1106. Parsing A Boolean Expression

leetcode editorial

Recursive

Intuition

We check character by character. When we come across a boolean value (`t` or `f`), we can immediately return it as the result. However, when we see an operator like `!`, `&`, or `|`, we know it controls what comes inside the parentheses following it. We skip the opening parenthesis and move into the subexpression.

For the `!` operator, we expect one boolean value. We simply negate this value and return the opposite. For `&`, we know all values inside must be true for the result to be true, so we evaluate each one, stopping if we find an `f`. The `|` operator works similarly, but we stop as soon as we find a `t`.

Take the expression `&(t, |(f, t))` as an example. We first encounter `&`, which tells us we need to evaluate everything inside the parentheses. We then encounter `|`, which tells us to evaluate its inner subexpression. When we find that one of the values is `t`, we return `t` for the `|` part. Now the expression simplifies to `&(t, t)`, which evaluates to `t`.

Here we don’t repeat work or manipulate the string like in the previous approach, making it a little more efficient.

Algorithm

- Initialize `index` to `0` and call the `evaluate` function with the current expression and index.
- In the `evaluate` function:
    - Read the current character from `expression` at `index`, and increment `index` by 1.
    - Base cases:
        - If the character is 't' (true), return `true`.
        - If the character is 'f' (false), return `false`.
    - Handle the NOT operation ('!(...)'):
        - If the character is '!', increment `index` to skip the '('.
        - Recursively evaluate the inner expression and negate the result (using `!`), then increment `index` to skip the ')'.
        - Return the negated result.
    - Handle the AND ('&(...)') and OR ('|(...)') operations:
        - Initialize an array `values` to store the results of subexpressions.
        - Increment `index` to skip the '('.
        - While the current character is not ')':
            - If the character is not a comma, recursively evaluate the subexpression and add the result to `values`.
            - If the character is a comma, increment `index` to skip it.
        - After exiting the loop, increment `index` to skip the ')'.
    - Manual AND operation:
        - If the character is '&', iterate through `values`.
            - If any value is `false`, return `false`.
        - If all values are `true`, return `true`.
    - Manual OR operation:
        - If the character is '|', iterate through `values`.
            - If any value is `true`, return `true`.
        - If all values are `false`, return `false`.
    - Return `false` at the end of the function (this point should never be reached).