Computational Biology
Tufts University Bio 35
Fall 2021

Assignment 7: Phylogenetic trees

Computing with trees: Place your code in a file called `PhylogeneticTrees.py`. The problems below will make use of the following sample tree, which you can cut-and-paste this into your homework file.

example
```
Groodies =  ( "X",
                ("Y",
                    ("W", (), ()),
                    ("Z",
                        ("E", (), ()),
                        ("L", (), ())
                    )
                ),
                ( "C", (), () )
            )
```

.

This assignment is taken (with minor adaptations) from the companion material for CFB. You can find the original at `https://www.cs.hmc.edu/twiki/bin/view/CFB/PhylogeneticTrees`.

Problems

Using the techniques we've discussed for operating on trees, write the following functions:

1. `leafCount(Tree)`: A function that counts the leaf nodes of the tree, *i.e.*, the nodes that have no descendants.

   example
   ```
   >>> leafCount(Groodies)
   4
   >>> leafCount((5,(3,("A", (), ()),("B", (), ())),("C", (), ())))
   3
   ```

2. `find(node, Tree)`. This function returns `True` if the given node is in the given `Tree` and returns `False` otherwise. Some examples of `find`:

   example
   ```
   >>> find('W',Groodies)
   True
   >>> find('A',Groodies)
   False
   >>> find('E',Groodies)
   True
   ```

3. `subtree(node, Tree)` Returns the `subtree` of the given `Tree` rooted at the given node. Said another way, this function returns the tree beginning at node. This function can be very short, and may have as few as four lines in its body. It will be recursive, but can also call the find function for help. In particular, if the root of the `Tree` matches node then we just return that `Tree`. If not, we can use find to determine whether the species is in the left `subtree` or the right `subtree`, and then proceed from there. You can assume that node is present in `Tree`. Here are some examples:

example
```
>>> subtree("W", Groodies)
('W', (), ())
>>> subtree("Y", Groodies)
('Y', ('W', (), ()), ('Z', ('E', (), ()), ('L', (), ())))
>>> subtree("Z", Groodies)
('Z', ('E', (), ()), ('L', (), ()))
```

4. `nodeList(Tree)` : Takes a `Tree` as input and returns a list of all of the nodes in that tree (including both leaves and ancestral nodes). This function can be done in three lines of code, but you are welcome to make it longer if you like. Here is an example:

example
```
>>> nodeList(Groodies)
['X', 'Y', 'W', 'Z', 'E', 'L', 'C']
```

5. `descendantNodes(node, Tree)`: Returns the list of all descendant nodes of the given node in the `Tree`. This function will not be recursive itself, but it will call the `nodeList` and `subtree` functions for help. This function can be written in two lines of code (not counting the def line and the docstring).

   Here are some examples of `descendantNodes`:

example
```
 >>> descendantNodes("X", Groodies)
 ['Y', 'W', 'Z', 'E', 'L', 'C']
 >>> descendantNodes("Y", Groodies)
 ['W', 'Z', 'E', 'L']
 >>> descendantNodes("Z", Groodies)
 ['E', 'L']
```

6. `parent(node, Tree)`: returns the parent of the given node in the `Tree`. If the node has no parent in the tree, the function should return the special value `None`.

example
```
>>> parent("Y", Groodies)
'X'
>>> parent("W", Groodies)
'Y'
>>> parent("Z", Groodies)
'Y'
>>> parent("E", Groodies)
'Z'
>>> parent("X", Groodies)
>>> parent("X", Groodies) == None
True
```

7. `scale(Tree,scaleFactor)`: As we've discussed, internal nodes represent the most recent common ancestor of the species which descend from them. In the example tree above, we've labelled these nodes using letters, the same way we've labeled the leaf nodes. However sometimes biologists like to include other kinds of information at these nodes, for example, information on when that most recent common ancestor lived. Here is a tree where the internal nodes are labeled with numbers.

example

```
Tree = (4.3,
         ('C', (), () ),
         (3.2,
           ('A',(), ()),
           ('B',(),()))
        )
```

Perhaps we're measuring time in millions of years before present. In that case, this tree would tell us that species A and B had a most recent common ancestor which lived 3.2 million years ago, and that all three species (A, B and C) had a most recent common ancestor that lived 4.3 million years ago.

Your task now is to write a function scale which takes a `Tree` as input, and multiplies the numbers at its internal nodes by `scaleFactor` and returns a new tree with those values. Here are some examples:

example

```
>>> Tree = (4.3, ('C', (), () ),(3.2, ('A',(), ()), ('B',(),())) )
>>> scale(Tree,2.0)
(8.6, ('C', (), ()), (6.4, ('A', (), ()), ('B', (), ())))
```

We will later use this function to calibrate trees so we can better estimate dates in human evolution.