Codility_

CodeCheck Report: training26RHW9-5CH

Test Name:

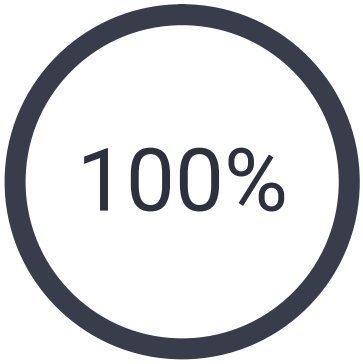
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
DisappearingPairs Python	5 min	100%

Total score



Tasks Details

1. **DisappearingPairs**

Hard

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

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:

```
def solution(S)
```

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

Solution

Programming language used:	Python
Total time used:	5 minutes
Effective time used:	5 minutes
Notes:	*not defined yet*

Task timeline

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

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'.

08:46:01 08:50:07

Code: 08:50:06 UTC, py, final, score: **100** show code in pop-up

```
1  # you can write to stdout for debugging pu
2  # print("this is a debug message")
3
4  def solution(S):
5      # Implement your solution here
6      arr = list(S)
7      length = len(arr)
8      write_index = 0
9
10     for read_index in range(length):
11         if write_index > 0 and arr[read_in
12             # Found a pair of consecutive
13             write_index -= 1
14         else:
15             # Keep the current character s
16             arr[write_index] = arr[read_in
17             write_index += 1
18
19     return ''.join(arr[:write_index])
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶	example1 first example test	✔ OK
▶	example2 second example test	✔ OK
▶	example3 third example test	✔ OK
expand all	**Correctness tests**	
▶	empty empty string	✔ OK
▶	one_char single-character strings	✔ OK
▶	simple A^3, B^4 and C^5	✔ OK
▶	short_palindrome short palindrome	✔ OK
▶	tricky tricky folding	✔ OK
▶	easy_greedy any greedy approach should pass	✔ OK
expand all	**Performance tests**	
▶	max_rand random max tests	✔ OK

▶	max_C max test with letters C only	✔ OK
▶	complicated random big test, complicated folding	✔ OK
▶	odd_palindrome big palindrome of odd length	✔ OK
▶	even_palindrome1 big palindrome of even length	✔ OK
▶	even_palindrome2 big palindrome of even length	✔ OK