Codility_

CodeCheck Report: training8K58J8-EP6

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
Brackets C++ ⚠	15 min	100%

Total score

100%

Tasks Details

Easy

1. **Brackets**
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:

```
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];

Solution

Programming language used:	C++
Total time used:	15 minutes
Effective time used:	15 minutes
Notes:	*not defined yet*

Task timeline

23:23:46 23:38:20

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

```
// you can use includes, for example:
// #include <algorithm>

```

- 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.

```
// you can write to stdout for debugging purpo
// cout << "this is a debug message" << endl;
#include <bits/stdc++.h>
int solution(string &S) {
    vector<int> A;
    int n=S.size(),cont;
    if(S.empty()){
        return 1;
    }
    for(int i=0;i<n;i++){
        if(S[i]=='['){
            cont=+1;
        }
        if(S[i]==']'){
            cont=-1;
        }
        if(S[i]=='('){
            cont=+2;
        }
        if(S[i]==')'){
            cont=-2;
        }
        if(S[i]=='{'){
            cont=+3;
        }
        if(S[i]=='}'){
            cont=-3;
        }
        if((i==0 && cont<0) || (cont<0 && cont
        else if(cont<0){A.pop_back();}
        else{A.push_back(cont);}
    }
    if(A.empty()){return 1;}
    else {return 0;}

}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶	example1 example test 1	✔ OK
▶	example2 example test 2	✔ OK
expand all	**Correctness tests**	
▶	negative_match invalid structures	✔ OK
▶	empty empty string	✔ OK
▶	simple_grouped simple grouped positive and negative test, length=22	✔ OK
expand all	**Performance tests**	
▶	large1 simple large positive test, 100K ('s followed by 100K)'s +)(	✔ OK
▶	large2 simple large negative test, 10K+1 ('s	✔ OK

	followed by 10K)'s +)(+ ()	
▶	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 some brackets 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=120K+	✔ OK