Due May 9, 11.59pm

In the final part of this project, you will use your current implementation to maintain a balanced binary search tree, specifically as an AVL tree. We will only do a limited form of the AVL tree by supporting only insertions; this would require at most one rotation operation after each insertion of a new key.

Implementation Details The AVL tree implementation for insert operations can be integrated into the insert operation. Remember the insert operation (from the BST class) is as follows:

```
private BSTElement<Key,E> inserthelp(BSTElement<Key,E> rt, Key k, E e) {
   if (rt == null) {
      BSTElement<Key, E> new_node = new BSTElement<Key, E>(k, e);
      return new_node;
   }
   if (rt.getKey().compareTo(k) > 0)
      rt.setLeft(inserthelp(rt.getLeft(), k, e));
   else
      rt.setRight(inserthelp(rt.getRight(), k, e));
                  // note that the reference to this node is returned,
                  // which is critical to the implementation  as the
                  // tree structure can change to maintain balance
   return rt;
```

In part 2 of the project you had augmented this module to also maintain the height and balance factors at each node. Thus, you can further modify the insert operation to maintain an AVL style balanced tree as follows (only psuedo code provided)

```
private BSTElement<Key,E> inserthelp(BSTElement<Key,E> rt, Key k, E e) {
   if (rt == null) {
      BSTElement<Key, E> new_node = new BSTElement<Key, E>(k, e);
                     // must ensure the height and balance factors
                     // of the leafnode is set correctly
      return new_node;
   }
   if (rt.getKey().compareTo(k) > 0) {
      rt.setLeft(inserthelp(rt.getLeft(), k, e));
   }
   else  {
      rt.setRight(inserthelp(rt.getRight(), k, e));
   }

                  // next compute the height and balance factor of this node
   rt.setValue( ......... )

                  // next check the balance factor (note that we are
                  // traversing bottom up

   if (rt_s_balance_factor == 2 || rt_s balance_factor == -2) {
                  // tree is unbalanced, must rotate
      L, R, LR or RL <------ findRotationType (rt)

      Perform the rotation (this usually 3-4 pointer changes, note
            root of subtree changes, hence the need to return the node (see
            below). LR and RL are calls to L and R
            // recompute and update the height of the entire subtree at
            // this node(use the function from part 2 of the project)
```

```
        UpdateHeight (rt);
    }
        // note that the reference to this node is returned,
        // which is critical to the implementation  as the
        // tree structure can change to maintain balance
    return rt;
}
```

Thus, the modified insert algorithm inserts the new key as before, but when returning from the recursion, it updates the height and balance factors and if needed, performs rotations, again updates the height and balance factors of the rotated subtree and continues upward.
[Requirements/Evaluation.]

1. **This project can be done in 2 person teams**; peer evaluation will be performed to indicate the role of each member.
2. You will use both part 1 and part 2 of the project and use the earthquake dataset (for debugging puposes, start with 10 earthquakes and then increase tree size).
3. All needed functions can be private to the BST class.
4. Display both the balance factors and the earthquake data (concatenate and format so the information is readable) on mouseover.
5. Evaluation: By Interactive Demo. You will test your implementation by
   (a) Inserting the earthquake records in the given order; visualize the tree.
   (b) You will sort the earthquakes by magnitude in ascending order and insert them into the tree. Demonstrate intially with just a few quake records to verify the rotations are working; then increase the number of records to 50, 100, 200.
   (c) Repeat with descending order.

Grading Rubric.

1. Left Rotation (2 point)
2. Right Rotation (2 point)
3. Left-Right Rotation (2 point)
4. Right-Left Rotation (2 point)
5. Documentation(1 point)
6. Survey (1 point)