Construct Binary Tree from String

You need to construct a binary tree from a string consisting of parenthesis and integers.

The whole input represents a binary tree. It contains an integer followed by zero, one or two pairs of parenthesis. The integer represents the root's value and a pair of parenthesis contains a child binary tree with the same structure.

You always start to construct the **left** child node of the parent first if it exists.

Example:

```
Input: "4(2(3)(1))(6(5))"
Output: return the tree root node representing the following tree:

       4
     /   \
    2     6
   / \   /
  3   1 5
```

Note:

1. There will only be `'('`, `')'`, `'-'` and `'0'` ~ `'9'` in the input string.
2. An empty tree is represented by `""` instead of `"()"`.

Solution 1

I change a string like `'4(2(3)(1))(6(5))'` to `'t(4,t(2,t(3),t(1)),t(6,t(5)))'` and then just let Python evaluate that (with the help of my TreeNode constructor).

```
def str2tree(self, s):
    def t(val, left=None, right=None):
        node, node.left, node.right = TreeNode(val), left, right
        return node
    return eval('t(' + s.replace('(', ',t(') + ')') if s else None
```

written by StefanPochmann original link here

Solution 2

This is a pretty common approach with static index variable.

```
let i = 0;

var construct = function (s) {
    let numStr = [];

    while (s[i] === '-' || s[i] >= '0' && s[i] <= '9')
        numStr.push(s[i++]);

    if (numStr.length === 0)
        return null;

    let node = new TreeNode(Number(numStr.join('')));
    if (s[i] === '(')
        i++, node.left = construct(s), i++;
    if (s[i] === '(')
        i++, node.right = construct(s), i++;
    return node;
}

var str2tree = function(s) {
    i = 0;
    return construct(s);
};
```

written by dettier original link here

Solution 3

We perform a recursive solution. There are four cases for what the string might look like:

1. empty
2. [integer]
3. [integer] ([tree])
4. [integer] ([tree]) ([tree])

When there is no '(', we are in one of the first two cases and proceed appropriately. Else, we find the index "jx" of the ')' character that marks the end of the first tree. We do this by keeping a tally of how many left brackets minus right brackets we've seen. When we've seen 0, we must be at the end of the first tree. The second tree is going to be the expression S[jx + 2: -1], which might be empty if we are in case #3.

```
def str2tree(self, S):
    ix = S.find('(')
    if ix < 0:
        return TreeNode(int(S)) if S else None

    bal = 0
    for jx, u in enumerate(S):
        if u == '(': bal += 1
        if u == ')': bal -= 1
        if jx > ix and bal == 0:
            break

    root = TreeNode(int(S[:ix]))
    root.left = self.str2tree(S[ix+1:jx])
    root.right = self.str2tree(S[jx+2:-1])
    return root
```

written by awice original link here