Tree Mirroring

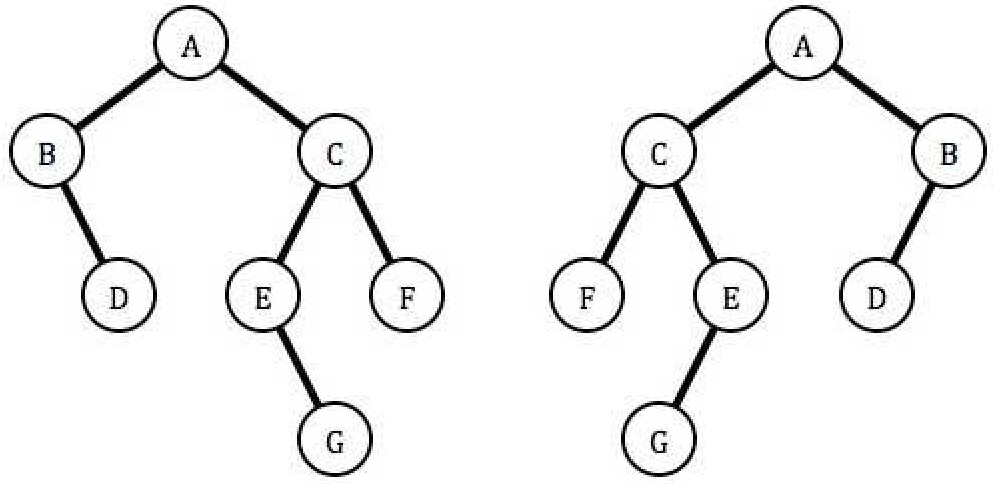
Implementing a Tree

Project Due: 07/21/20 @ 2:30pm

1.0 Objective

This project is meant to help you develop skills at working trees and writing entire programs from scratch. In this project you will only be provided with a project description and list of program requirements. No code will be provided at all. This means you will need to learn to design and implement an entire program on your own.

