codility

Check out Codility training tasks

Candidate Report: Anonymous

Test Name:

Summary Timeline

Test Score

100 out of 100 points

100%

Tasks in Test

	Time Spent	Task Score
Brackets Submitted in: C++	1 min	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.

Solution

Programming language used:	C++
Total time used:	1 minutes
Effective time used:	1 minutes
Notes:	*not defined yet*

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 ")".

Task timeline

06:26:22 06:27:14

Code: 06:27:14 UTC, cpp, final, score: **100** show code in pop-up

```cpp
#include <string>
#include <stack>

int matches(std::stack<char>& chars, char expect
   char startb = chars.top();
   chars.pop();
   if (startb != expected) {
      return 0;
   }
   return 1;
}

int solution(std::string& S)
{
   int result = 1;
   std::stack<char> chars;
   if (!S.empty()) {
      for (auto c : S) {
         switch (c) {
         case '(':
         case '{':
         case '[':
            chars.push(c);
            break;
         case ')': result = matches(chars, '(');
            break;
         case ']': result = matches(chars, '[');
            break;
         case '}': result = matches(chars, '{');
            break;
         }
         if (!result) {
            break;
         }
      }
   }
   if (!chars.empty()) {
      result = 0;
   }
   return result;
}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

collapse all — Example tests

▼ example1 example test 1		✓ OK
1.	0.001 s	OK
▼ example2 example test 2		✓ OK
1.	0.001 s	OK

collapse all — Correctness tests

▼ negative_match invalid structures		✓ OK
1.	0.001 s	OK
2.	0.001 s	OK
3.	0.001 s	OK
4.	0.001 s	OK
5.	0.001 s	OK
▼ empty empty string		✓ OK
1.	0.001 s	OK
▼ simple_grouped simple grouped positive and negative test, length=22		✓ OK
1.	0.001 s	OK
2.	0.001 s	OK

3.	0.001 s	OK
4.	0.001 s	OK
5.	0.001 s	OK

collapse all

Performance tests

large1 ✓ OK

simple large positive test, 100K ('s followed by 100K)'s +)(

1.	0.004 s	OK
2.	0.001 s	OK
3.	0.001 s	OK

large2 ✓ OK

simple large negative test, 10K+1 ('s followed by 10K)'s +)(+ ()

1.	0.001 s	OK
2.	0.001 s	OK
3.	0.001 s	OK

large_full_ternary_tree ✓ OK

tree of the form T=(TTT) and depth 11, length=177K+

1.	0.004 s	OK

multiple_full_binary_trees ✓ OK

sequence of full trees of the form T=(TT), depths [1..10..1], with/without some brackets at the end, length=49K+

1.	0.001 s	OK
2.	0.001 s	OK
3.	0.001 s	OK
4.		

	0.001 s	OK
5.	0.001 s	OK

▼ broad_tree_with_deep_paths ✓ OK

string of the form [TTT...T] of 300 T's, each T being '{{{...}}}' nested 200-fold, length=120K+

1.	0.004 s	OK
2.	0.004 s	OK

PDF version of this report that may be downloaded on top of this site may contain sensitive data including personal information. For security purposes, we recommend you remove it from your system once reviewed.