CSC148 Summer 2018: Lab 7

Introduction

The goals of this lab are:

- To get you familiar with how Binary Trees work
- To give you practice traversing Binary Trees

Don't hesitate to make use of other resources for this lab, including the course notes, your TAs, instructor, or other students.

General Lab Notes

1. Make sure you have lab__pyta.txt downloaded and placed in the directory (or directories) where you'll be working.
2. To use PythonTA, include the following code (if you already have a main block, just add the body to the end of it):

```
if __name__ == '__main__':
    import python_ta
    python_ta.check_all(config="lab_pyta.txt")
```

Your lab__pyta.txt should be in the same folder as the .py files you're running. PythonTA will raise errors regarding style, specifying the lines you need to fix. You should get familiar with what the errors mean, and how to fix them: this will be important for your exercises and assignments.

Getting Started

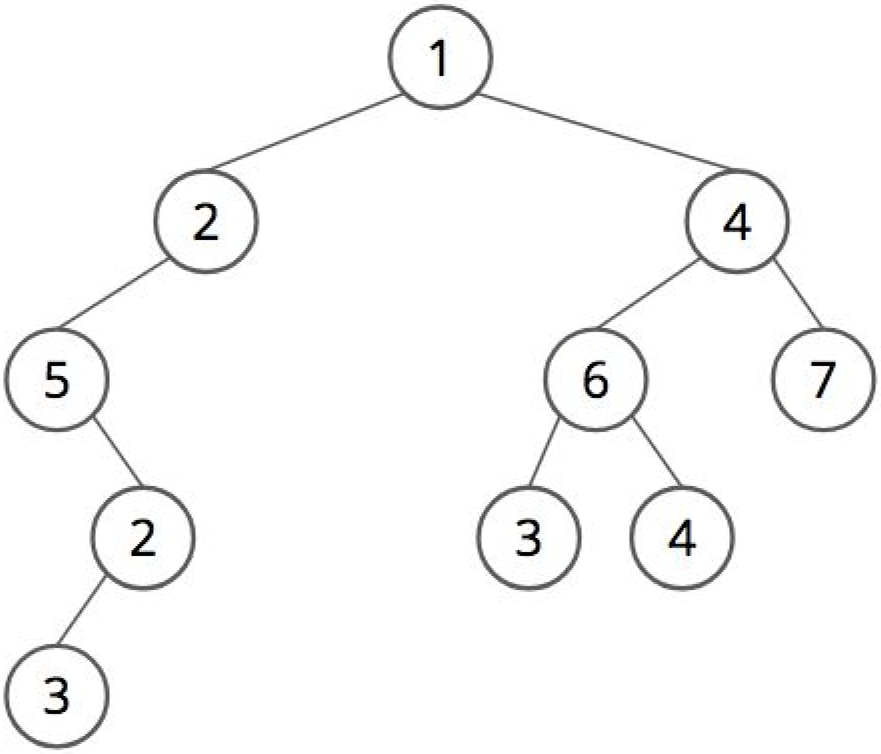
This lab will have you using BinaryTrees which only have very basic functionality. Download lab__binary__trees.py or write your own BinaryTree classes. This class contains only the ___init___ and a ___str___ function for BinaryTrees.

Note: Some of these questions were also present in the previous lab for Trees -- the main differences being: 1) We're working with BinaryTrees instead of Trees, and 2) We're writing **functions** instead of **methods**. The reason we're writing functions instead of methods is that, with Binary Trees, when we make recursive calls we can end up recursing on None. i.e. doing something like t.get__value() when t == None would result in an error, but calling get__value(t) when t == None isn't a problem. With general Trees, we'll never reach None since the list of children won't ever contain None in it.

Creating a BinaryTree

Within the `if __name__ == '__main__'` block, the BinaryTree t was created. Make sure you understand how it was created, what it looks like, and what the subtrees in it are.

Create a new tree called `large_tree` that looks like the following:



We'll use `large_tree` as the example for all of the functions to be implemented in this lab. You'll want to make additional trees to test these functions on.

Traversals

If we printed the nodes of `large_tree` using a pre-order traversal, what's the order of values to be printed?

What if we used a post-order traversal? Level-order? In-order?

Getting all of the values of the internal nodes (get_internal_values)

Recall that an internal node is any node that's not a leaf (i.e. it has at least 1 child).

Write a function called get_internal_values that takes in a BinaryTree (or None) and returns the values of all internal nodes of the BinaryTree in pre-order.

