Topic: Binary Search Trees

Activity Guidelines

Group Size: 3

Method of Assigning Students: Count the number of students in the class, divide by 3, count off from 1 to the quotient, and group identical numbers.

Materials:

- ✓ Handout (one copy per group) with questions to be answered at the end of the session

Roles:

Coordinator/Leader: Clarifies goals and objectives, allocates roles for each team member and divides the tasks within the group.

Monitor/Evaluator: Person designed to evaluate the different ideas to approach the problem and make an accurate judgment of the most beneficial option.

Implementer: Person in charge to transform discussions and ideas into a technical solution for the given problem.

Individual Accountability: Each team member gets assigned a specific role in order to ensure every student within a team participates and contributes to reach a solution for each problem presented in the activity.

Activity Summary

1) Every team is required to implement 4 different methods using binary search trees that perform the following functions:

   a. Print the elements of the binary search tree in-order.
   b. Add an element to the binary search tree.
   c. Search for an element in the binary search tree.
   d. Obtain the minimum value contained in the binary search tree.

Elementary Data Structures

Peer Session

Binary Search Trees

1. Add a method print to the TreeNode class that prints the elements of the tree, separated by spaces.

   – A node's left subtree should be printed before it, and its right subtree should be printed after it.

```
private void print(TreeNode root) {
        // (base case is implicitly to do nothing on null)
        if (root != null) {
            // recursive case: print left, center, right
            print(root.left);
            System.out.print(root.data + " ");
            print(root.right);
        }
    }
}
```

2. Add a method add to the TreeNode class that adds a given integer value to the tree. Assume that the elements of the TreeNode constitute a legal binary search tree, and add the new value in the appropriate place to maintain ordering.

```
private void add(TreeNode root, int value) {
    if (root.data > value) {
        if (root.left == null) {
            root.left = new TreeNode(value);
        } else {
            add(root.left, value);
        }
    } else if (root.data < value) {
        if (root.right == null) {
            root.right = new TreeNode(value);
        } else {
            add(root.right, value);
        }
    }
    // else root.data == value; a duplicate (don't add)
}
```

3. Add a method contains to the TreeNode class that searches the tree for a given integer, returning true if found.

```
// Returns whether this tree contains the given integer.

private boolean contains(TreeNode root, int value) {
    if (root == null) {
        return false;
    } else if (root.data == value) {
        return true;
    } else if (root.data > value) {
        return contains(root.left, value);
    } else {   // root.data < value
        return contains(root.right, value);
    }
}
```

4. Add a method getMin to the TreeNode class that returns the minimum integer value from the tree. Assume that the elements of the TreeNode constitute a legal binary search tree. Throw a NoSuchElementException if the tree is empty.

```
private int getMin(TreeNode root) {
    if (root.left == null) {
        return root.data;
    } else {
        return getMin(root.left);
    }
}
```