# Codility\_

# CodeCheck Report: training8K58J8-EP6

Test Name:

Check out Codility training tasks

Summary Timeline

#### Mail Status: Not Applicable

Finished: 2021-10-21 23:38 UTC

Started: 2021-10-21 23:23 UTC

Invitation Created: 2021-10-21 23:23 UTC

#### **Tasks Details**

Easy

#### 1. Brackets

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

Task Score 100% Correctness

Performance

100%

orrorrinarioe

100%

23:38:20

#### 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:

int solution(string &S);

that, given a string S consisting of N characters, returns 1 if 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.

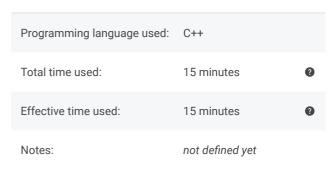
Write an efficient algorithm for the following assumptions:

- N is an integer within the range [0..200,000];
- string S consists only of the following characters: "(", "{", "[", "]", "}" and/or ")".

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

#### Solution

23:23:46







Code: 23:38:20 UTC, cpp, show code in pop-up final, score: 100

```
// you can use includes, for example:
     // #include <algorithm>
 3
     // you can write to stdout for debugging purpo
 4
     // cout << "this is a debug message" << endl;</pre>
 5
     #include <bits/stdc++.h>
 7
     int solution(string &S) {
 8
         vector<int> A;
 9
         int n=S.size(),cont;
10
         if(S.empty()){
11
              return 1;
12
13
         for(int i=0;i<n;i++){</pre>
```

```
if(S[i]=='['){
15
                 cont=+1;
16
17
             if(S[i]==']'){
18
                 cont=-1;
19
             }
             if(S[i]=='('){
20
21
                 cont=+2;
22
23
             if(S[i]==')'){
24
                 cont=-2;
25
             }
             if(S[i]=='{'){
26
27
                 cont=+3;
28
29
             if(S[i]=='}'){
30
                 cont=-3;
31
32
             if((i==0 && cont<0) || (cont<0 && cont
33
             else if(cont<0){A.pop_back();}</pre>
34
             else{A.push_back(cont);}
35
36
         if(A.empty()){return 1;}
37
         else {return 0;}
38
39
     }
```

## Analysis summary

The solution obtained perfect score.

### Analysis

Detected time complexity: O(N)

Detected time complexit	, O(N)
expand all Example tests	
example1 example test 1	✓ OK
example2 example test 2	✓ OK
expand all Correctness to	ests
negative_match invalid structures	✓ OK
empty empty string	✓ OK
simple_grouped simple grouped positive and negative test, length=22	✓ OK
expand all Performance to	ests
► large1 simple large positive test, 100K ('s followed by 100K )'s + )(	✓ OK
► large2 simple large negative test, 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=	✓ OK

(TT), depths [1..10..1], with/without some brackets at the end, length=49K+

► broad\_tree\_with\_deep\_paths 
string of the form [TTT...T] of 300 T's,
each T being '{{...}}' nested 200-fold,
length=120K+