Minimum String Length After Removing Substrings

solved by	Senan
Platform	LeetCode
difficulty	Easy
Serial	2696
tags	Stack
language	C++
solved on	@07/10/2024
link	https://leetcode.com/problems/minimum-string-length-after-removing-substrings/description/
Completion	✓

Intuition

We are given a string `s`, and we want to reduce its length by performing the following operations: remove the substrings "AB" and "CD" whenever they appear in the string. The goal is to determine the minimum length of the string after all possible reductions are applied. The approach leverages a stack to efficiently remove matching pairs as we traverse the string.

Approach

We use a stack to simulate the removal of "AB" and "CD" pairs. As we iterate through the string `s`, we check whether the top of the stack forms one of these pairs with the current character. If it does, we remove the top of the stack (i.e., pop it), as the pair "AB" or "CD" is eliminated. Otherwise, we push the current character onto the stack. At the end of the traversal, the size of the stack gives the length of the reduced string.

1. Initialize a stack and start with a dummy character (e.g., '#') to handle empty cases.
2. Traverse through each character of the string.
3. Check if the top of the stack forms either "AB" or "CD" with the current character.
4. If so, pop the stack (remove the pair). Otherwise, push the character onto the stack.
5. After traversing the entire string, the size of the stack minus one (due to the dummy character) gives the final reduced length.

Complexity

Time Complexity:

The time complexity of this approach is **O(n)**, where `n` is the length of the string `s`. We iterate through the string once, and each operation on the stack (push or pop) takes constant time, O(1).

Space Complexity:

The space complexity is **O(n)**, where `n` is the length of the string `s`. In the worst case, no pairs are removed, and we store all characters of `s` in the stack, making the space complexity linear.

Code

```
class Solution {
public:
    int minLength(string s) {
        stack<char> st;
        st.push('#');  // Dummy character
        for (auto ch : s) {
            if (st.top() == 'A' && ch == 'B') st.pop();
            else if (st.top() == 'C' && ch == 'D') st.pop();
            else st.push(ch);
        }
        return st.size() - 1;
    }
};
```