California State University, Sacramento
College of Engineering and Computer Science

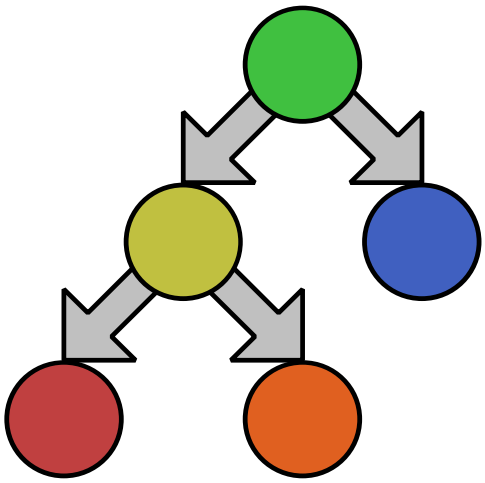
Computer Science 130: Data Structures and Algorithm Analysis

Assignment #2 – Basic Binary Tree

Overview

Sorry for getting this to you late. As a result, I made the assignment far simpler than I originally planned.

Anyway, for this assignment, you are going to a very basic tree with a minimal interface. You don't have to balance the tree. Not yet, at least. We'll cover that pretty soon.



Part 1: Node Class

Overview

In recursively defined structures, like trees, all the coding (and complexity) is found in the recursive structure itself. In the case of trees, the node will contain the vast amount of the logic and behavior.

You are going to create a very, very basic node class. Make sure to come up with a nice, well-documented, and well-written solution. We will build upon it, later, to create more advanced tree structures.

Interface

Create a very basic node class. All this requires is a left and right link (to another node) and a generic data field. It should also have a few constructors. In particular, you need one that will create a node with links to two other nodes.

`public class Node`		
	`Node()`	Constructor
	`Node(Object, Node, Node)`	Constructor that assigns the data, left and right nodes.
`Node`	`left`	
`Node`	`right`	
`Object`	`data`	The value that the node contains.
`void`	`printValues()`	Prints the contents of the tree using an **infix tree traversal**. They should be sent to standard out with spaces between each value. Feel free to redirect the stream if you like.
`void`	`printTree(int indent)`	Unlike the other print method, this one will print the structure of the tree. One node will be printed per line. You should use prefix tree traversal. Feel free to redirect the stream if you like.

Printing the Tree

To print the tree, you will use the same recursion logic as printing the values. However, rather than an infix traversal, this will use prefix. Don't worry, it is not hard. If you wrote the other print method already, this will be easy.

```
Method printTree(int indent)
    Print spaces for indent
    Print Node.data & newline

    left.printTree(indent + 1)
    right.printTree(indent + 1)
End Method
```

The binary tree, in the lecture nodes, would create the following output. I am using spaces followed by a string of "**--+**" for readability. You can use spaces, dashes, etc… You can use any factor you like (2, 3, etc…).

```
+-- 1
  +-- 3
    +-- 42
    +-- 26
  +-- 5
    +-- 7
      +-- 9
      +-- 12
    +-- 74
```

Pretty cool, huh? You can see the tree.

Part 2: BinaryTree Class

Overview

While all the major logic will be found in the nodes, we need to have a single, centralized class. The purpose of the BinaryTree class is to start recursion, store some global attributes (for future versions), and much more.

Interface

The tree class needs to store the root node.

public class BinaryTree		
	BinaryTree()	Constructor.
	BinaryTree(Node root)	Constructor.
Node	**root**	
String	**about()**	Returns text about you – the author of this class.
void	**printValues()**	Starts recursion from the root node.
void	**printTree()**	Starts recursion from the root node.

Part 3: Testing

Once you have finished your code, you need to test it using some good test data.

Turn in

- The source code.
- The main program that runs the tests. These tests should be hard-coded. I will run your programs and make modifications to test them.
- Output generated by your tests.