IE260 Homework #2 Report

20150359 Chiyoon Park

1. Description

- BulidTree part

```
def buildTree(self,tokens):
    for itr in range(len(tokens)):
        token = tokens[itr]
        if self.isDigit(token) == True:
            tokenNode = Node(token, -1, True, False)
            self.stackOperand.push(tokenNode)
        elif token == "-" and ( self.stackOperator.getLength() == 0 or self.stackOperator.top().getValue() == '('):
            tokenOperator = Node(token, self.unaryPriority, False, False)
            self.stackOperator.push(tokenOperator)
        elif token == "ln":
            tokenOperator = Node(token, self.unaryPriority, False, False)
            self.stackOperator.push(tokenOperator)
        elif token == ')':
            while self.stackOperator.top().getValue() != '(':
                self.mergeLastTwoOperand()
            self.stackOperator.pop()
        else:
            tokenNode = Node(token, self.priorities[token], False, False)
            if tokenNode.getValue() == '(':
                self.stackOperator.push(tokenNode)
            else:
                while (self.stackOperator.getLength() != 0) and (self.stackOperator.top().getPriority() >= tokenNode.getPriority()):
                    self.mergeLastTwoOperand()
                self.stackOperator.push(tokenNode)
        print("Step. " + str(itr + 1))
        print("Stack Operator : " + str(self.stackOperator))
        print("Stack Operand : " + str(self.stackOperand))
    while self.stackOperator.top() != None:
        self.mergeLastTwoOperand()
    return self.stackOperand.top()
```

트리를 만드는 함수입니다. If 문을 통해서 토큰의 종류를 총 다섯 가지로 분류하는데, 먼저 숫자가 들어오는 경우 Node 클래스를 통해 새로운 노드를 생성한 후 바로 operand 스택에 push합니다. 다음으로 unaryoperator가 들어오는 경우입니다. StackOperator가 비어있는 상태에서 들어오는 "-"의 경우 unaryoperator이므로, 이때는 priority가 4인 노드를 만들어 stackoperator에 push합니다. Ln인 경우 역시 동일하게 진행합니다. 토큰이 ')'인 경우는 괄호 안의 계산을 우선적으로 해야 하는 상황입니다. 이 경우 '(' 괄호가 나올 때까지 operand와 operator를 결합하는 mergeLas tTwoOperand() 메소드를 실행합니다. 그리고 '(' 괄호를 제거합니다. 마지막으로 ')'가 아닌다른 연산자가 들어오는 경우, 만약 '(' 괄호라면 무조건 operand 스택에 push합니다. 그렇지 않은 경우 토큰의 우선 순위가 현재 operator 스택의 top보다 낮거나 같다면, 우선순위가 더 낮은 operator가 나올 때까지 계속 mergeLastTwoOperand () 메소드를 실행합니다. 여기서 operator 스택에 아무 원소가 없는 경우 에러가 날 수 있으므로, 이 while문의 제어문을 통해서 이런 경우를 제거했습니다. (이를 위해서 Stack 클래스에 getLength() 메소드를 추가했습니다)

- mergeLastTwoOperand() part

```
def mergeLastTwoOperand(self):
    nodeOperator = self.stackOperator.pop()
    if nodeOperator.getPriority() == self.unaryPriority:
        operand = self.stackOperand.pop()
        nodeOperator.setLeft(operand)
        self.stackOperand.push(nodeOperator)
    else:
        operand2 = self.stackOperand.pop()
        operand1 = self.stackOperand.pop()
        nodeOperator.setLeft(operand1)
        nodeOperator.setRight(operand2)
        self.stackOperand.push(nodeOperator)
```

Operand 스택에서 두 개를 pop하여 operator와 결합하여 다시 operand 스택에 집어넣는 메소드입니다. Unary 연산자인 경우 그 뒤에 들어온 숫자와 바로 연산을 해줘야 하고, 연산해야 할 operand가 하나이므로 연산자의 왼쪽 자식 노드에 operand를 넣어주었습니다. 다른 연산자가 들어오는 경우 operand 스택에서 두 원소를 pop하여 opearator 노드의 좌, 우 자식 노드로 설정하였습니다. 그리고 다시 operand 스택에 push합니다.

