Parsing A Boolean Expression

solved by	Senan
Platform	LeetCode
difficulty	Hard
Serial	1106
tags	Stack String Manipulation
language	C++
solved on	@20/10/2024
link	https://leetcode.com/problems/parsing-a-boolean-expression/
Completion	✓

Intuition

The problem involves parsing and evaluating a boolean expression, which can contain the operations `'!'` (NOT), `'&'` (AND), and `'|'` (OR). The key idea is to use a stack to simulate the evaluation of the expression step-by-step as we parse it character by character, handling the operations and values in a nested manner based on the structure of the input.

Approach

1. **Stack for parsing**: Use a stack to store the characters in the expression until we hit a closing bracket `')'`. At that point, we evaluate the expression inside the parentheses based on the operator (`'!'`, `'&'`, `'|'`) that precedes the opening bracket.
2. **Processing logic**:
    - If we encounter a boolean value (`'t'` or `'f'`), we push it to the stack.
    - If we encounter an operator (`'!'`, `'&'`, `'|'`), we also push it to the stack.
    - When encountering a closing parenthesis `')'`, we pop the stack until we encounter the operator at the start of the current sub-expression. Based on the operator, we evaluate the boolean values popped from the stack.
    - For `'!'`, we negate the single boolean value.
    - For `'&'`, we perform an AND operation over all boolean values.
    - For `'|'`, we perform an OR operation over all boolean values.
3. Finally, the top of the stack will contain the result of the expression, which we return.

Complexity

Time Complexity:

- **O(n)**, where `n` is the length of the input string. We traverse the entire input string once and process each character either by pushing it onto the stack or by popping values off the stack.

Space Complexity:

- **O(n)**, for the space used by the stack, where `n` is the length of the input string in the worst case (when the stack holds all characters temporarily).

Code

```
class Solution {
    bool isOperation(char ch) {
        return ch == '&' || ch == '!' || ch == '|';
    }
public:
    bool parseBoolExpr(string expression) {
        stack<char> st;

        for (char ch : expression) {
            if (ch == ',' || ch == '(') continue;
            else if (isOperation(ch) || ch == 't' || ch == 'f') {
                st.push(ch);
            }
            else { // closing bracket ')'
                stack<char> st2;
                bool hasTrue = false;
                bool hasFalse = false;


                while (!isOperation(st.top())) {
                    hasTrue |= st.top() == 't';
                    hasFalse |= st.top() == 'f'
                    st2.push(st.top());
                    st.pop();
                }

                char operation = st.top();
                st.pop();

                if (operation == '!') {
                    st.push(hasTrue ? 'f' : 't');
                }
                else if (operation == '&') {
                    st.push(hasFalse ? 'f' : 't');
                }
                else {
                    st.push(hasTrue ? 't' : 'f');
                }
            }
        }

        return (st.top() == 't') ? true : false;
    }
};
```