Codility_

CodeCheck Report: trainingFJCD5R-VEK

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
DisappearingPairs ⚠ Java 11	35 min	100%

Total score

100%

Tasks Details

1. DisappearingPairs
Reduce a string containing instances of the letters "A", "B" and "C" via the following rule: remove one occurrence of "AA", "BB" or "CC".

Hard

Task Score	Correctness	Performance
100%	100%	100%

Task description

A string S containing only the letters "A", "B" and "C" is given. The string can be transformed by removing one occurrence of "AA", "BB" or "CC".

Transformation of the string is the process of removing letters from it, based on the rules described above. As long as at least one rule can be applied, the process should be repeated. If more than one rule can be used, any one of them could be chosen.

Write a function:

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

that, given a string S consisting of N characters, returns any string that can result from a sequence of transformations as described above.

For example, given string S = "ACCAABBC" the function may return "AC", because one of the possible sequences of transformations is as follows:

Solution

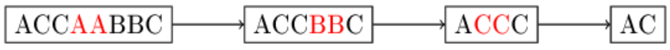
Programming language used:	Java 11
Total time used:	35 minutes
Effective time used:	35 minutes
Notes:	*not defined yet*

Task timeline

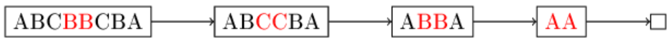
22:28:51 23:03:48

Code: 23:03:48 UTC, java11, final, score: **100**

show code in pop-up



Also, given string S = "ABCBBCBA" the function may return "", because one possible sequence of transformations is:



Finally, for string S = "BABABA" the function must return "BABABA", because no rules can be applied to string S.

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

- the length of string S is within the range [0..50,000];
- string S is made only of the following characters: 'A', 'B' and/or 'C'.

```
// you can also use imports, for example:
import java.util.*;

// you can write to stdout for debugging purpos
// System.out.println("this is a debug message"

class Solution {
    public String solution(String S) {
                StringBuffer sb = new StringBuf

                char last = 0;

                for (char ch : S.toCharArray())
                        if (ch == last) {
                                sb.deleteCharAt
                                last = sb.lengt
                        } else {
                                sb.append(ch);
                                last = ch;
                        }
                }

                return sb.toString();
    }
}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

collapse all **Example tests**

▼ example1 ✔ OK
first example test

1. 0.012 s **OK**

▼ example2 ✔ OK
second example test

1. 0.012 s **OK**

▼ example3 ✔ OK
third example test

1. 0.012 s **OK**

collapse all **Correctness tests**

▼ empty ✔ OK
empty string

1. 0.012 s **OK**

▼ one_char ✔ OK
single-character strings

1. 0.012 s **OK**
2. 0.012 s **OK**
3. 0.012 s **OK**

▼ simple ✔ OK
A^3, B^4 and C^5

1. 0.012 s **OK**
2. 0.008 s **OK**
3. 0.008 s **OK**

▼ short_palindrome ✔ **OK**
short palindrome

1. 0.012 s **OK**
2. 0.012 s **OK**
3. 0.012 s **OK**
4. 0.008 s **OK**

▼ tricky ✔ **OK**
tricky folding

1. 0.012 s **OK**

▼ easy_greedy ✔ **OK**
any greedy approach should pass

1. 0.012 s **OK**

collapse all

Performance tests

▼ max_rand ✔ **OK**
random max tests

1. 0.080 s **OK**
2. 0.084 s **OK**

▼ max_C ✔ **OK**
max test with letters C only

1. 0.076 s **OK**

▼ complicated ✔ **OK**
random big test, complicated folding

1. 0.064 s **OK**

▼ odd_palindrome ✔ **OK**
big palindrome of odd length

1. 0.056 s **OK**

▼ even_palindrome1 ✔ **OK**
big palindrome of even length

1. 0.084 s **OK**

▼ even_palindrome2 ✔ **OK**
big palindrome of even length

1. 0.084 s **OK**