CISC220 Project 1

Project due November 10. You may work in groups of 2-4 people, but you may not work on this project alone.

You may not help other people debug their programs, except in the very limited way described on the class website. You may not use any code written by others, whether they are students or not. The TA and the instructor are available for office hours, and by appointment if you can't make hours because of a class conflict.

Overview

Our goal for this project is to visualize the balancing of a binary tree. In order to do this, we will create a fractal image that represents the underlying structure of our binary tree. Fractals are defined as a geometric figure that can be split into parts, where each part is approximately a reduced-size copy of the whole (see http://en.wikipedia.org/wiki/Fractal for details and examples).

Additional help is available for the OpenGL toolkit (GLUT) on the course website under project 1.

Assignment

1. **Part 1**

   A basic FractalGenerator is available on the course website under project 1. This generator creates a fractal of squares that are nested on two ends of the square. The code that draws this picture does not use any BinaryTree structure; instead it has a series of calls to the drawElement function that draws a picture representing a balanced BinaryTree of depth 4 (you should see 1 + 2 + 4 + 8 = 15 squares in the picture).

   Come up with your own fractal shape (not a square) and modify the given `drawElement` function. You should be able to handle all four arguments for size, center x/y, and rotation for your new fractal shape.

2. **Part 2**

   - Modify the GraphicsScene class to include a BinaryTree structure as one of its data elements.
   - Modify the main method in your FractalGenerator class so that you add new nodes to your BinaryTree structure, creating a balanced BinaryTree (using your code for add(int key) from Homework 3). For example, calling add(10) add(5) add(2) add(7) add(15) add(12) add(17) will create a balanced binary tree of depth 3.
   - Remove the existing calls to `drawElement` from `display`. Replace them with a pre-order traversal of the SCENE.binaryTree, with the output of each node being a call to your `drawElement` function.

3. **Part 3**

   Modify the update function to randomly insert one value into your BinaryTree every time update is called. If you implemented Parts 1 and 2 correctly your fractal image should gradually change to reflect an additional node added to the tree each second.

4. **Part 4**

- Modify your BinaryTreeNode class to allow for AVL-Tree balancing. Add an integer field called `balance` and the following three functions:

```
/**
 * Rotates the subtree rooted at this node to the right.
 * For example, a rotateRight on node 10:
 *
 *          10                        5
 *         /  \                     /  \
 *        5   15                   3   10
 *       / \         --->         / \  / \
 *      3   7                    1  4 7  15
 *     / \
 *    1   4
 *
 * Returns a pointer to the new root of the subtree, in this
 * example it would be a pointer to node 5.
 */
BinaryTreeNode* rotateRight();

/**
 * Rotates the subtree rooted at this node to the left.
 * For example, a rotateLeft on node 10:
 *
 *          10              10              7
 *         /  \            /  \           /  \
 *        5   15          7   15         5   10
 *       / \      --->   / \     --->   / \  / \
 *      3   7           5   8          3  6 8  15
 *         / \         / \
 *        6   8       3   6
 *
 * Returns a pointer to the new root of the subtree, in this
 * example it would be a pointer to node 7.
 */
BinaryTreeNode* rotateLeft();
```

```
/**
 * Rebalances the subtree rooted at this node. Checks the balance
 * field to determine the correct case and rotation to perform:
 *   LeftLeft case (balance=2 and left child balance = 1)
 *   LeftRight case (balance=2 and left child balance = -1)
 *   RightLeft case (balance=-2 and right child balance = 1)
 *   RightRight case (balance=-2 and right child balance = -1)
 *
 * This function should also update the balance field of this node
 *   and the balance fields of affected children.
 */
BinaryTreeNode* rebalance();
```

- Test your rotateRight, rotateLeft, and rebalance functions by creating the example trees above and manually setting the balance fields on each node.

5. **Part 5**

   Modify your insert function so that it sets the initial value for `balance` to zero for the new node. The insert function now needs to update the `balance` field every time one of its left or right subtrees increases in depth. The increase in depth should be detected the way we discussed in class (by comparing the left or right child's previous and current balance field) and not by calling a depth function on its children. If the insert function detects that a child's current balance field is equal to -2 or 2 then it should call the rebalance function on that child. Don't forget to update child pointers if the rebalance function returns a different root for the subtree.

Submission of the final Project 1 is due November 10th at midnight. Additional Parts will be added to the Project as we cover material in class. You do not have additional Homework assignments, so make sure to keep on track by completing Parts 1 through 5 by November 3rd.