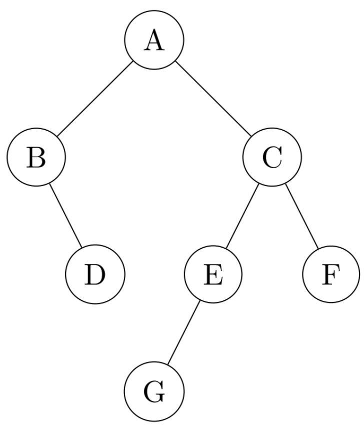
- This lab will cover Binary Trees, Depth-First Search, and Breadth-First Search.
- It is assumed that you have reviewed chapter 7 & 8 of the textbook. You may want to refer to the text and your lecture notes during the lab as you solve the problems.
- When approaching the problems, think before you code. Doing so is good practice and can help you lay out possible solutions.
- Think of any possible test cases that can potentially cause your solution to fail!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time. Ideally you should not spend more time than suggested for each problem.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (15 minutes)

---

1. Given the following binary tree, complete the following (5 minutes):



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

b. Give the level order traversal (Breadth-First) of the tree:

c. What is the height of the tree?

d. What are the depths of nodes E, B, and G?

2. Draw the binary tree given the following traversals (10 minutes):

   <u>preorder</u> : 11, 6, 4, 5, 8, 10, 19, 17, 43, 31, 49

   <u>inorder</u> : 4, 5, 6, 8, 10, 11, 17, 19, 31, 43, 49

   Is it possible to draw a unique binary tree given only its preorder and postorder? If not, draw two trees with the same preorder and postorder traversal.

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code. For each problem, **you may not call any methods defined in the `LinkedBinaryTree` class. Specifically, you should manually traverse the tree in your function. Each node of the tree contains the following references:** `data, left, right, parent`**.**

Download the **LinkedBinaryTree.py** file under Labs on NYU Brightspace

1. Write a *recursive* function that returns the sum of all even integers in a `LinkedBinaryTree`. Your function should take one parameter, *root* node. You may assume that the tree only contains integers. (5 minutes)

```
def bt_even_sum(root):
     ''' Returns the sum of all even integers in the binary
     tree'''
```

2. Write a *recursive* function that determines whether or not a value exists in a `LinkedBinaryTree`. Your function should take two parameters, a *root* node and *val*, and return True if *val* exists or False if not. (10 minutes)

```
def bt_contains(root, val):
     ''' Returns True if val exists in the binary tree and
     false if not'''
```

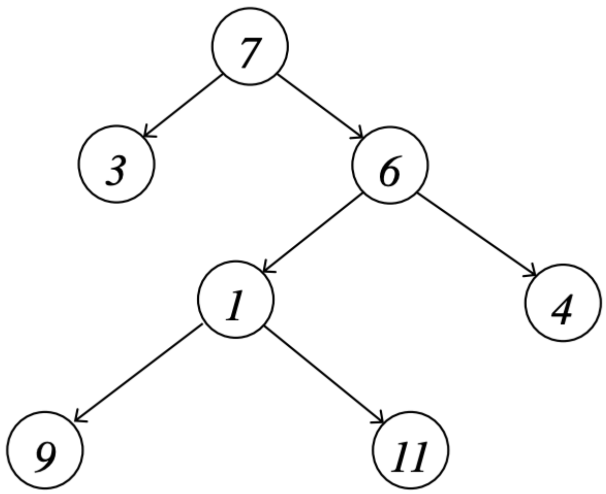
3. A **full binary tree** (or **proper binary tree**) is a **binary tree** in which every node in the tree has 2 or 0 children.

   Implement the following function, which takes in the root node of a `LinkedBinaryTree` and determines whether or not it is a full tree. <u>This function should be recursive.</u> (20 minutes)

```
def is_full(root):
```

```
''' Returns True if the Binary Tree is full and false
if not '''
```

ex)



4. Write a function that will invert a `LinkedBinaryTree` in-place (mutate the input tree)

   You will write two implementations, one recursive, and one non recursive.

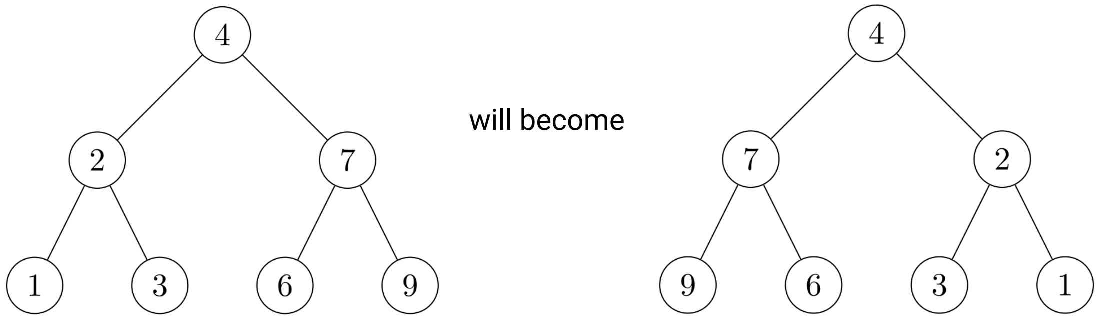
   Both functions will be given a parameter, *root*, which is the root node of the binary tree. (25 minutes)

```
def invert_bt(root):
    ''' Inverts the binary tree using recursion '''
```

```
def invert_bt(root):
    ''' Inverts the binary tree without recursion '''
```

Hint: For the non recursive implementation, you should use the *breadth-first search*.

ex)



5. Add the following method to the `LinkedBinaryTree` class (35 minutes).

```
def preorder_with_stack(self):
    ''' Returns a generator function that iterates through
    the tree using the preorder traversal '''
```

The method is a generator function that when called, will iterate through the binary tree using preorder traversal without recursion. **You will use one ArrayStack and Θ(1) extra space.**

Preorder is as followed: **Data, Left, RIght**. When you're tracing the preorder traversal of the binary tree, imagine how you would place the nodes in the stack, and when you would pop the nodes from the stack. Think of recursion and the call stack!

