Remove all occurrences of Substring from a given String LeetCode

Given a string and a substring, write a function to remove all occurrences of the substring from the string and return the modified string.

Input: String: "daabcbaabcbc" Substring: "abc"

Output: Modified String: "dab"

Explanation: In this example, the original string is "daabcbaabcbc" and the substring to remove is "abc". We need to remove all occurrences of "abc" from the string.

- The first occurrence of "abc" is found at index 2 in the string. It is removed, resulting in the string "dabbaabcbc".
- The next occurrence of "abc" is found at index 6 in the string. It is removed, resulting in the string "dabbaabcbc".
- There are no more occurrences of "abc" in the string, so we stop. The modified string is "dab".

Approach 1: Function to remove all occurrences of a substring from a given string

The code provided defines the **removeOccurrences** function, which takes two string references: **str** represents the original string, and **part** represents the substring to remove.

- The **removeOccurrences** function uses a while loop to iterate until either the string becomes empty (**str.length() != 0**) or the substring is not found anymore (**str.find(part) < str.length()**).
- Inside the loop, it finds the position of the first occurrence of the substring in the string using **str.find(part)**.
- The **erase** function is then used to remove the substring from the string at the found position (**str.erase(str.find(part), part.length())**).
- The loop continues until all occurrences of the substring are removed or the string is empty.
- Finally, the modified string is returned.

Time Complexity: The time complexity of the code is O(n * m), where n is the length of the string and m is the length of the substring.

Note: The time complexity of the code depends on the length of the string (**n**) and the length of the substring (**m**). In the worst case, if the substring occurs at every position in the string, the **find** and **erase** operations will take O(n * m) time. However, if the substring occurs sparsely or only once, the time complexity will be lower.

Space Complexity: The space complexity of the code is O(1) as it does not use any extra data structures. The modifications are done in-place on the original string.