CS 120 (Spring 22): Introduction to Computer Programming II

Short Project #8 - Trees (Short)

due at 5pm, Thu 17 Mar 2022

REMEMBER: The `itertools, copy` and `collections` libraries in Python are **banned.**

1 Overview

In this project, you will be practicing some simple functions involving trees. Most of them will require that you write a function which recurses over a tree, and print or return something about it; one of the functions will require you to build a tree.

When you build the tree, make sure to use the `TreeNode` class, which I've provided to the class in the file `tree_node.py`. (For simplicity, we are going to limit ourselves to binary trees in this project. But don't assume that they are BSTs - we're just dealing with trees in general, so far.)

Put all of your functions into a file named `tree_funcs_short.py`.

1.1 Other Rules

- Every one of your functions must be recursive.
- In this project, helper functions (and default arguments) are banned.
- Your functions must always handle any size tree: large, small, single node, or even empty.

2 `tree_count(root)`

Write a function, `tree_count()`, which takes a tree as a parameter, and returns the number of nodes in the tree.

3 `tree_sum(root)`

Write a function, `tree_sum()`, which takes a tree as a parameter, and returns the sum of all of the values in the tree.

You may assume that all of the values are numeric; an empty tree should return zero.

4 `tree_depth(root)`

Write a function, `tree_depth()`, which takes a tree as a parameter, and returns the depth of the deepest leaf, anywhere in the tree.

The **"depth"** is defined as the distance, in links, from the root node to the leaf; if the tree has only a single node (the root itself), then this function should return 0. Weirdly, this means that, if the tree is empty, you should return -1. Odd, I know! That's just a weird quirk of how Russ defines his trees...

5 `tree_print(root)`

Write a function, `tree_print()`, which takes a tree as a parameter, and prints out the value stored in every node, one per line.

Note that this function does **not** impose a particular order on you! You may print the values in **any order!** My testcase-grading code will be smart enough to give you full points if you print all of the correct lines - no matter what order you use.

6 `tree_build_left_linked_list(data)`

This is a tree version of the `array_to_list` problem, except that it builds it using `TreeNode` objects. Use only the `left` links; all of the `right` links, in the entire tree, must be `None`.

The first value from the input data must become the root of the tree.

Remember: All of the functions in this project must be recursive!

7 Turning in Your Solution

You must turn in your code using GradeScope.

8 Acknowledgements

Thanks to Saumya Debray and Janalee O'Bagy for many resources that I used and adapted for this class.