Problem 1106: Parsing A Boolean Expression

Problem Information

Difficulty: Hard
Acceptance Rate: 69.78%
Paid Only: No
Tags: String, Stack, Recursion

Problem Description

A **boolean expression** is an expression that evaluates to either `true` or `false`. It can be in one of the following shapes:

* `'t'` that evaluates to `true`. * `'f'` that evaluates to `false`. * `'!(subExpr)'` that evaluates to **the logical NOT** of the inner expression `subExpr`. * `'&(subExpr1, subExpr2, ..., subExprn)'` that evaluates to **the logical AND** of the inner expressions `subExpr1, subExpr2, ..., subExprn` where `n >= 1`. * `'|(subExpr1, subExpr2, ..., subExprn)'` that evaluates to **the logical OR** of the inner expressions `subExpr1, subExpr2, ..., subExprn` where `n >= 1`.

Given a string `expression` that represents a **boolean expression** , return _the evaluation of that expression_.

It is **guaranteed** that the given expression is valid and follows the given rules.

Example 1:

Input: expression = "&(|(f))" **Output:** false **Explanation:** First, evaluate |(f) --> f. The expression is now "&(f)". Then, evaluate &(f) --> f. The expression is now "f". Finally, return false.

Example 2:

Input: expression = "|(f,f,f,t)" **Output:** true **Explanation:** The evaluation of (false OR false OR false OR true) is true.

Example 3:

Input: expression = "!(&(f,t))" **Output:** true **Explanation:** First, evaluate &(f,t) --> (false AND true) --> false --> f. The expression is now "!(f)". Then, evaluate !(f) --> NOT false --> true. We return true.

Constraints:

* `1 <= expression.length <= 2 * 104` * expression[i] is one following characters: `'('`, `')'`, `'&'`, `'|'`, `'!'`, `'t'`, `'f'`, and `','`.

Code Snippets

C++:

```
class Solution {
public:
bool parseBoolExpr(string expression) {

}
};
```

Java:

```
class Solution {
public boolean parseBoolExpr(String expression) {

}
}
```

Python3:

```
class Solution:
def parseBoolExpr(self, expression: str) -> bool:
```