Daily Coding Problem Blog

Daily Coding Problem #274

Problem

This problem was asked by Facebook.

Given a string consisting of parentheses, single digits, and positive and negative signs, convert the string into a mathematical expression to obtain the answer.

Don't use `eval` or a similar built-in parser.

For example, given '-1 + (2 + 3)', you should return 4.

Solution

One approach to solving this is to use two stacks, one for operators and one for digits. The idea is that each positive or negative sign will correspond to two digits, so every time we come across a part of the expression that looks like a+b, we can pop off one value from the operation stack and two values from the digit stack, compute the result, and add the result back onto the digit stack. This will look like the following:

```
def apply_ops(ops, values):
    op = ops.pop()
    right, left = values.pop(), values.pop()
    if op == '+':
        values.append(left + right)
    else:
        values.append(left - right)
```

As we move through our expression, if we encounter either digits or signs, we append them to the appropriate stack. We will also add parentheses to the operations stack, with the condition that as soon as we see a close parenthesis, we keep performing `apply_ops` until we can pop off the corresponding open parenthesis.

Finally, if we reach the end of the expression and there are still operations left on the stack, we run `apply_ops` repeatedly until all values have been computed.

```
def parse(equation):
    values = []
    ops = []

    for char in equation:

        if char.isdigit():
            values.append(int(char))
        elif char == '(':
            ops.append(char)
        elif char == ')':
            while ops and ops[-1] != '(':
                apply_ops(ops, values)
            ops.pop()
        elif char == ' ':
            continue
        else:
            while ops and ops[-1] not in "()":
                apply_ops(ops, values)
            ops.append(char)

    while ops:
        apply_ops(ops, values)

    return values[0]
```

One final consideration is that we have not handled the case where the expression starts with a negative sign, or the first digit inside a set of parentheses is negative. There are a few ways to fix this, but the simplest may be to add a zero in front of the negative sign for each of these cases.

```
def fix_negatives(equation):
    if equation[0] == '-':
        equation = '0' + equation

    equation = equation.replace('(-', '(0-')

    return equation
```

We would run this function first, before stepping through our expression.

This solution will use an additional `O(N)` space, where `N` is the length of our input, for the operations and digits stacks. Since we are appending and popping elements from these stacks no more than `N` times, our `parse` function will take `O(N)` time.

Python's `replace` function runs in `O(N)` time, making `fix_negatives` also linear in time complexity. Therefore, the whole algorithm will run in `O(N)` time and space.