Lab 4

Steve Boucher

November 19, 2022

Linking Lists

1 Introduction

The goal of lab 3 was to code multiple Graphs and a Binary Search tree. The graphs required for this lab were A linked list, a matrix, and an adjacency list while also being required to perform an in-order traversal. For my initial setup, I created a main.java program while importing some previous code such as my file reader, My CWO (Completed, WorkInProgress, And Other) text file, along with the magic items text file, three Linked List(BST.java, Graph.java, NeighborList.java), and Two Node Classes(Components.java and Node.java). New this time around were two text files Commands.txt and magicitems-find-in-bst.txt

2 Part 1?

I made a few separate versions of a Linked List throughout the duration of this lab as the requirements for the graph list were quite different from the BST, easily seen from their node constructors

```
public class Node {
   int vertex;
   boolean processed;
   NeighborList neighbors = new NeighborList();
   Node nextNeighbor;
   Node next;
}
```

and

```
public class Component {
   String data;
   Component left;
   Component right;
   Boolean attunded;
}
```

The component class requires two branches, a left, and a right branch in order to sort different values to each side as opposed to the Node class. The Node class contains one possible branch but contains a separate list for neighbors.

2.1 Part One

I could not figure out how to import the commands from graphs.txt. I did write the commands in the file out as simple graphname(value) and edges were (vertice 1, vertice 2) as shown below

```
   Graph full8 = new Graph();
   for (int i = 1; i < 9; i++) {
     full8.insert(i);
   }
   full8.edge(2, 1);
```

Although it was not the intended way to read the commands, I hope by the next lab I will have figured out how to read commands. Getting into the actual creation of the three graphs, I had the easiest time coding the Linked List Graph. A brief 16 lines of code was all it took to ready every item into the Graph

```
public void insert(int data) {
    Node node = new Node();
    node.vertex = data;
    node.neighbors = new NeighborList();
    node.next = null;
    node.processed = false;

     if (head == null) {
         head = node;
     } else {
         Node n = head;
         while (n.next != null) {
             n = n.next;
         }
         n.next = node;
     }
}
```

It took me a decent amount of time to correct code the adjacency graph, as I have to learn and figure out how to navigate the Neighbors array which lives inside each point on the graph. My biggest issue making a graph was the hours of trial and error it took to correctly arrange and execute the Matrix.

I was unable to code the Depth First Traversal and Breath First Traversal. These Traversals are two different means for exploring a graph, with slightly different tactics. A depth-first traversal is an act of exploring/searching a graph for an item by looking as deep, or farthest away from the root as possible horizontally. This would have a run time of Oh(M + N), as it does take linear time to calculate the data set, we need to factor in an extra amount of time-based on the number of edges.

A breath First Traversal is done in a similar way, but instead of exploring the data structure downwards, it's explored outwards, crawling and analyzing in layers before moving to the next set of leaves in the tree. This although a different approach, has the same run time as DepthFirst Traversal for similar reasons.

2.1.1 Part 2

The initial coding of a binary search tree was not too bad. Modifications were made to my Array List class so we can have two outputs for the new nodes. It was simply filling up a linked list again, but if the data was lower than the current point it would trail to the left, and higher data would trail to the right.

```
public Component insert(Component component, String data) {
    if (component == null) {
        return createNewComponent(data);
    }
    if (component.data.compareToIgnoreCase(data) <= 0) {
        component.right = insert(component.right, data);
        System.out.println("R");
    } else if (component.data.compareToIgnoreCase(data) > 0) {
        component.left = insert(component.left, data);
        System.out.println("L");
    }
```

```
        return component;
    }
```

.

2.1.2 Conclusion

There are a few things that I absolutely wish I did better, such as reading the commands from the commands files and achieving the BST and DST. But on the other hand, I am very happy with myself that I was able to adapt my previous existing code to solve a new problem.