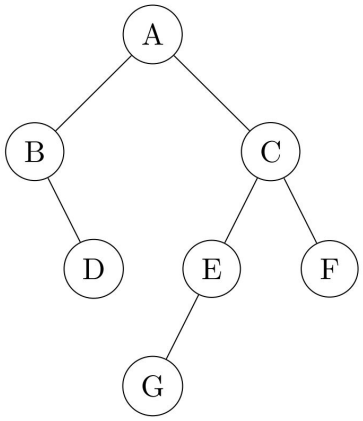
- This lab will cover Binary Trees.
- You may want to refer to the text and your lecture notes during lab as you solve the problems.
- When approaching the problems, think before you code. It is good practice and generally helpful to lay out possible solutions for yourself.
- You should write test code to try out your solutions.
- You must stay for the duration of the lab. If you finish early, you can help other students to complete the lab. If you don't finish by the end of the lab, it is recommended that you complete the lab on your own time.
- Your TAs are available to answer your questions in lab, during office hours, and on Piazza.

Vitamins (maximum 30 minutes)

---

1. Given the following binary tree:



   a. Give the postorder, preorder, and inorder traversals of the tree?

   b. What is the total height of the tree?

   c. What is the depth of node E?

2.

a. Draw the expression tree that corresponds to the expression:

((3+2) * (5-1))/(8+6)

Write the prefix form of the expression:

_____

b. Draw the expression tree that corresponds to the following (postfix) expression:

2 4 1 - * 5 +

What is the value of the above expression?

_____

3. For the following question, use the barebones definition of a binary tree node.

```
class TreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
```

Give each function a meaningful name. You *may* find it helpful to trace the execution of this code. Think about the different sizes and shapes of trees possible.

a.

```
def mystery(root):
    if root is None:
        return 0
    elif root.left is None and root.right is None:
        return 1
    return mystery(root.left) + mystery(root.right)
```

b.

```
def mystery(p, q):
    if p is not None and q is not None:
        return (p.data == q.data) and \
               mystery(p.left, q.left) and \
               mystery(p.right, q.right)
    return p is q
```

4. Given the following preorder and inorder traversals, recreate the binary tree and give the postorder traversal.

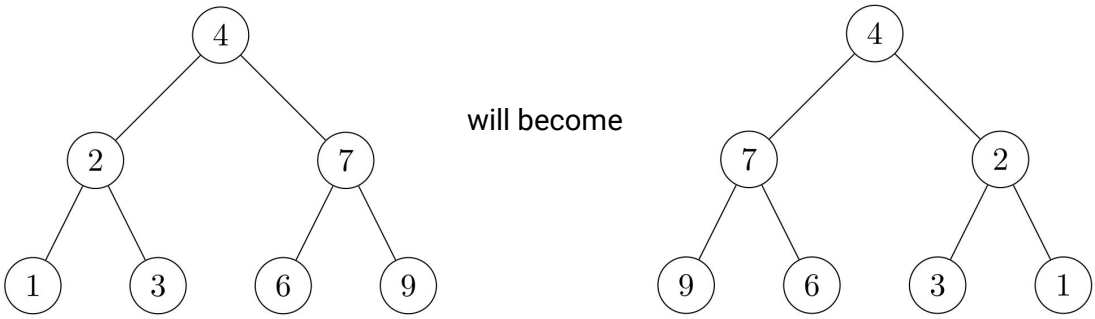
    i. Pre: BCAD
    In: CBAD

    ii. Pre: DBACEGF
    In: ABCDEFG

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code.

1. Write a function that will invert a binary tree. It should take a `LinkedBinaryTree` type and return a new `LinkedBinaryTree` that has its values inverted from (mirror image) of the original tree, leaving the original tree untouched. Here is an example:



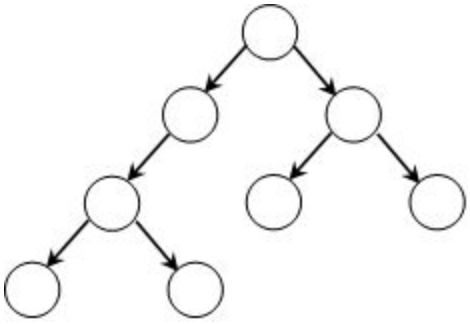
2. Give a **recursive** implementation of the following method in the `LinkedBinaryTree` class:

```
def subtree_children_dist(self, curr_root):
```

When given `curr_root`, a reference to a root of some subtree, it will return a list, of length 3, which represents the distribution of the number of children of all nodes in the subtree rooted with `curr_root`. That is:

- The first element in the list should be the number of leaves (nodes with no children)
- The second element in the list should be the number of nodes with a single child
- The third element in the list should be the number of nodes with two children

For example, let `curr_root` be a reference the root of the tree below:

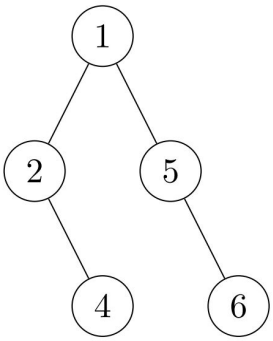


The call to subtree_children_dist with curr_root should return [4, 1, 3] (there are 4 leaves, 1 node with a single child, and 3 nodes with 2 children).

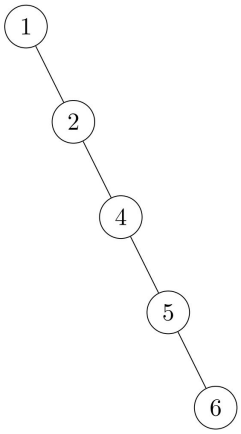
Implementation requirements: Your function should run in **linear time**.

3. Given a binary tree, flatten it to a linked list **in-place**. This means you should **not** return a new binary tree or linked list.

   For example:



will become



   Note that the left child of all of the nodes in the resulting tree will be `None`. We are not requiring you to put the flattened tree nodes in a particular order.

   Extra: Now think about how to solve the problem so that an inorder traversal of both trees will yield the same result.