
                                case "]":
                                        if (!p


                                        }

                                        break;

                                default:
                                        push(b
                                        break;
                                }

                                index++;
                        } while (index<size&&c

                }

                if(isNestedString&&bracketStac
                        output = 1;
                }else {
                        output = 0;
                }

                return output;

    }
```

```
70  }
```

## Analysis summary

The solution obtained perfect score.

## Analysis

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

| expand all | Example tests | |
|---|---|---|
| ▶ | example1<br>example test 1 | ✓ OK |
| ▶ | example2<br>example test 2 | ✓ OK |
| expand all | **Correctness tests** | |
| ▶ | negative_match<br>invalid structures | ✓ OK |
| ▶ | empty<br>empty string | ✓ OK |
| ▶ | simple_grouped<br>simple grouped positive and negative test, length=22 | ✓ OK |
| expand all | **Performance tests** | |
| ▶ | large1<br>simple large positive test, 100K ('s followed by 100K )'s + )( | ✓ OK |
| ▶ | large2<br>simple large negative test, 10K+1 ('s followed by 10K )'s + )( + () | ✓ OK |
| ▶ | large_full_ternary_tree<br>tree of the form T=(TTT) and depth 11, length=177K+ | ✓ OK |
| ▶ | multiple_full_binary_trees<br>sequence of full trees of the form T=(TT), depths [1..10..1], with/without some brackets at the end, length=49K+ | ✓ OK |
| ▶ | broad_tree_with_deep_paths<br>string of the form [TTT...T] of 300 T's, each T being '{{{...}}}' nested 200-fold, length=120K+ | ✓ OK |