- evaluate part

```
def evaluate(self, node=None):
    if node == None:
        node = self.root
    if node.getLeft() != None:
        valueLeft = self.evaluate(node.getLeft())
    if node.getRight() != None:
        valueRight = self.evaluate(node.getRight())
    if node.getLeft() != None and node.getRight() == None:
        operator = node.getValue()
        if operator == '-':
            return -1 * float(valueLeft)
        elif operator == 'ln':
            return np.log(float(valueLeft))
    elif node.getLeft() != None and node.getRight() != None:
        operator = node.getValue()
        if operator == '+':
            return float(valueLeft) + float(valueRight)
        elif operator == '-':
            return float(valueLeft) - float(valueRight)
        elif operator == '*':
            return float(valueLeft) * float(valueRight)
        elif operator == '/':
            return float(valueLeft) / float(valueRight)
        elif operator == '^':
            return float(valueLeft) ** float(valueRight)
    else:
        return float(node.getValue())
```

Recursion을 통해 tree의 계산 결과를 구하는 메소드를 구현하였습니다. 왼쪽 노드만 있는 경우 unary 연산자이므로, 왼쪽 자식노드의 값을 연산자와 연산하여 반환하게 했습니다. 양 쪽 자식 노드가 모두 있는 경우, operator의 종류에 따라 좌 우 노드와 연산을 하게 만들었습니다. 그리고 이 연산 과정에서 valueLeft와 valueRight 값을 호출하여 evaluate를 통해서 계산하게 만들어서 재귀적으로 값을 구합니다.

- travers part

```
def traversPreFix(self, node=None):
    if node == None:
        node = self.root
    ret = ""
    if node.getLeft() == None and node.getRight() == None:
        ret = ret + node.getValue()
    elif node.getLeft() != None and node.getRight() == None:
        ret = ret + node.getValue() + self.traversPreFix(node.left)
    elif node.getLeft() == None and node.getRight() != None:
        ret = ret + node.getValue() + self.traversPreFix(node.right)
    elif node.getLeft() != None and node.getRight() != None:
        ret = ret + node.getValue() + self.traversPreFix(node.left) + self.traversPreFix(node.right)
    return ret

def traversPostFix(self, node=None):
    if node == None:
        node = self.root
    ret = ""
    if node.getLeft() == None and node.getRight() == None:
        ret = ret + node.getValue()
    elif node.getLeft() != None and node.getRight() == None:
        ret = ret + self.traversPostFix(node.left) + node.getValue()
    elif node.getLeft() == None and node.getRight() != None:
        ret = ret + self.traversPostFix(node.right) + node.getValue()
    elif node.getLeft() != None and node.getRight() != None:
        ret = ret + self.traversPostFix(node.left) + self.traversPostFix(node.right) + node.getValue()
    return ret
```

Recursion을 통해 tree를 pre-order와 post-order로 탐색하는 메소드입니다. traversPreFix의 경우 tree를 pre-order 탐색을 통해서 얻어진 결과를 반환하는 메소드입니다. 자식 노드의 유무를 통해서 총 4가지 케이스를 분류하였으며, 자식 노드가 있는 경우 자식 노드의 pre-order 탐색 결과를 호출하게 하여 재귀적으로 탐색하도록 만들었습니다. 그리고 ret에 값을 더하는 순서를 보면 node – left – Right 순서입니다. traversPostFix는 tree를 post-order 탐색을 통해서 얻어진 결과를 반환하는 메소드입니다. 위의 메소드와는 반대로 ret에 값을 더하는 순서를 보면 left – Right – node 순서입니다. 이를 통해서 재귀적으로 pre-order 탐색을 진행합니다. 그리고 최종적으로 만들어진 문자열 ret을 반환합니다.

