Remove All Adjacent Duplicates in String LeetCode

You are given a string that may contain adjacent duplicate characters. Your task is to remove these adjacent duplicates and return the modified string.

Example: Input: "abbaca" Output: "ca" Explanation: The input string contains adjacent duplicates "bb" and "aa". After removing these duplicates, the resulting string is "ca".

Approach 1: Function to remove adjacent duplicates from the string by erasing them

- It iterates through the string using a while loop and a pointer **i**.
- If the current character (**str[i]**) is equal to the next character (**str[i+1]**), it indicates an adjacent duplicate.
- In such cases, the function erases the adjacent duplicates from the string using **str.erase(i, 2)**.
- If **i** is not at the beginning of the string (**i != 0**), it decrements **i** to check for new adjacent duplicates.
- If the characters are not duplicates, it moves to the next character by incrementing **i**.
- The function returns the modified string.
- **The time complexity is O(n^2) due to the potential erase operation in each iteration, where n is the length of the string.**
- **The space complexity is O(1) as it modifies the original string in-place without using additional space.**

Approach 2: Function to remove adjacent duplicates from the string using a stack

- It uses a stack to remove adjacent duplicates.
- It iterates through the string using a range-based for loop.
- If the stack is not empty and the top of the stack (**st.top()**) is equal to the current character (**s**), it indicates an adjacent duplicate.
- In such cases, the duplicate character is removed from the stack using **st.pop()**.
- If the characters are not duplicates, the non-duplicate character is pushed onto the stack using **st.push(s)**.
- After the loop, the resulting string is constructed by popping characters from the stack and concatenating them in reverse order.
- The function returns the resulting string.

- **The time complexity is O(n) as it requires a single pass through the string.**
- **The space complexity is O(n) as it uses a stack to store non-duplicate characters, where n is the length of the string.**

Which Function to Use:

- If memory usage is a concern and you can modify the original string, Approach 1 (**removeDuplicates**) can be used.
- If you prefer to keep the original string intact or memory usage is not a concern, Approach 2 (**removeDuplicatesUsingStack**) can be used.