CSCI 1933 Lab 12

Binary Trees

Introduction

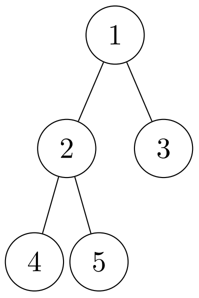
This lab will focus on the binary trees data structure. You will be provided two files: `Node.java` and `BinaryTree.java`. `Node.java` contains a baseline implementation of a binary tree node. `BinaryTree.java` contains a code skeleton that you will need to fill out. Both `Node.java` and `BinaryTree.java` are implemented using generics that extend `Comparable`.

Rules

You may check off your milestones during online office hours or lab time as normal, or you may email your completed code and screenshots of your output for each milestone to your lab TAs. **Please send your emails only to your lab TAs**. You may work individually or with *one* partner. You have until the following Monday at 11:55pm to check off each milestone. If you worked with a partner, send in one email per pair and include your partner's name. You will only receive credit for the milestones you have checked off by a TA. There is nothing to submit to Canvas for this lab.

1 Traversing

Unlike linear data structures, binary trees can be traversed in several different ways. The most common way to traverse them are in either Inorder, Preorder, or Postorder order. Note the recursive nature of each traversal method.



The **Inorder traversal** method first traverses the left branch, then visits the node, then traverses the right branch. In the above tree, an inorder traversal would visit the nodes in order 4, 2, 5, 1, 3.

The **Preorder traversal** method first visits the root, then traverses the left branch, then traverses the right branch. In the above tree, a preorder traversal would visit the nodes in order 1, 2, 4, 5, 3.

The **Postorder traversal** method first traverses the left branch, then the right branch, and then visits the root. In the above tree, a postorder traversal would visit the nodes in order 4, 5, 2, 3, 1.

Milestone 1:
Fill out the `printInorderHelper`, `printPreorderHelper`, and `printPostOrderHelper` functions in `BinaryTree` class and test it using the provided binary tree.

2 Flattening

This section requires your to complete the `flatten` method within `BinaryTree`.

```
public V[] flatten();
```

This function should return an array of all the values in a binary tree in ascending order. The length of the array should be equal to the number of elements in the tree and duplicate values should be included.

You should do this by first adding all the values of the binary tree to the array using one of the traversal algorithms discussed in milestone (1) of the lab, and then sort it using one of the sorting algorithms learned in this class (ex: bubble sort, insertion sort, ect...). You may need to create a recursive helper function to get all the elements of the tree into an array.

Milestone 2:
Fill out the `flatten` method and show it works by un-commenting out the tests in `BinaryTree`.

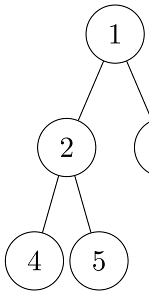
3 Determining Subtrees

This section requires you to complete the `containsSubtree` method within `BinaryTree`.

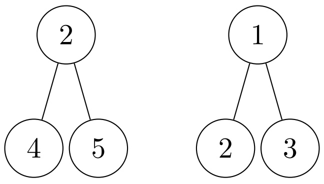
```
public boolean containsSubtree(BinaryTree<V> other);
```

This 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`, the function should return `true`.

For example, for the following tree:



Then the following left tree is considered a subtree of the above tree but the right tree is not a subtree:



Milestone 3:
Fill out the `containsSubtree` and show it works by un-commenting out the tests in `BinaryTree`