Practical Questions CSCA48 – Week 5

Question #1

1) Create a BST by inserting the nodes 3, 6, 4, 1, 2, 7, 9, 5, 8 in the order given
2) Write down an inorder, pre-order and post-order traversal of your tree
3) Draw a binary tree such that 10, 6, 8, 12, 11, 15 is a pre-order traversal of that tree
4) Draw a binary tree such that 8, 6, 12, 10, 11, 15 is an in-order traversal of that tree
5) Can you draw a single tree that meets the criteria for both of the previous questions?
6) Can you come up with a general algorithm to solve this problem? That is: given a pre-order and an in-order traversal of a tree, re-construct that tree? Is it always possible? Is the tree always unique?
7) For the following questions, you may add any helper methods you wish (think about whether they should be public or private)
   a) Implement a BinTreeNode class and a BinaryTree class, with all the basic methods described in class. We know you have the code already, but make sure you can do it on your own without looking at Nick's code.

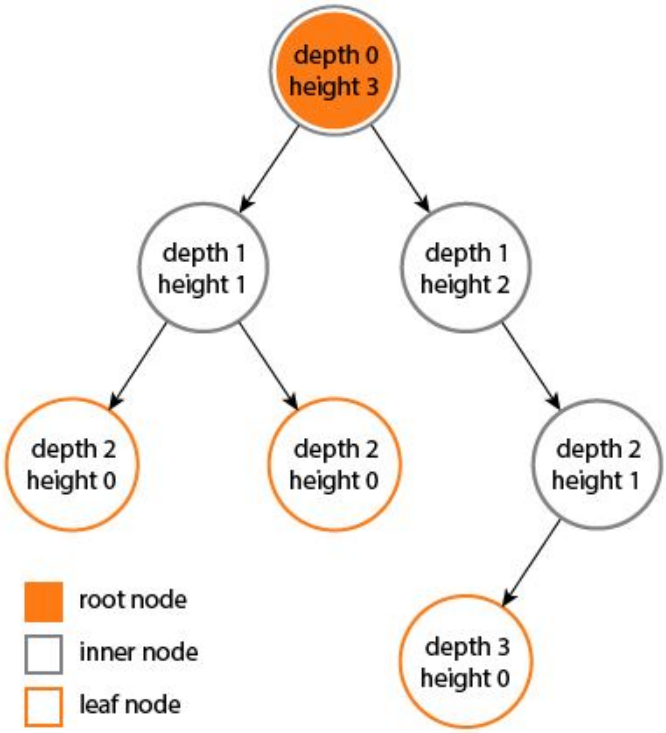
   b) Add a method to your BinTreeNode class called *size(self),* it returns the number of nodes in the tree rooted at that node.

   c) Add a method to your BinaryTreeNode class called *second_smallest(self)*, it returns the second smallest data value found in the tree rooted at that node. Hint: draw a few trees first… where is the 2^{nd} smallest node found?

   d) Add a method to your BinTreeNode class called *depth(self, value),* it returns the depth of the node with *value* in the tree rooted at

the given node. The depth of a node is the number of edges from the node to the tree's root node.

e) Add a method to your BinaryTree class called *height(),* it returns the height of tree. The height of a tree is the number of edges on the longest path from the root to a leaf.



f) **(Challenge)** Add a method to your BinaryTree class called *sameTree(self, other_tree),* it returns True this tree and other_tree are structurally identical; they are made of nodes with the same values arranged in the same way.

g) (Challenge)

Add a method to your BinaryTree class called *mirror(self),* it changes the tree so that the tree is reversed. So the tree..

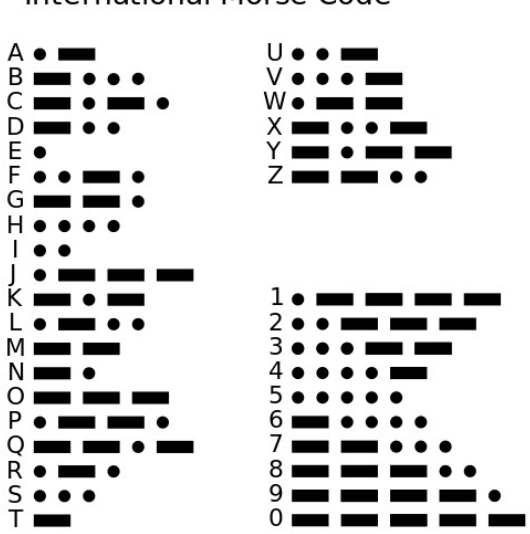
```
     4                                   4
    / \                                 / \
   2   5          is changed to …      5   2
  / \                                     / \
 1   3                                   3   1
```

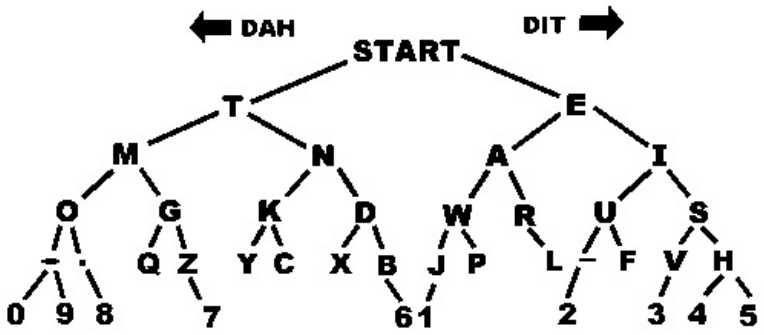
h) (Challenge)

Take a look and study the Morse code and the tree below.

Now you write a function called decrypt that decrypts morse code messages

>>> decrypt("_. . . _._. _._")
'NICK'





Discussion Questions

1. Finding things in binary search trees is more efficient than in a normal linked list. Is this true of all binary trees? Is it necessarily true of all binary search trees? What is the best case scenario for a BST? What is the worst-case scenario?
2. The trees we've been dealing with are unidirectional. That means that parents link to their child nodes, but children nodes don't know where to find their parent node (that came out sounding a lot sadder than originally intended). What if we wanted them to be bi-directional? So that every node has a link to its parent. Sort of like a tree version of a doubly linked list. How would that work? Would there be any benefits to it? Any drawbacks?
3. The trees we've been dealing with are acyclic. That means that there is exactly one path from the root to every node in the tree. No node can have 2 parent nodes, and a node can't have a child node that is its own ancestor. Are there situations where we may want to relax these constraints? What might be the benefits/drawbacks of these sorts of data structures?

Logic Question (Challenge):

In a bag, I have 1 red ball, 2 orange balls, 3 green balls, 4 yellow balls, 5 blue balls, 6 purple balls and 7 brown balls. Removing one ball at a time from the sack, without replacement, how many balls must I draw to guarantee that I'll have at least 3 balls of the same colour?