2.0 Project Description

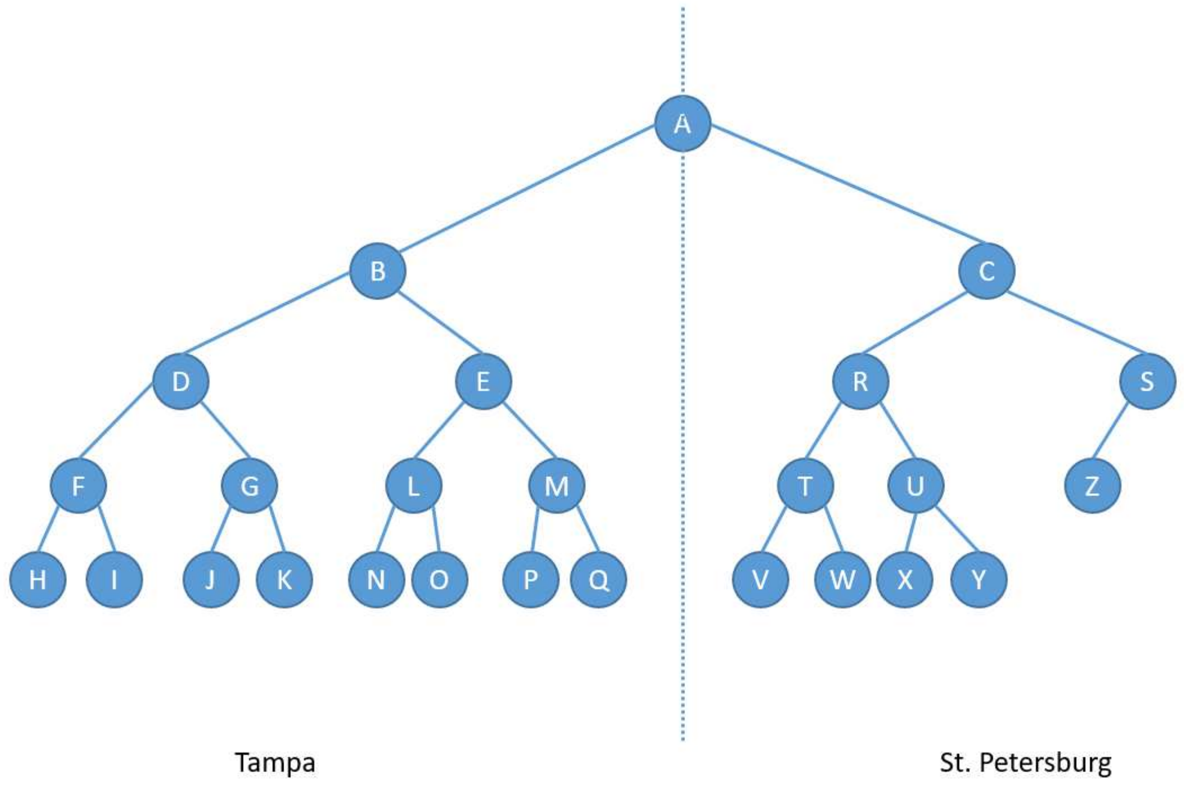


2.1 Basic Program

In order to start paying back your enormous student loans, you have set up a very profitable stolen car parts operation. Now that you've pretty much made all of the money that you need, you are planning on turning your operation over to one of your trusted lieutenants: B or C (everyone uses code names for security). B has a larger organization and has been doing better than C. However, being the good criminal manager that you are, you think that it might be because B steals from Tampa and C has to steal from St. Petersburg. In order to make a decision about who gets to inherit your organization, you are planning on making a top-to-bottom change in the organization to see how each group performs in a new environment. You've created a binary tree that represents how your organization is currently set up. You want to now create a mirror image of it so that you'll know how you want to go about restructuring it.

You are going to create a program that will allow you to create a tree. Once you have done this, you will print it out. Then you will create a mirror of the tree and print that tree out. This means that you will have to complete a function named **mirror** that alters the tree so that the final tree is a mirror image of the original tree. For example, if we use on this method on the tree shown on the left, we end up with the tree shown on the right.

The tree that you will create for this homework looks like this:



Note that you are limited to a tree depth of 5. The solution to this homework can be implemented using either Python lists or Python objects.

2.2 Files Names As Program Parameters

You must take one file (HW3Nodex.txt) as query string arguments. To make life easier for the TAs, include the following code in your program:

```
import sys


        if len(sys.argv) != 2:
            raise ValueError('Please provide one file name.')

        sys.argv[0] = sys.argv[0][0:len(sys.argv[0]) - sys.argv[0][::-1].find('/')]

        inputFile1 = sys.argv[0]+sys.argv[1]


        print("\nThe file that will be used for input is {0}".format(sys.argv[1]))
```

To pass the file name argument to your python program, in PyCharm run your program (it will fail) and then go to "Run > Edit Configurations" and set the Parameters value to " HW3Nodes.txt " and save it.

2.3 Reading from a File

To read the music playlist and the playlist commands from a file ("HW3Node.txt"), you must first use the Python *inputfile1 = open(filename,"r")* command. At this point you can start reading from the file using *variable = inputfile1.readline()*. Do be careful, if you program can't find the file that it is looking for it will crash.

2.4 What is a Binary Tree?

A binary tree is a data structure that stores information in sorted order. The main component of a binary tree is a node. The node holds the data, and also two other nodes, called left and right. The data in the left node precedes that of the containing node, and the data in the right node follows that of the containing node. The left and right nodes in turn also can have left and right nodes, and so forth.

2.7 Statistics

Another major part of this project is recording statistics. A summary of all the statistics you must keep are presented in the following table:

Statistic	**Description**
Number of Nodes	This is the total number of nodes in your binary tree.
Number of Nodes On Left Side [of initial tree]	Total number of nodes on the left side of your binary tree. (left of the root node)
Number of Nodes On Right Side [of initial tree]	Total number of nodes on the right side of your binary tree. (right of the root node)
Smallest Node Value	The smallest value of a node stored in your tree.

Largest Node Value	The largest value of a node stored in your tree.

