CS 120 (Fall 21): Introduction to Computer Programming II

Long Project #9

due at 5pm, Tue 26 Oct 2021

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

1 Overview

In this project, you will be writing some more complex functions involving trees. Most of them will require that you write a function which recurses over a tree.

Some of the functions will assume that the tree is a BST; some will not. All of the trees will be binary trees, and will use the `TreeNode` class that I've provided in `tree_node.py`.

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

1.1 Common Rules

- Except where explicitly stated in the function descriptions below, you may choose whether to use recursion or a loop in the functions. (Note that the recursive solution will be much easier in many cases.)
- In this project, helper functions (and default arguments) are banned.
- In three of the functions in this project, you may assume that the tree we pass you is not empty. But **only** do this if we explicitly say so! Most of your functions must handle the empty-tree case.

2 `bst_search_loop(root, val)`

Search the tree to find the value, if it exists. If it exists, then return the **node** which contains it. If it does not exist, return `None`.

This function should assume that the tree is a BST.

RESTRICTION: Your implementation must use a loop; recursion is forbidden in this function.

3 `tree_search(root, val)`

Search the tree to find the value, if it exists. If it exists, then return the **node** which contains it. If it does not exist, return `None`.

This function should not assume anything about the order of values in the tree; it might **not** be a BST.

Feel free to use a loop or recursion, your choice.

4 bst_insert_loop(root, val)

Insert the value into the BST. You may assume that the value does **not** already exist in the tree.

Note that this function is explicitly **not** in the `x = change(x)` style. Instead, the following rules apply:

- You may assume that the tree will not be empty.
- Return nothing from this function.
- You **must** use a loop, recursion is forbidden.

Why all the limitations? Two reasons. First, I've already shown you (in lecture) the solution for an `x = change(x)` version of `insert()`. Second, I hope that you'll see that this version is harder - and you won't want to use it in the future!

5 pre_order_traversal_print(root) in_order_traversal_print(root) post_order_traversal_print(root)

Traverse the tree, printing out each of the values inside it (one per line). For each of the three functions, print it out in the order required.

Do **not** assume that the tree is a BST. Thus, the "in order" tree will not actually print out the values in order; rather, they will print it out in the order required by a n **in-order traversal.**

6 in_order_vals(root)

Return all of the values stored in the tree as an array. Like `in_order_traversal_print()` above, do not assume that the tree is a BST; instead, return the values in the order they would be encountered in an in-order traversal.

7 bst_max(root)

Return the maximum value in the tree. You **may** assume that the tree is a BST. Additionally, you may assume that the tree is not empty.

8 tree_max(root)

Return the maximum value in the tree. You **must not** assume that the tree is a BST. However, you may assume that the tree is not empty.

9 Turning in Your Solution

You must turn in your code using GradeScope.

10 Acknowledgements

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