- This lab will cover Binary Trees.
- 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!
- If you finish early, you may leave early after showing the TA your work. Or you may stay and 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

---

1. 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)
```

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`. You should overload in operator for your linked binary tree class so that it takes an item and returns True if the item exists or False if not.(10 minutes)

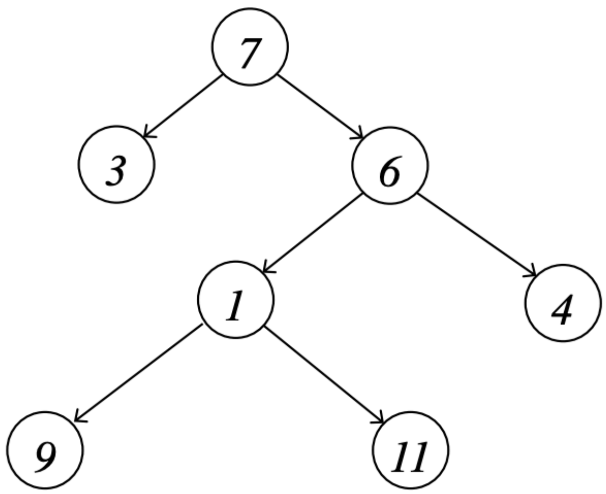
```
def __contains__(self, item):
    ''' Returns True if val exists in the binary tree and
    false if not'''
```

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 method for **LinkedBinaryTree.py** to overload __add__ operator. This method should merge two instances of `LinkedBinaryTree`. Implement it such that it takes two binary trees with roots root1 and root2 and returns a new binary tree. In the merged tree, nodes from corresponding positions in the original trees are combined. If nodes exist in both positions, their values should be added. If a node exists at a position in only one of the trees, that node should be directly included in the new tree. You may also define additional helper functions. (25 minutes)

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

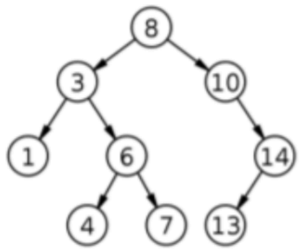
In the following example, notice that the new root is 10, the result of merging root1 and root2 (8 + 2).

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.

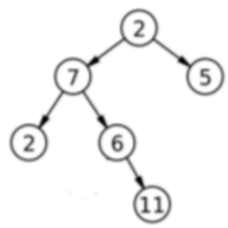
The root node of t3, containing 10, should be returned by the merge_bt function.

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

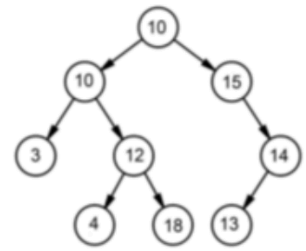
t1



t2



t3



Coding optional

---

5. 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.

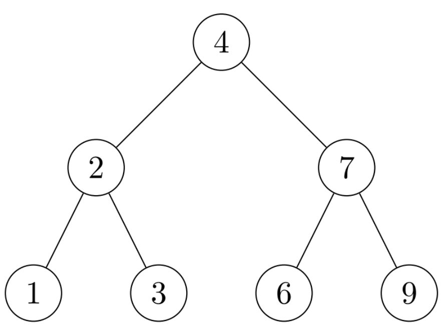
   Both functions will be given a parameter, *root*, which is the root node of the binary tree. (25 minutes)

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

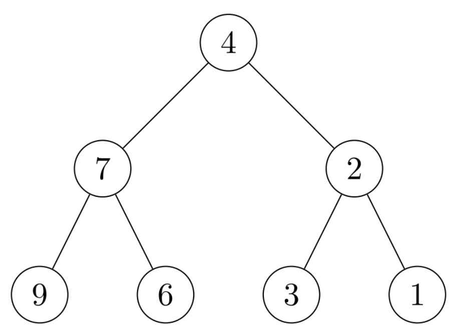
```
def invert_bt(root):
    ''' Inverts the binary tree without recursion '''
```

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

   ex)



will become



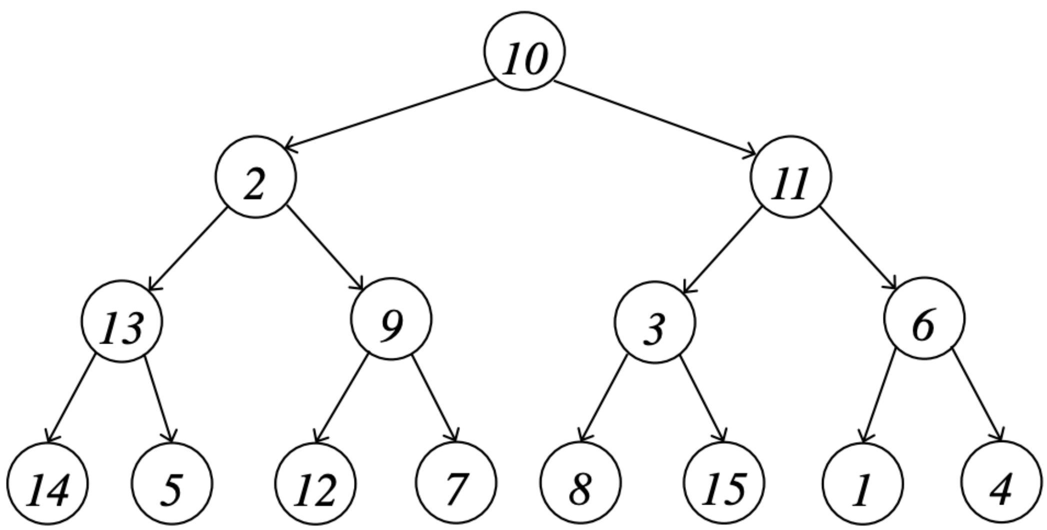
6. A **complete binary tree** is a **binary tree** in which every level of the tree contains all possible nodes.

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

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

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

   ex)



1. 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