ALGORITHM OF AI ASSIGNEMNT -2

Algorithm is divided in the following major part.

1. **Checking if the formula given is a Well formed formula or not.**
2. **Cleaning the input provided by User**
3. **Converting Infix version of user input to post-fix**
4. **Creating expression tree.**
5. **Adding parenthesis by traversing the tree.**

Details of Each Part of Algorithm

Checking if the formula given is a Well formed formula or not :

function name : _check_if_well_formed_

- **a.** Firstly check if there is any special char in the expression entered by user.
- **b.** If expression is of length 1 & first char is operator then it is not WFF
- **c.** If last char in expression is operator then also it is not WFF
- **d.** Iterate through the expression if found any of these condition return Not WFF
  - **i.** Two consecutive chars are alphabet
  - **ii.** If they are in form A>B>C
  - **iii.** If it is of form A!B
  - **iv.** Two consecutive chars are operators.

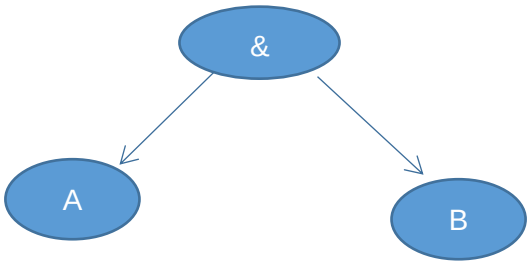
Class TreeNode:

This is class for creating nodes of tree for storing the operands & operators

Class : ExpressionTree

This is the main class which basically take care of cleaning input, converting user input from in-fix to post-fix, add post -fix expression in tree & then add parenthesis to the expression.

Function Details



Cleaning the input provided by User

Function Name : _clean_user_input

A. It removes all the white chars

B. It process the expression for “!” case

Converting Infix version of user input to post-fix

Function Name : convert_infix_postfix

0. Variable: precedence order and associativity helps to determine expression is needs to be calculated first

1. Create a empty stack

2. Iterate through each char of expression.
3. check if char is operator
4. check if the stack is empty or the top element is '(' if so just push the operator into stack
5. otherwise compare the curr char with top of the element
6. peek the top element
7. check for precedence
8. if they have equal precedence then check for associativity
9. pop the top of the stack and add to the postfix variable
10. if associativity of char is Right to left if so push the new operator to the stack
11. If precedence of chat is bigger than top of element in stack then push the char into stack else pop the top of the element.
12. If “(“ found then add it to the stack
13. Else if “)” then it pop the top element and add it to the postfix variable and update top of the stack also we remove parenthesis & discard it.
14. If char is operand, simply append the postfix variable.
15. If stack has element remove & add it to the postfix variable
16. Return the postfix variable.

Creating expression tree. (Insert the postfix expression into the tree using stack of nodes.)

Function Name : _insert_in_tree

1. if max size is 0, then it is infinite
2. create a node for the first element of the expression
3. iterator for expression
4. If char is alphabet then create a nod and push into the stack of nodes.
5. Else - create a root node for operands
6. Pop the last pushed item and create right_child
7. Pop item one before the last item and create left_child
8. Assign those as a child of the (parent)operator
9. Push back the operator node(subtree) to the stack
10. check if we reach last element in the expression so we can define the root of the tree by popping the element from stack.

Adding parenthesis by traversing the tree. (It simply takes the tree and add the parenthesis)

Function name : _add_parenthesis_exp

1. It fist check if tree is not empty if yes return none.
2. If it is operand it does not add parenthesis but if it is operand with negation then it adds.
3. Recursively traverse though the tree and add parenthesis on expression created using left part, right & operand which is at root of each sub tree

Note : I have refereed internet for algorithm like how to convert infix to postfix & then how to create expression tree. There are helper function which I have not mentioned here as they were used just for debugging.