2. Result

Input : 3 + 5 – 2 * 7

```
Step. 1
Stack Operator :
Stack Operand : 3,
Step. 2
Stack Operator : +,
Stack Operand : 3,
Step. 3
Stack Operator : +,
Stack Operand : 3,5,
Step. 4
Stack Operator : -,
Stack Operand : +,
Step. 5
Stack Operator : -,
Stack Operand : +,2,
Step. 6
Stack Operator : -,*,
Stack Operand : +,2,
Step. 7
Stack Operator : -,*,
Stack Operand : +,2,7,
Tokens : ['3', '+', '5', '-', '2', '*', '7']
Prefix : -+35*27
Postfix : 35+27*-
Evauate : -6.0
Tree :
-
....+
........3
........5
....*
........2
........7
```

Input : 3 + (5 – 2) * 7

```
Step. 1
Stack Operator :
Stack Operand : 3,
Step. 2
Stack Operator : +,
Stack Operand : 3,
Step. 3
Stack Operator : +,(,
Stack Operand : 3,
Step. 4
Stack Operator : +,(,
Stack Operand : 3,5,
Step. 5
Stack Operator : +,(,-,
Stack Operand : 3,5,
Step. 6
Stack Operator : +,(,-,
Stack Operand : 3,5,2,
Step. 7
Stack Operator : +,
Stack Operand : 3,-,
Step. 8
Stack Operator : +,*,
Stack Operand : 3,-,
Step. 9
Stack Operator : +,*,
Stack Operand : 3,-,7,
Tokens : ['3', '+', '(', '5', '-', '2', ')', '*', '7']
Prefix : +3*-527
Postfix : 352-7*+
Evauate : 24.0
Tree :
+
....3
....*
........-
............5
............2
........7
```

Input : - 3 + (ln 5 – 2) * 7

```
Step. 1
Stack Operator : -,
Stack Operand :
Step. 2
Stack Operator : -,
Stack Operand : 3,
Step. 3
Stack Operator : +,
Stack Operand : -,
Step. 4
Stack Operator : +,(,
Stack Operand : -,
Step. 5
Stack Operator : +,(,-,
Stack Operand : -,
Step. 6
Stack Operator : +,(,-,ln,
Stack Operand : -,
Step. 7
Stack Operator : +,(,-,ln,
Stack Operand : -,5,
Step. 8
Stack Operator : +,(,-,
Stack Operand : -,-,
Step. 9
Stack Operator : +,(,-,
Stack Operand : -,-,2,
Step. 10
Stack Operator : +,
Stack Operand : -,-,
Step. 11
Stack Operator : +,*,
Stack Operand : -,-,
Step. 12
Stack Operator : +,*,
Stack Operand : -,-,7,
Tokens : ['-', '3', '+', '(', '-', 'ln', '5', '-', '2', ')', '*', '7']
Prefix : +-3*--ln527
Postfix : 3-5ln-2-7*+
Evaluate : -28.266065387038704
Tree :
+
....-
........3
....*
........-
............-
................ln
....................5
............2
........7
```

Input : 3 + (7 – 2) ^ 4 – 3 * 8 + 22

```
Enter formula : 3 + ( 7 - 2 ) ^ 4 - 3 * 8 + 22
Step. 1
Stack Operator :
Stack Operand : 3,
Step. 2
Stack Operator : +,
Stack Operand : 3,
Step. 3
Stack Operator : +,(,
Stack Operand : 3,
Step. 4
Stack Operator : +,(,
Stack Operand : 3,7,
Step. 5
Stack Operator : +,(,-,
Stack Operand : 3,7,
Step. 6
Stack Operator : +,(,-,
Stack Operand : 3,7,2,
Step. 7
Stack Operator : +,
Stack Operand : 3,-,
Step. 8
Stack Operator : +,^,
Stack Operand : 3,-,
Step. 9
Stack Operator : +,^,
Stack Operand : 3,-,4,
Step. 10
Stack Operator : -,
Stack Operand : +,
Step. 11
Stack Operator : -,
Stack Operand : +,3,
Step. 12
Stack Operator : -,+,
Stack Operand : +,3,
Step. 13
Stack Operator : -,+,
Stack Operand : +,3,8,
Step. 14
Stack Operator : +,
Stack Operand : -,
Step. 15
Stack Operator : +,
Stack Operand : -,22,
Tokens : ['3', '+', '(', '7', '-', '2', ')', '^', '4', '-', '3', '*', '8', '+', '22']
Prefix : +-+3^-724+3822
Postfix : 372-4^+38*-22+
Evaluate : 626.0
Tree :
+
....-
........+
............3
............^
..............-
..................7
..................2
..............4
........*
............3
............8
....22
```