Programming Assignment IV
COP3502H

Due: 3/31/07 11:59pm

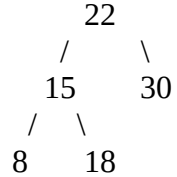
The CEO of the Computer Sciences Corporation, Dr. Link Queue has just learned that G.M. Adelson-Velsky and E.M. Landis will be visiting the company in two weeks. They want to invest in the company, so Dr. Queue wants to give them a good demo. Thus, he wants you to build an interactive AVL tree tool. In this assignment, you will write an interactive balancing routine for binary trees that utilizes the AVL algorithm.

Requirements

You will develop an interactive program that will let users insert and delete nodes in a binary search tree (BST) and also run the AVL algorithm. The program should also output the contents of the tree after each operation. Thus, you should provide a menu driven interface that lets users

1. Create an empty tree (and delete the old one if it exists).
2. Insert a node into the BST.
3. Delete a node from the BST.
4. Find the balancing factor for the root of the tree.
5. Balance the BST with AVL.

For operations 2,3, and 5, you should output the tree level, the nodes in that level, and the children each node points to. For example, if I have a binary tree that looks like the following:



The output would be

Level 0: Node: 22 – Left Child Node: 15 – Right Child Node: 30
Level 1: Node: 15 – Left Child Node 8 – Right Child Node: 18
Node: 30 – Leaf Node
Level 2: Node: 8 – Leaf Node
Node: 18 – Leaf Node

This output makes it very easy for you to draw out the trees on paper.

Approach

Use the following data structure for your BSTs

```
struct tree_node {
     int data;
     struct tree_node *left;
     struct tree_node *right;
     int height;
};
```

You should need to write 5 primary functions and some helper functions for this assignment.

```
struct tree_node * createBST();

struct tree_node* insertBSTNode(struct tree_node *root, int
data);

struct tree_node* deleteBSTNode(struct tree_node *root, int
data);

int findBalanceFactor (struct tree_node *root);

struct tree_node* balanceTreeAVL(struct tree_node *root);
```

The helper functions should be used as part of the `balanceTreeAVL` function to help determine which one of the 4 cases the BST could be in (RR, LL, RL, LR). I would also strongly encourage you to write 4 helper functions that do the rotations for the each case. This will make your code easier to read and debug. Also, make use of the notes for implementing the insert and deletion routines.

Also note, your balancing algorithm only needs to work for balancing factors up to 2 or -2.

Extra Credit

1. Write a routine to print out the BSTs in a pretty format.

2. Analyze the running time of your `balanceTreeAVL` algorithm.

Documentation

You must document your routines. Each function should be commented with what the function does, its input parameters, and what it returns. Comments are very important and will account for 20% of your grade.

Testing

You must test your program and ensure that it works. You should provide output files that show you can insert and delete nodes from the tree and that the AVL algorithm does the correct balancing.

Deliverables

You must submit all source files in addition to your test files to Sean and Dr. LaViola by the due date. Also, include a README file explaining how you tested your code and what problems you encountered. Note the code must be able to be compiled and executed on the Olympus unix system.

Grading

Grading will be based on the following:

60% correct functionality
20% documentation
20% testing