Time	You need to record the total time of your program. This time should include the time to read the songs words, process the music player commands, and record all the statistics. It is suggested that you start a timer as one of the first things you do in the program and stop it immediately before printing out the statistics. The following is a little sample code that shows how you can time events.
	
	`import time`
	
	`start = time.time()`
	`...`
	`end = time.time()`
	`print(end - start)`

When the program is finished reading the command text file, it should print out a list of statistics. The output should look exactly like the following:

```
        **********************
        ***** Statistics *****
        **********************
        Number of Nodes xxx
        Number of Nodes On Left Side xxx
        Number of Nodes On Right Side xxx
        Smallest Node Value xxx
        Largest Node Value xxx
        Total Time of Program: xxxxx milliseconds
```

The x's should be replaced with actual numbers.

When the table is printed out, the format should be as follows:

```
A
     B
        D
           F
              H
              I
           G
              J
              K
        E
           L
              N
              O
           M
              P
              Q
     C
        R
           T
              V
              W
           U
              X
              Y
        S
           Z
```

3.0 Program Design Tips

The number one rule about writing a program from scratch is not to write a single line of code until you have developed a good design. Developing this good design is probably the hardest thing to learn how to do in programming. Here are a few of my suggestions on how you should go about designing and writing this project.

1. Read the program through once to get an idea of what is expected. Then, read it again. The second time through you may want to take some notes. At this point, you should have a thorough understanding of what is expected of you. If you need to read it a third time, do so.

2. Do a very rough design. This basically means figuring out what you are going to want to create and what the purpose of each part of the program will be. It is completely up to you; however, be sure you know why you need each part and what its purpose will be for.

3. Once you have a list of your parts, go through each part and determine what functions it will need and what each of these functions will do. This should include information about what parameters each function will take and what type of value it will return.

4. Then go through each function and write psuedo-code describing how the function will flow. This is a very critical step because if it is done properly, your code writing will be much simpler. A good piece of advice at this stage is when you are examining a function, assume all the other functions already work - even if you have not written the pseudo-code for a function. If you did Step 3 properly, you will know how each function should work, what arguments it will take, and what values it will return.

5. The very last step is to go ahead and write your project up. Do not write all the code at once. Write functions one or two at a time, test them and make sure they work as you expect and then go on to the next function or two. It is always best to start with the lowest level functions first and work your way up. In other words, if you have some function that does not call any other of your functions, then that is the one you want to start with. You can then test it and make sure it works (this often requires a little creativity). Once you have all of these functions written, you can then move up the chain to functions that only call these lowest level functions. Once you finish all of these, move up another level. Continue to do this until all of your functions are written.

One of the things you will notice about this design is that Step 4 calls for a top-down approach (describe all of the higher level functions assuming the lower level ones are done and then move down to the next level). Whereas, Step 5 calls for a bottom-up approach (write and test all of the low level functions before moving on to the higher level functions). I think if you follow this approach, you should find your programs (for this or any other class) get written much more quickly and with far fewer bugs.

4.0 Handing in Your Project

Be sure to comment your code well so that it can be easily browsed to see what is going on.

Code that is excessively difficult to read will be penalized.

Turn in a single file called "COP4530–HW#3.py" to the assignment on Canvas.

5.0 Grading

The project is due at 2:30pm on July 21, 2020. The project will be graded out of 100 points and be graded on the following criteria:

- Non-trivial program compilation
- Correct Output
- Performance (running time)

We will grade your code by running it against a series of test files and checking the output of the program against a known, correct implementation. Any differences in your output versus that of the correct implementation will result in a point deduction. Without a doubt, much more in depth tests will be run than the initial one provided. This means you should write some of your own test files and check for any errors.

We will also examine parts of your code to make sure that you are implementing the data structures as required. If you are getting the correct results but not implementing the proper data structure, your score will suffer severely. You must implement the proper data structures – it is a data structures course after all!

6.0 Python Tip

The Python language allows you to create strings of different lengths using multiplication.

The statement:

```
spaces ="aaa" * 3
```

would result in a string of 9 "a".

This may be useful when you are deciding how you want to go about printing out the binary trees.