CSCI 1933 Project 5 / Lab 13

Binary Search Trees

Due Date: May 2, 2022 before 11:55pm

Total points: 50 points

Note: This project is *optional*. If you complete it, it will be averaged in with other projects to compute your project grade for the course. If you choose not to complete Project 5, your project grade will be based only on Projects 1-4. This project is worth 50 points, whereas Projects 1-4 are worth 100 points each. Completion of the project is highly encouraged as it will reinforce concepts covered in the last two weeks of the course, which will be tested on Midterm 3.

Lab 13 will consist of work time for this project. Your lab grade will be based solely on attendance. All remaining steps must be completed before the due date above. You may work with one partner to complete this assignment. If you choose to work with a partner, 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.

Introduction

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 than 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.*

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. You will need to implement an `add`, `find`, and `flatten` method. You should also be familiar with the `remove` method, but we have provided you with this code. 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. Although the tests are useful for debugging, 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. Refer to the JUnit helper document on how to run JUnit tests.

1 Adding

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

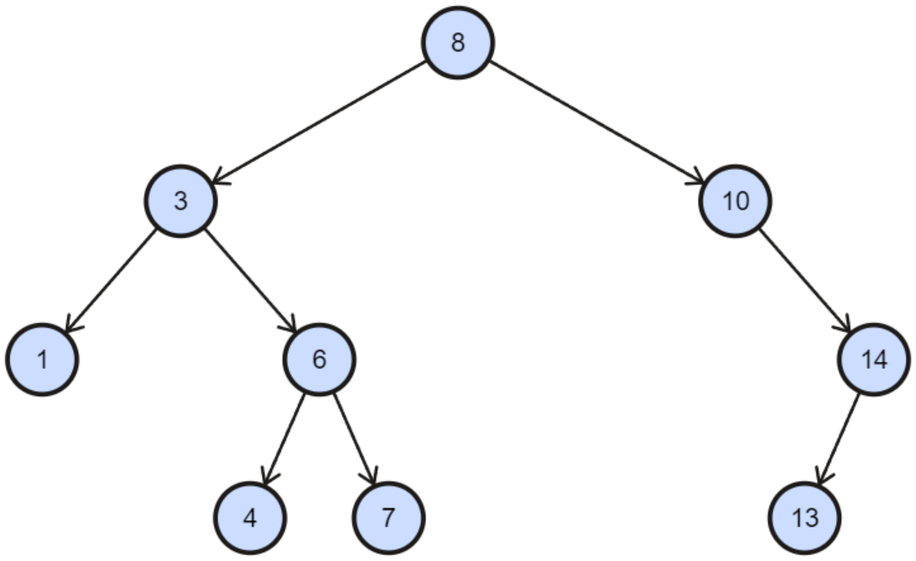

Figure 1: A Binary Search Tree of size 9 and depth 3.

```
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 rather than adding a new node. If the tree is an empty tree, a root should be created with the key and value passed into the method. You may implement your `add` method in such a way that any new nodes inserted are leaf nodes.

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`:

```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 Determining Subtrees

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

```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`.

Hint: As with many solutions to coding problems, a helper function is useful/necessary for this method.

5 Submission

Once you've completed your assignment, submit your .java files on Canvas. Also, be sure to include a README.txt file with your submission that contains the following information:

- Group member's names and x500s
- Contributions of each partner (if working with a partner)
- How to compile and run your program
- Any assumptions
- Additional features that you implemented (if applicable)
- Any known bugs or defects in the program
- Any outside sources (aside from course resources) consulted for ideas used in the project, in the format:

  - idea1: source

  - idea2: source

  - idea3: source

- Include the statement: "I certify that the information contained in this README file is complete and accurate. I have both read and followed the course policies in the 'Academic Integrity - Course Policy' section of the course syllabus." and type your name(s) underneath