CS 120: Intro to Computer Programming II

In-Class Activity - 06 Trees - Day 1

Activity 1 - Turn in this one

Discuss the following tree-related terms with your group. Give the definition for each one. Turn in your definitions; if your group has any interesting discussion or difference of opinion about these, report that as well.

- root

  Solution: The starting point for a tree; it has no parent. Analogous to the `head` pointer in a linked list.

- leaf

  Solution: A node that has no children

- child

  Solution: A node that has an incoming reference, from its parent. (A child can only have one parent.)

- parent

  Solution: A node that has one or more references to other nodes (its children)

- node

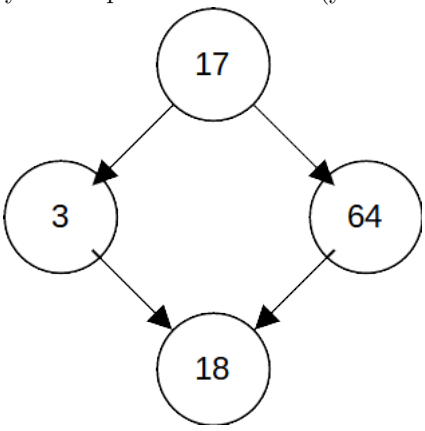
  Solution: A single object in the tree. Holds one value, and has some references to its children (or `None`)

- **binary** tree

  Solution: A tree where every node is limited to having no more than two children.

Activity 2 - Turn in this one

Why is this picture not a tree (you can notice something about its **shape**)?



Solution: Node 18 has two parents!

(activity continues on next page)

Activity 3 - Turn in this one

Draw a **binary** tree containing 7 nodes; make it as "wide" as you can, so that it will be as short as possible. Turn in this picture.

Consider the tree you just drew: how many node were there at each "level" of the tree? How many nodes would you need to add, in order to fill up another level? Discuss this with the group, and turn in your observations.

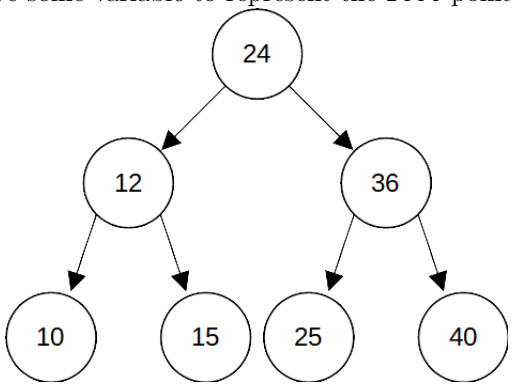
Now, draw a tree that contains the **same 7 values** - but which is now as **tall** as you can make it. What do you observe about its structure? Turn in your picture, and also you observations.

Solution: The picture in Activity 4 (see below) is a great example of a "wide" tree!
For the "tall" tree, just build a linked list of nodes.

Activity 4 - Turn in this one

In the video (and the slides) we showed you a `TreeNode` class, which is nothing more than a slightly-more complex version of `ListNode`.

Create a bunch of `TreeNode` objects, and link them together as follows. Remember, you will need to have some variable to represent the `root` pointer, just like you needed a `head` pointer with linked lists!



Solution:

```
# the actual order in which you build nodes doesn't matter,
# so long as you build a node *after* its parent!

root = TreeNode(24)

root.left  = TreeNode(12)
root.right = TreeNode(36)

root.left .left  = TreeNode(10)
root.left .right = TreeNode(15)
root.right.left  = TreeNode(25)
root.right.right = TreeNode(40)
```

Activity 5 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

Draw an example tree with a few nodes (give it an interesting shape, so that the code we're about to run has to handle a number of different cases). Then, run the following recursive function on the tree, and report what it returns.

Can you see what this function does? If not, draw a couple more trees and see if you can understand it better.

Once you think you understand, draw a couple more to prove that you're correct. Try to achieve some challenging goals: for instance, can you draw a large tree where this function will return 1? Can you draw a tree with relatively few nodes, where this returns a high number?

```
def mystery(root):
    if root is None:
       return 0

    if root.left is None and root.right is None:
        return 1
    else:
        return mystery(root.left) + mystery(root.right)
```

Solution: This function returns the number of leaves in a tree! The "best" tree - the one with the most leaves per nodes - is a "complete" tree (like our example from Activity 3.

(Actually, I realized later that **any** tree which never has any node with exactly one child is optimal in this sense; it's not optimal for search performance, but it's optimal in the sense of how many leaves it has!)

The "worst" tree (in the sense of fewwest leaves per node) is a "linked list" tree; it has only one leaf, no matter how many nodes it has.