Back

Short Python solutions, one O(n)

StefanPochmann ★ 47270 Last Edit: August 25, 2018 11:35 AM 2.0K VIEWS

10

O(n) with stack

Collect chars from back to front on a stack, evaluate ternary sub expressions as soon as possible:

```
def parseTernary(self, expression):
    stack = []
    for c in reversed(expression):
        stack.append(c)
        if stack[-2:-1] == ['?']:
            stack[-5:] = stack[-3 if stack[-1] == 'T' else -5]
    return stack[0]
```

Originally my check was `stack[-4::2] == [':', '?']`, but @YiL1228's is right, looking for `?` is enough.

$O(n^2)$, several versions

Always evaluate/replace the last included ternary. So somewhat the same as the stack solution but only $O(n^2)$. Didn't think of that and instead went right for the stack solution, but now that I saw it from others, I just had to write a few ways myself :)

Version just working on the string:

```
def parseTernary(self, s):
    while len(s) > 1:
        i = s.rfind('?') - 1
        s = s[:i] + s[i+2 if s[i] == 'T' else i+4] + s[i+5:]
    return s
```

Version working on a list version of the string because it can be modified (need to reverse it because lists can only tell the first occurrence, not the last):

```
def parseTernary(self, s):
    a = list(s)[::-1]
    while len(a) > 1:
        i = a.index('?') - 3
        a[i:i+5] = a[i+2 if a[i+4] == 'T' else i]
    return a[0]
```

Version using a regular expression (reversing the string because `sub` doesn't support replacing the last occurrence but does support replacing the first):

```
def parseTernary(self, s):
    s = s[::-1]
    while len(s) > 1:
        s = re.sub('(.):(.)\?(.)', lambda m: m.group(1 + (m.group(3) == 'T')), s, 1)
    return s
```