Determine if there is Redundant Brackets CodeStudio

Given an expression string containing parentheses and arithmetic operators, write a function to determine whether the expression contains any redundant brackets. Redundant brackets are those that do not contribute to the evaluation of the expression and can be removed without affecting the expression's value.

Example

Consider the following expression strings:

1. Expression: "(a+b+(a*b))"
    1. Output: There are no redundant brackets.
2. Expression: "(a+c*b)+(c))"
    1. Output: There is a redundant bracket.

Approach 1: Find redundant brackets using stack

In this approach, a stack is used to keep track of opening brackets and operators. For each character in the expression, if it's an opening bracket or an operator, it's pushed onto the stack. When a closing bracket is encountered, we check for operators between the opening and closing brackets. If no operators are found, the brackets are considered redundant.

Steps:

1. Initialize an empty stack.
2. For each character in the expression:
    - If the character is an opening bracket or an operator, push it onto the stack.
    - If the character is a closing bracket:
        - Pop elements from the stack until an opening bracket is encountered.
        - If no operators are found during the process, the brackets are redundant.

Time Complexity:

- The approach involves iterating through each character of the expression once.
- The pop operation may result in multiple characters being popped, but overall, the time complexity is linear.
- **Time Complexity**: **O(n), where n is the length of the expression.**

Space Complexity:

- The space complexity is determined by the stack's maximum size, which depends on the depth of nested brackets.
- In the worst case, the stack could hold all opening brackets and operators.
- **Space Complexity**: **O(n), where n is the length of the expression.**