Mutable Trees

Aditya Iyer

Tree Structure

A tree has the following structure (at least the way we define it in 61A)

- A root node
- A list of branches (note that each branch itself is a tree. This is a very important idea to understand for exam problems, as you'll see a lot of recursion in tree problems)

Another way to look at trees:

- Each node has a label
- A node can be a parent/child of another

Tree Class (the way it is defined in 61A)

```
class Tree:

    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches

    def __repr__(self):
        if self.branches:
            branch_str = ', ' + repr(self.branches)
        else:
            branch_str = ''
        return 'Tree({0}{1})'.format(self.label, branch_str)

    def __str__(self):
        return '\n'.join(self.indented())

    def indented(self):
        lines = []
        for b in self.branches:
            for line in b.indented():
                lines.append('  ' + line)
        return [str(self.label)] + lines
```

Example of tree traversal

The goal if this question is to print the labels of the tree.

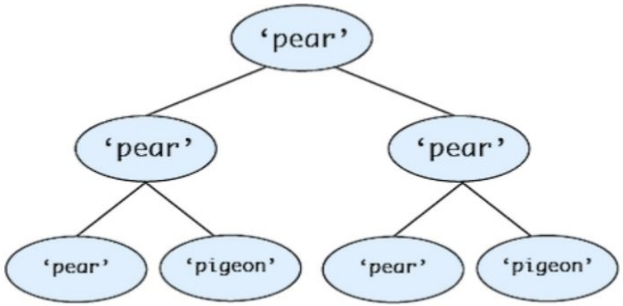
```
def traverse(t):
    print(t.label)
    for b in t.branches:
        traverse(b)
```

Trees Exam Level Question

8. (6.0 points) Pigeons in a Pear Tree

The function `pigeon_locations` accepts a single parameter `t`, an instance of the `Tree` class where all non-leaf nodes have two branches, the label of each node is either 'pear' or 'pigeon', and only leaf nodes can have the 'pigeon' label.

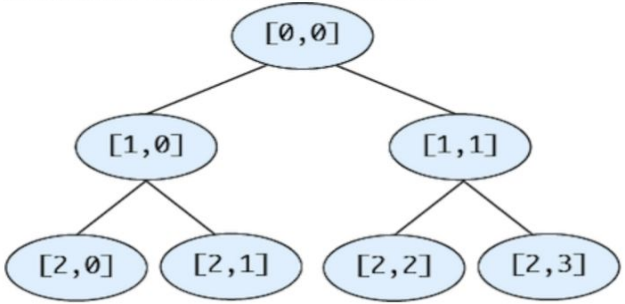
Here's a drawing of an example input tree:



The function returns a list of locations of pigeons in `t`, where each location is a two-element list with the "depth" as the first element and the "left" as the second element.

The "depth" of the root node is 0, and it increases by one at each level of the tree down from the root node. The left-most node of each level of the tree has a "left" of 0, and it increases from there.

Here's the same tree where each node is labeled with its location:



When `pigeon_locations` is called on the example tree, it returns `[[2, 1], [2, 3]]`, since the two pigeons are located at `[2, 1]` and `[2, 3]`.

Complete the `pigeon_locations` function below per the doctests and description.

```
def pigeon_locations(t):
    """
    Returns a list of location of pigeons in the tree T, where each location
    is a two-element list, with the depth as the first element
    and the left as the second element. The depth of the root node is 0 and
    increases from there. The left starts at 0 from the left-most branch of each tree.
    Every non-leaf node has two branches.

    >>> t1 = Tree('pear', [Tree('pigeon'), Tree('pear')])
    >>> print(t1)
    pear
      pigeon
      pear
    >>> pigeon_locations(t1)
    [[1, 0]]
    >>> t2 = Tree('pear', [Tree('pear', [Tree('pear'), Tree('pigeon')]),
    ...                    Tree('pear', [Tree('pigeon'), Tree('pear')])])
    >>> print(t2)
    pear
      pear
        pear
        pigeon
      pear
        pigeon
        pear
    >>> pigeon_locations(t2)
    [[2, 1], [2, 2]]
    >>> no_pigeons = Tree('pear', [Tree('pear'),
    ...                            Tree('pear', [Tree('pear'), Tree('pear')])])
    >>> pigeon_locations(no_pigeons)
    []
    """
    def helper(t, depth, left):
        if t.label == 'pigeon':
            return __________
                     (a)
        locations = [helper(_____, _____, _____)  for i in _____]
                                (b)    (c)    (d)              (e)
        return sum(locations, [])
    ----------
       (f)
```

Trees Exam Level Question Solution

(a) **(1.0 pt)** Fill in blank (a).

```
[[depth, left]]
```

(b) **(1.0 pt)** Fill in blank (b).

```
t.branches[i]
```

(c) **(1.0 pt)** Fill in blank (c).

```
depth + 1
```

(d) **(1.0 pt)** Fill in blank (d).

```
(left * 2 ) + i
```

(e) **(1.0 pt)** Fill in blank (e).

```
range(len(t.branches))
```

(f) **(1.0 pt)** Fill in blank (f).

```
return helper(t, 0, 0)
```