We will have a quiz on Thursday over trees. Mostly definitions but maybe fix a method that is broken or add a missing line. Make sure to study Days 22 and 23!!!!

A. Tree code

1. Study treecode.py we already wrote add2tree() and find() in the last class

2. Study prettyprint() and notice how it prints out the tree vertically. If you turn your head 90° to the left, the root will be at the top. Study the code and notice how it conforms to the same recursive pattern. Also notice how the indent level is increased for each recursive call so that it will print "tree-like"

3. Write a method isLeaf() that returns True if the node is a leaf. (easy)

4. Now we want to find the longest path in a tree, i.e. the most nodes (or hops) in a path from the root to a leaf. The root counts as 1. Use isLeaf(). If there is a simple tree like the following:

   ![Tree Diagram]

   the longest (or deepest) path is 4 because dog is the 4th node in a path cat → ape → asp → dog.

   How would you do this?
B. Types of trees based on children

**Complete binary tree:** every node has 0 or 2 children:

![Complete binary tree diagram]

An imbalanced tree violates this property somewhere. You can also have ternary trees.

**Balanced tree:** There are $k$ levels. Every level 0, 1, 2, ..., $k-1$ are complete. The $k^{\text{th}}$ level might be incomplete. So every complete binary tree is balanced.

![Balanced tree diagram]

A fascinating way to implement a balanced tree is with an array (a list):

Here's a binary tree represented in memory, and pictorially:

![Binary tree representation]

The root is at position 1. The left child of a node at position $p$ is $2p$. The right child of a node at position $p$ is $2p+1$. (see exercises)

This assumes there are no "holes" in the tree. If there were, we would just use a special value to designate them. You can insert at the end only, but surprisingly you can still conceptually insert anywhere in the tree! It is just that no leaf in the $k^{\text{th}}$ level can have a child until all the $k-1$ nodes have a full quota of 2 children. So you could not give 107 a child yet. Not until 50, 84, 71 and 29 are full.
9 x 8 x 7 x ... x 1 = 9!

combinatorial explosion

cannot enumerate

"investigate"
D. Now for some mathematics:

Path: sequence of nodes that are traversed going from one node to another (or the sequence of edges if they are uniquely labeled.) Path may not contain cycles.

Height: the longest path in the tree

Levels: the number of levels in a tree is essentially the high

\[ \log_2(N) = \text{height of a balanced tree where } N = \text{number of nodes} \]

\[ 2^k = \text{maximum number of nodes at level } k \]

\[ \sum_{k=0}^{m} 2^k = 2^{m+1} - 1 \]

\[ 1 + 2 + 4 + 8 = 15 \]

\[ \sum_{k=0}^{m} 2^k = 2^0 + 2^1 + 2^2 + \ldots + 2^m \]

The logarithmic nature of a balanced tree is what gives it its power. You may have a trillion nodes, but the longest path is only 40.

Bushiness of a tree: its maximal outcount (BST, it is 2)

A large maximal outcount will cut down on the number of levels dramatically.
ape, bear, cat, dog, kangaroo, orca

N nodes

O(N)
\[(3 + 4) \times 7^{k+1}\]
E. Tree Traversals: 

Inorder: Visit left child; print root; visit right child

Preorder: print root; Visit left child; visit right child

Postorder: Visit left child; visit right child; print root

def inorder(self):
    self._inorder_aux(self.root)

def _inorder_aux(self, nodeptr):
    if nodeptr == None:
        return  # nothing to visit!

    self._inorder_aux(nodeptr.lchild)

    print(nodeptr.value+", ")

    self._inorder_aux(nodeptr.rchild)

You would see 10,20,25,30,40,50,66 in that order.

10 20 25 30 40 50 66

an inorder traversal of a BST results in a sorted list