For example, calling `get_internal_values(large_tree)` should return `[1, 2, 5, 2, 4, 6]`.

If you wanted this to be in post-order, what would you have to change? What would the post-order version be? How about in-order?

Finding the maximum depth (get_max_depth)

Write a function called get_max_depth that takes in a BinaryTree (or None) and returns its maximum depth.

For example, calling `get_max_depth(large_tree)` should return `4`.

Finding the depth of a value (get_depth_of)

Write a function called get_depth_of that takes in a BinaryTree (or None) and a single value and returns the depth of the node with that value. This should return the depth of the first node found with that value during a pre-order traversal.

For example, calling `get_depth_of(large_tree, 3)` should return `2` and `get_depth_of(large_tree, 5)` should return `0`.

What would the post-order version be? If we called `get_depth_of(large_tree, 5)` and used a post-order traversal, what would be returned instead of `0`?

Finding all values at a certain depth (get_values_at_depth)

Write a function in called get_values_at_depth which, given a BinaryTree (or None) and a depth, returns all of the values in the BinaryTree at that depth.

For example, calling `get_values_at_depth(large_tree, 2)` should return `[5, 6, 7]`. If no such nodes are found, then return an empty list.

Hint: You'll want to adjust depth with each recursive call you make. If we're looking for a value that's has a depth of 3 away from the root, then how far should it be from the subtrees?

Create a copy (copy)

Write a function that takes in a BinaryTree (or None) and returns a copy of the BinaryTree -- if None is passed in, it just returns None. All changes to the copy shouldn't affect the original, and vice versa.

For example, if you did:

```
large_tree_copy = copy(large_tree)
large_tree_copy.value = 10
large_tree_copy.left.value = 3
print(large_tree.value)  # This should print 1
print(large_tree_copy.value) # This should print 10
print(large_tree.left.value) # This should print 2
print(large_tree_copy.left.value) # This should print 3
```

Then large_tree itself shouldn't be changed at all when you print it, but large_tree_copy should have its root value as 10, and its first child's value being 3.

Hint: You'll want to create and return new tree using BinaryTree(self.value). The children of this BinaryTree should be copies of each subtree.

Printing the values of a BinaryTree in Level-order (print_level_order)

Last week, we asked you to write level-order printing for Trees. This week, do the same but for BinaryTrees. To do this:

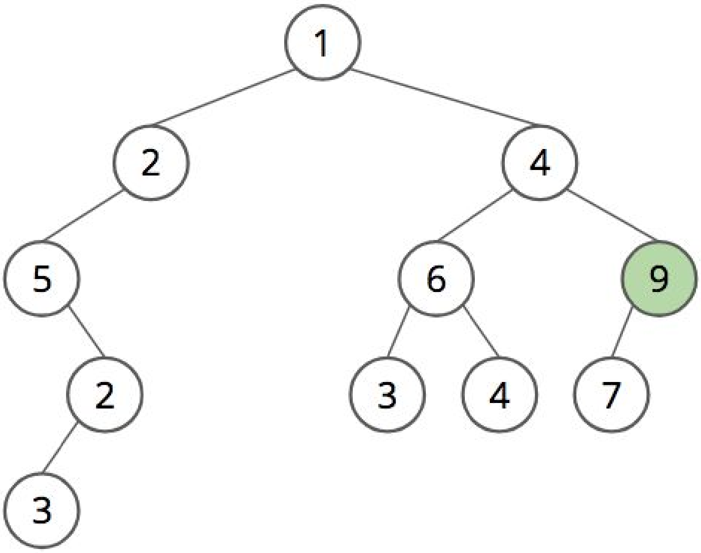
1. Make a Queue
2. Add our BinaryTree to it
3. Remove the BinaryTree at the front of the Queue
4. Print its value
5. Add the left subtree to the end of the Queue
6. Add the right subtree to the end of the Queue
7. Repeat steps 3-6 until the Queue is empty.

What happens if we flipped steps 5 and 6? (i.e. added right before left?)

Insert a BinaryTree before a value (insert_before)

Write a function that takes in a BinaryTree (or None) and 2 values: to_insert and to_find, which adds a new BinaryTree such that it appears wherever a node with the value to_find occurs. The subtree with to_find as its value should become the left subtree of the new BinaryTree. This function should return the root of the BinaryTree (or None if None was passed in). If to_find isn't in the BinaryTree, nothing should happen.

For example, if we called `insert_before(large_tree, 9, 7)`, large_tree should become the following afterwards:


1
2
4
5
6
9
2
3
4
7
3