Computational Problem Solving **CSCI-603-01**
Least Common Ancestor **Exam 2: Practical**

Full Name (printed): _____

Time Finished: _____

Instructions

- You have 75 minutes to complete this practical and upload it to MyCourses.
- Your program should run using Python version 3.
- You are not required to comment the code you write for this exam.
- You may not communicate with anyone except for the proctor.
- You are not allowed to look at any other programs you have written prior to this exam.
- The only applications allowed are PyCharm (or any other IDE), and a web browser for accessing MyCourses. You are not allowed to use any other programs!

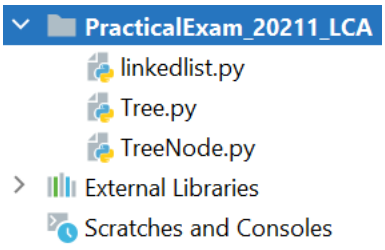
Failure to follow these instructions will result in automatic failure of the exam!

Getting started

For this programming assignment, you will be working on implementing a tree that represents a taxonomy - a representation of concepts and sub-concepts.

First, go to MyCourses, click on Assignments, and then Exam 2 Practical, Section 1. Download the starter code. Add the provided files to a new Pycharm project (if using Pycharm). Do not use any other versions of these files!

Your project structure must look as follows:



Here is a brief description of what is provided to you:

- `TreeNode` - represents a node in the tree. Instead of two children (`left` and `right`) we allow a `TreeNode` to have an arbitrary number of children. We do this using a list - note that the order the children appear in the list is not important and could just as well be represented as a set.
  This class provides a constructor, equality check and methods used to print the nodes on the standard output.
- `Tree` - represents the taxonomy tree. It has two fields, `root`, that points to the root node (or `None` if it doesn't exist) and `nodeLookup` which is a Python dictionary. We use `nodeLookup` to locate any node in our tree by its value (see `getNodeByValue`). For instance, `nodeLookup['dog']` will give us the node representing the value 'dog' if it is in the tree. Note that this is only possible because we assume that nodes have unique values and so 'dog' will not appear twice in the same tree.
- `linkedlist` - This is a simplified version of the `LinkedList` from lecture. This version uses a cursor, and provides a constructor, methods to manipulate the cursor, and the `append` and `prepend` methods. **This class is complete. You are NOT allowed to modify it.**

The following questions will ask you to finish writing some of the methods required to complete the implementation of the modules `TreeNode.py` and `Tree.py`. **Unless instructed, you are NOT allowed to modify anything else in the files.**

Feel free to add any test code outside of the classes to check your methods.

Efficacy counts! For example, do not implement $\mathcal{O}(n)$ algorithm if a $\mathcal{O}(1)$ solution exists!

1 Add a Parent Field to `TreeNode` (10 points)

Add a new field, `parent`, to `TreeNode` that will point to the parent of each node (or `None` if no parent exists). You may modify `TreeNode.__init__` but not any other methods in `TreeNode`.

2 `Tree.addChildTo` (20 points)

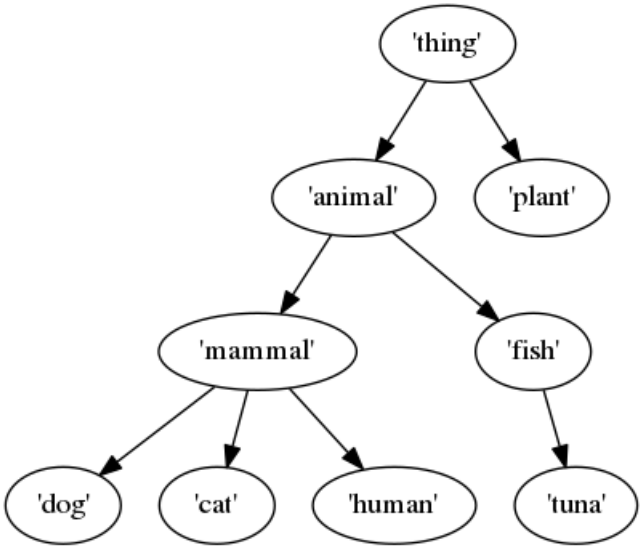
Implement the method `addChildTo(self, newChildValue, parentValue)` which creates a new child node with the value `newChildValue` under the node in the tree with the value equal to `parentValue`. You may find it helpful to look at the implementation of `addRoot` to get you started.

Remember you can get a node by its `value` using `getNodeByValue` or using `nodeLookup`.

Be sure to add any nodes you create to the `nodeLookup` dictionary.

3 Represent a Taxonomy (15 points)

Find the `test()` function at the bottom of the file. We have already created a tree `t`, initialized a root, 'thing', and added a node for 'animal' as a child of that root. Where the comment reads "add children here", write code that builds the rest of the tree depicted below. Remember that the order the children appear does not matter.



4 `Tree.getPathToAncestor` (25 points)

Implement `getPathToAncestor(self, nodeValue, ancValue)` which returns a linked list of all of the nodes between the node with the value `nodeValue` and the node with the value `ancValue` (both included) in order from shallowest to deepest. You can assume that `ancValue` will always point to a node that is an ancestor of the node specified by `nodeValue`.

- `getPathToAncestor('dog','animal')` returns `animal --> mammal --> dog`
- `getPathToAncestor('dog','mammal')` returns `mammal --> dog`
- `getPathToAncestor('animal','thing')` returns `thing --> animal`
- `getPathToAncestor('animal','animal')` returns `animal`

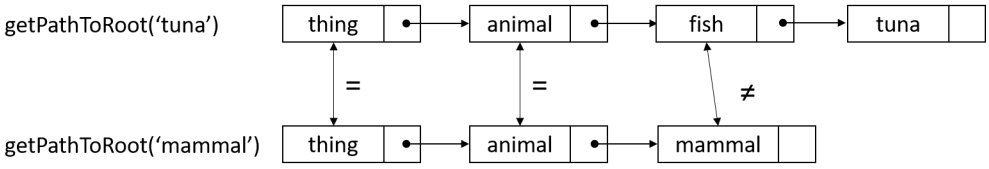
5 `Tree.getLCA` (30 points)

In inheritance graphs it is often important to find the least common ancestor (LCA) of two nodes. The LCA of x and y is the deepest (the one furthest away from the root) common ancestor of x and y.

For instance:

- `getLCA('cat','cat')` returns `'cat'`
- `getLCA('cat','dog')` returns `'mammal'`
- `getLCA('cat','tuna')` returns `'animal'`
- `getLCA('tuna','mammal')` returns `'animal'`

Implement `getLCA(self, node1Value, node2Value)` which returns the LCA of the nodes with the given values. To do so, we first call `getPathToTop` for both nodes, then we walk forward both lists comparing the elements at each position. The last element that matches is the LCA.



Submission

Submit only your `TreeNode.py` and `Tree.py` modules to the MyCourses dropbox. Verify your submission by reloading and checking the dropbox.