CS-UY 1134: Data Structures and Algorithms
Lab 8: Linked Lists and Binary Trees

Friday, July 19, 2019

Instructions

This lab will focus on linked lists and binary trees.

Try to solve all of the problems by the end of the lab time. As added practice, it is a good idea to try writing your code with pen and paper first, as you will have to do this on the exam.

When you type up your code, write some test cases. Try multiple test cases for each function you have to implement, and consider edge cases carefully.

As a reminder, you *may* (and are encouraged to!) discuss these problems with your classmates during lab and help each other solve them. However, make sure the code you submit is your own.

What to submit

If you believe you have solved all the problems, let a TA or the professor know and they will check your work. This is a good chance to get feedback on problems similar to what you might see on the exam.

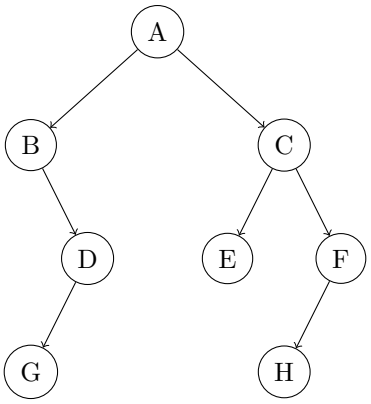
If you don't finish by the end of the lab, you may submit something on NYU Classes by Sunday July 21, 11:55 PM.

Either when a TA has checked your solutions, or by the Sunday submission deadline, submit one ".py" file containing your solutions to the programming problems, and one additional file containing your answers to the first (written) problem. Submit your files to the "Lab 8: Linked Lists and Binary Trees" assignment on NYU Classes.

1 Written Problems (Do These!)

Complete the following written problems, and submit your answers on Classes along with your code. Submit one file with all of your answers (so you should submit a

Consider the following binary tree.



1. Give the preorder, inorder, and postorder traversals of the tree.
2. What is the total height of the tree?
3. What is the depth of node D?

2 Circular Shift Linked List

We define a "right circular shift" operation as follows. If we have a sequence $\langle a_1, a_2, \ldots, a_n \rangle$, the **right circular shift** of that sequence is $\langle a_n, a_1, a_2, \ldots, a_{n-1} \rangle$. That is, we move the last element to the first position and shift all other elements down to the next position.

For example, the right circular shift of $\langle 2, 4, 6, 8, 10 \rangle$ is $\langle 10, 2, 4, 6, 8 \rangle$.

Implement the function:

```
def right_circular_shift(lnk_lst)
```

which performs the right circular shift operation on the given *doubly linked list*, `lnk_lst` in place. The function should *mutate* the linked list, and should **not** create and return a new linked list. Use the `DoublyLinkedList` class from lecture (available on Classes).

For example, if `lnk_lst` is `[2 <--> 4 <--> 6 <--> 8 <--> 10]`, the function should modify `lnk_lst` to be `[10 <--> 2 <--> 4 <--> 6 <--> 8]`.

Implementation Requirements

1. Your implementation must run in **worst-case constant time**.
2. For this implementation, you are **not** allowed to use the `delete_node`, `delete_first`, `delete_last`, `add_after`, `add_before`, `add_first`, or `add_last` methods of the DoublyLinkedList class. You **should** change the values of `prev` and `next` for some nodes.

3 Count Values in Binary Tree

Write a function to count the total number of occurrences of a given value in a binary tree. Your function should take two parameters: a `root` node, and a `value` to count. The function will return the number of times that the `value` appears in the tree. Implement this as:

```
def count_val(root, value)
```

Use the LinkedBinaryTree we wrote during lecture (available on Classes). Specifically, the parameter `root` is of type `LinkedBinaryTree.Node`.

Your function must run in **worst-case linear time**. That is, time $O(n)$ for a binary tree containing n nodes.

4 Invert a Binary Tree

Write a function that will "invert" (create a mirror image of) a binary tree. It should accept a single argument of type `LinkedBinaryTree` (use the code from lecture, available on Classes) and return a new `LinkedBinaryTree` object that has its values *inverted* (that is, a mirror image) from the original tree. The original tree that was passed in should *not* be modified at all. Here is an example:

Write your implementation in a function:

```
def invert_binary_tree(bin_tree)
```

Note that unlike most of the other functions, you the parameter and return value here is a `LinkedBinaryTree` object, and **not** a `Node` object. **Hint:** You may want to define a helper method.

5 Count Number of leaves, 1-child, and 2-child nodes

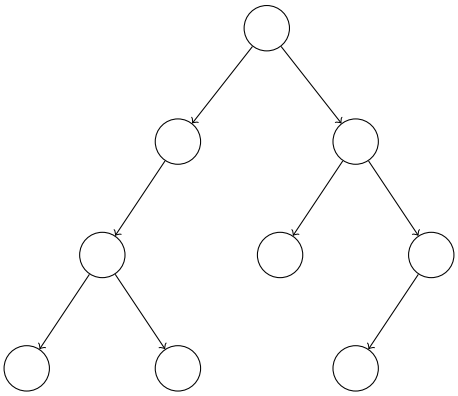
Give a **recursive** implementation of the following method in the `LinkedBinaryTree` class (that is, add this method to the class):

```
def subtree_children_dist(self, subtree_root)
```

Which computes the number of nodes in the subtree rooted by `subtree_root` that have 0 children, 1 child, and 2 children; and returns each of these numbers. That is, the function will return a list of length 3:

- The first element of the returned list is the number of leaves (i.e., nodes with no children) in the subtree.
- The second element of the returned list is the number of nodes in the subtree with a single child.
- The third element of the returned list is the number of nodes in the subtree with two children.

As an example, let `subtree_root` be a reference to the root of the following tree:



Then calling the function `subtree_children_dist` with `subtree_root` should return `[4, 2, 3]` because the subtree rooted by `subtree_root` has 4 leaves, 2 nodes with a single child, and 3 nodes with two children.

Your implementation should run in **linear time** (that is, $O(n)$ for a subtree with n nodes).