Python O(n) Solution using recursion

heroic ★ 71 Last Edit: October 24, 2018 5:23 AM 2.7K VIEWS

40

Similar to Basic Calculator II and I use recursion to handle bracket.

```
class Solution:
    def calculate(self, s):
        """
        :type s: str
        :rtype: int
        """
        s = s + "$"
        def helper(stack, i):
            num = 0
            sign = '+'
            while i < len(s):
                c = s[i]
                if c == " ":
                    i += 1
                    continue
                if c.isdigit():
                    num = 10 * num + int(c)
                    i += 1
                elif c == '(':
                    num, i = helper([], i+1)
                else:
                    if sign == '+':
                        stack.append(num)
                    if sign == '-':
                        stack.append(-num)
                    if sign == '*':
                        stack.append(stack.pop() * num)
                    if sign == '/':
                        stack.append(int(stack.pop() / num))
                    num = 0
                    i += 1
                    if c == ')':
                        return sum(stack), i
                    sign = c
            return sum(stack)
        return helper([], 0)
        ...
```

Comments: 7 Best Most Votes Newest to Oldest Oldest to Newest

Type comment here... (Markdown is supported)

Post

Kushina ★ 31 Last Edit: August 23, 2019 12:20 AM

Similar Python3 solution but more understandable, I think. `getOpd` gets the next number or evaluation of expression within brackets, whereas `evalExpr` actually evaluates it. Both functions return the index of the next unprocessed character in the string, in addition to the result. Beats 98%.

```
    def calculate(self, s: str) -> int:
        s = ''.join(s.split())
        L = len(s)

        def getOpd(i):
            if s[i] == '(':
                return evalExpr(i+1, 1)
```

Read More

2 Reply

benxiaoliu ★ 2 December 8, 2019 1:26 AM