Ternary Expression Parser

Given a string representing arbitrarily nested ternary expressions, calculate the result of the expression. You can always assume that the given expression is valid and only consists of digits `0-9`, `?`, `:`, `T` and `F` (`T` and `F` represent True and False respectively).

Note:

1. The length of the given string is ≤ 10000.
2. Each number will contain only one digit.
3. The conditional expressions group right-to-left (as usual in most languages).
4. The condition will always be either `T` or `F`. That is, the condition will never be a digit.
5. The result of the expression will always evaluate to either a digit `0-9`, `T` or `F`.

Example 1:

```
Input: "T?2:3"

Output: "2"

Explanation: If true, then result is 2; otherwise result is 3.
```

Example 2:

```
Input: "F?1:T?4:5"

Output: "4"

Explanation: The conditional expressions group right-to-left. Using parenthesis, it
 is read/evaluated as:

             "(F ? 1 : (T ? 4 : 5))"                  "(F ? 1 : (T ? 4 : 5))"
          -> "(F ? 1 : 4)"                 or       -> "(T ? 4 : 5)"
          -> "4"                                    -> "4"
```

Example 3:

```
Input: "T?T?F:5:3"

Output: "F"

Explanation: The conditional expressions group right-to-left. Using parenthesis, it
 is read/evaluated as:

             "(T ? (T ? F : 5) : 3)"                  "(T ? (T ? F : 5) : 3)"
          -> "(T ? F : 3)"                 or       -> "(T ? F : 5)"
          -> "F"                                    -> "F"
```

Solution 1

Iterate the expression from tail, whenever encounter a character before '?', calculate the right value and push back to stack.

P.S. this code is guaranteed only if "the given expression is valid" base on the requirement.

```
public String parseTernary(String expression) {
    if (expression == null || expression.length() == 0) return "";
    Deque<Character> stack = new LinkedList<>();

    for (int i = expression.length() - 1; i >= 0; i--) {
        char c = expression.charAt(i);
        if (!stack.isEmpty() && stack.peek() == '?') {

            stack.pop(); //pop '?'
            char first = stack.pop();
            stack.pop(); //pop ':'
            char second = stack.pop();

            if (c == 'T') stack.push(first);
            else stack.push(second);
        } else {
            stack.push(c);
        }
    }

    return String.valueOf(stack.peek());
}
```

written by NathanNi original link here

Solution 2

In order to pick out useful "?" and ":", we can always begin with**the last "?" and the first ":" after the chosen "?"**.

Therefore, directly seek for the last "?" (or you can simply put all "?" into a stack) and update the string depending on T or F until no more "?"s.

e.g.
"(F ? 1 : (T ? 4 : 5))" => "(F ? 1 : 4)" => "4"
"(T ? (T ? F : 5) : 3)" => "(T ? F : 3)" => "F"

EDIT:
Removed stack, added Java version.

Python:

```
class Solution(object):
    def parseTernary(self, expression):
        """
        :type expression: str
        :rtype: str
        """
        while len(expression) != 1:
            i = expression.rindex("?")    # begin with the last '?'.
            tmp = expression[i+1] if expression[i-1] == 'T' else expression[i+3]
            expression = expression[:i-1] + tmp + expression[i+4:]
        return expression
```

Java (It costs 7-lines):

```
public class Solution {
    public String parseTernary(String expression) {
        while (expression.length() != 1) {
            int i = expression.lastIndexOf("?");    // get the last shown '?'
            char tmp;
            if (expression.charAt(i-1) == 'T') { tmp = expression.charAt(i+1); }
            else { tmp = expression.charAt(i+3); }
            expression = expression.substring(0, i-1) + tmp + expression.substrin
g(i+4);
        }
        return expression;
    }
}
```

written by YJL1228 original link here

Solution 3

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 @YJL1228's is right, looking for `?` is enough.

O(n²), several versions

Always evaluate/replace the last included ternary. So somewhat the same as the stack solution but only O(n²). 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
```

written by StefanPochmann original link here