CISC 160
Lab 04

Background
Trees are hierarchical structures that allow for many different uses in data storage and application. However, in order to be able to utilize this structure, you have to be able to build and traverse a tree first. This lab will look specifically at how to do those two things.

Nodes
Before we can make a linked tree structure, we need a node to build our trees from.

1. Download double_node.py. **Extend** DNode to make a BinaryNode. This BinaryNode will represent a node in a binary tree. Your BinaryNode should contain references to its parent node, to its left child node, its right child node, and the element that the node contains. The BinaryNode object is to be contained in file called Binary_Node.py. Be sure that the BinaryNode object includes the following methods:
    - Constructor which takes an element, a parent node, a left child and a right child, in that order and that default to None if no parameter is given
        - See Node and DNode for examples of how to do this
    - get_element()
    - get_parent)
    - get_left()
    - get_right()
    - set_element(new_element)
    - set_parent(new_parent)
    - set_left(new_left)
    - set_right(new_right)

Tree Creation
Next, we need to be able to create a tree. There are many algorithms which we can use to do this, but we will use an algorithm which generates a sorted binary tree:

```
If root is NULL
   then create root node
   return root

Else If root exists then
   compare the data with node.data

   while until insertion position is located

      If data is greater than node.data
         goto right subtree
      else
         goto left subtree

   endwhile

   insert data

end If

return root
```

2. Download the file LinkedTreeFunctions.py. Use (and do not copy or alter) the BinaryNode object that you altered in question 1 to implement the `insert` function declared in LinkedTreeFunctions.py. As mentioned in the associated comment, this function should take a BinaryNode (which represents the root of a sorted binary tree) and a new element to be inserted into the tree. The function will then insert the new element into the sorted binary tree using the algorithm presented above so that it stays sorted. When you are done, you should return the BinaryNode at the root.

   NOTE 1: You may assume that elements in tree added element are compatible with respect to equivalence and comparison operators.

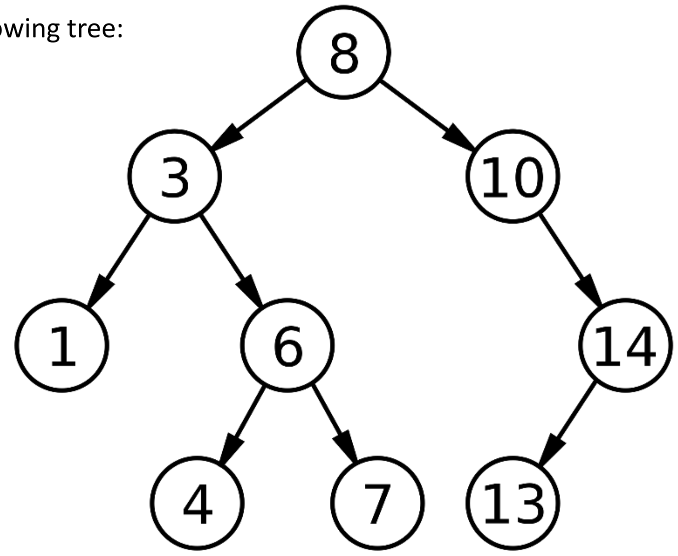
   NOTE 2: Please be sure to examine and use the publicly available methods BinaryNode objects, not the hidden attributes.

3. Implement the private `_print_tree_inorder` function in the LinkedTreeFunctions file which will take the root of the tree and print the tree using an inorder traversal. Inorder traversal prints the left subtree first, then prints the element in the current node, then prints the right subtree.

Tree Search

Finally, we need a way to search our binary tree. Luckily, as it is sorted, our search algorithm can use that knowledge to be much simpler. Our goal is to return the “path” of elements from the root to the node in question. To do this, the element in the root is placed on the front of a list. Then the next node to go to is determined and traversed to. The element for that node is added to the list. If the node represents the element being searched for, then the list is returned. If not, continue searching. This process stops when the element is found or when the next node to traverse to from the current node is None.

For example, given the following tree:



The path to 7 is [8, 3, 6, 7], the path to 10 is [8, 10], the path to 8 is [8], and the path to 9 is None.

4. Implement the `iterative_path` function declared in LinkedTreeFunctions.py. As mentioned in the associated comment, this function should take a BinaryNode (which represents the root of a sorted binary tree) and an element to search for within the tree. The function searches through the tree using **<u>an iterative methodology</u>** and constructs a list of each element along the path. Once the function finds a node with the element being search for, the function returns the list of the path. If the element, isn't found, the function should return `None`.

   <u>NOTE 1</u>: You may assume that elements in tree added element are compatible with respect to equivalence and comparison operators.

   <u>NOTE 2</u>: Please be sure to examine and use the publicly available methods BinaryNode objects, not the hidden attributes.

5. Implement the `recursive_path` function declared in LinkedTreeFunctions.py. As mentioned in the associated comment, this function should take a BinaryNode (which represents the root of a sorted binary tree) and an element to search for within the tree. The function searches through the tree using **<u>a recursive methodology</u>** and constructs a list of each element along the path. Once the function finds a node with the element being search for, the function returns the list of the path. If the element, isn't found, the function should return None.

   <u>NOTE 1</u>: You may assume that elements in tree added element are compatible with respect to equivalence and comparison operators.

   <u>NOTE 2</u>: Please be sure to examine and use the publicly available methods BinaryNode objects, not the hidden attributes.

Lab Requirements

Download all source code files on the Canvas page. Create Binary_Node.py and alter LinkedTreeFunctions.py. You may add any additional helper methods as needed. Helper functions should be named with an underscore at the front as they are not designed to be publicly accessible. If you add any helper functions, be sure to comment it in the same style as the other, included functions.

Additionally, the "testing structure" of `if __name__ == '__main__'` has been included at the bottom of the LinkedTreeFunctions.py file. You should use that section for your test code. I may look at it for understanding of how you are approaching testing but you will not be explicitly graded on code included there.

Remember: These requirements may not be all encompassing. Use your brain, your knowledge of the system, and any descriptions within the code as sanity checks and reminders to make a complete system.

Submission: Submit your Binary_Node.py and LinkedTreeFunctions.py code files. You do not need to submit this lab booklet, the code files themselves will be sufficient.

Submissions must be submitted in unzipped .py files. **FAILURE TO SUBMIT UNZIPPED WILL CONSTITUTE FAILURE TO SUBMIT.**