CSCI 1933 Project 5

Binary Search Trees

.

Due Date: December 14, 2017 on or before 11:59 pm

The fifth and final project will focus on binary search trees. A binary search tree is a binary tree which maintains the restriction that each node has a key that is larger all keys in the left subtree and smaller than all keys in the right subtree. This restriction is known as the Binary Search Tree Principle.

As discussed in lecture, you may work with one partner to complete this assignment (max team size = 2). If you choose to work as a team, please only turn in one copy of your assignment. At the top of your class definition that implements your main program, include in the comments both of your names and student IDs. In doing so, you are attesting to the fact that both of you have contributed substantially to completion of the project and that both of you understand all code that has been implemented.

For this project, you will be provided three files: `Node.java`, `BinaryTree.java`, and `BinaryTreeTest.java`. `Node.java` contains a baseline implementation of a binary tree node. `BinaryTree.java` contains a code skeleton that you will need to fill out. Both `Node.java` and `BinaryTree.java` are implemented using generics. `BinaryTreeTest.java` contains JUnit tests for each of the methods you are required to implement. Please note that the JUnit tests are only provided for convenience. They may not test all edge cases and your performance on the test cases is not necessarily indicative of your final grade on the project.

1 Adding

This section requires you to complete the following method within `BinaryTree.java`:

```
public void add(K key, V value) {

}
```

The `add` method should add a new node to the existing tree with the key and value passed into the method. If the key already exists in the binary tree, you should update its value to reflect the value passed into the method. If the tree is an empty tree, a root should be created with the key and value passed into the method.

2 Finding

This section requires you to complete the following method within `BinaryTree.java`:

```
public V find(K key) {
    return null;
}
```

The `find` method should return the value associated with the key passed into the method. If the key does not exist within the binary tree, `null` should be returned.

3 Flattening

This section requires you to complete the following method within `BinaryTree.java`:

```
public V[] flatten() {
    return (V[]) new Object[0];
}
```

The `flatten` function should return an array of all of the values in a binary tree, ordered by key. The length of the array should be equal to the number of elements in the tree and duplicate values should be included.

4 Removing

This section requires you to complete the following method within `BinaryTree.java`:

```
public void remove(K key) {

}
```

The `remove` method should remove the key from the binary tree and modify the tree accordingly to maintain the Binary Search Tree Principle. If the key does not exist in the binary tree, no nodes should be removed

5 Determining Subtrees

This section requires you to complete the following method within `BinaryTree.java`:

```
public boolean containsSubtree(BinaryTree<K, V> other) {
    return false;
}
```

The `containsSubtree` function should return whether the tree passed into the method is a subtree of the tree that it is called from. If the subtree passed into the function is null, `containsSubtree` should return `true`.