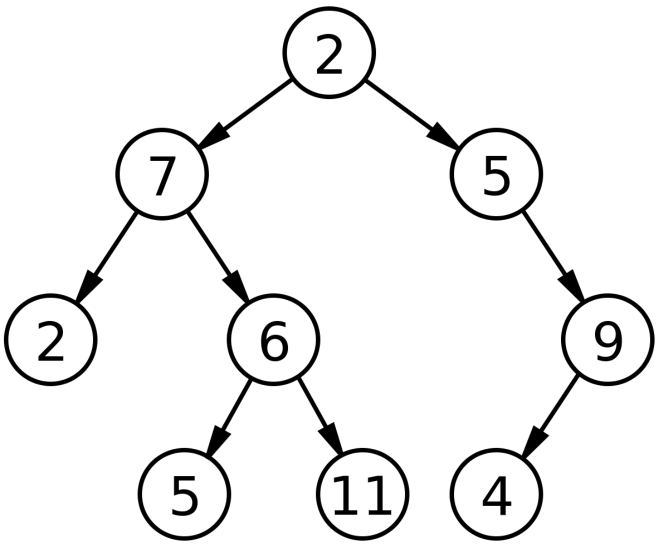
ex) Given the following code using the tree example below:

```
#t is a LinkedBinaryTree
for item in t.preorder_with_stack():
```

```
    print(item, end = ' ')
print()
```

You should expect the following output:

```
2 7 2 6 5 11 5 9 4
```



Optional - Vitamins

3. Draw the expression tree for the following expressions (given in prefix, postfix, or infix): Remember that the numbers should be leaf nodes. (10 minutes)

   a. 3 4 - 2 + 5 *

   b. (3 * 2) + (4 / 6)

   c. / + 9 9 2

4. Draw the following Binary Tree structure after executing the following code (5 minutes):

```
a = LinkedBinaryTree.Node(5)
b = LinkedBinaryTree.Node(4)
c = LinkedBinaryTree.Node(6, a, b)
d = LinkedBinaryTree.Node(8)
e = LinkedBinaryTree.Node(10, None, d)
f = LinkedBinaryTree.Node(12, e, c)

bin_tree = LinkedBinaryTree(f)
```

Optional - Coding

6. Write a function that will merge two `LinkedBinaryTree`. The function will take two parameters, root1, root2, of 2 binary trees and return the root of a new binary tree containing nodes created by merging each node of the same positions from the original

trees. If only one node exists in a specific position, simply use that value as the node for the new tree. **You may also define additional helper functions.** (25 minutes)

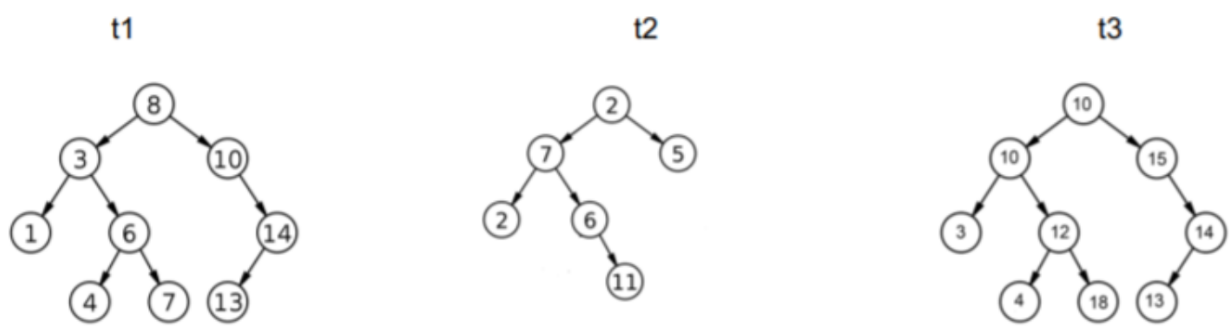
```
def merge_bt(root1, root2):
     ''' Creates a new binary tree merging tree1 and tree2
     and returns its root. '''
```

In the following example, notice that the new root is 10, the result of merging root1 and root2 (8 + 2).

In the case where there is only one node at a given position (14 in t1, there is no corresponding node in t2), the value 14 is used instead for t3.

The root node of t3, containing 10, should be returned by the merge_bt function.

Note that all of the nodes must be newly created. t3 should not have any nodes referencing a subtree of t1 or t2.



7. A **complete binary tree** is a **binary tree** in which every level of the tree contains all possible nodes.

   Implement the following function, which takes in the root node of a `LinkedBinaryTree` and determines whether or not it is a full tree. <u>This function should be iterative.</u>
   (20 minutes)

```
def is_complete(root):
    ''' Returns True if the Binary Tree is complete and
    false if not '''
```

<u>Hint:</u> For the non recursive implementation, you should use the *breadth-first search*.

ex)