CMPT 435 - Fall 2021 - Daniel Bilotto

Assignment Four – LaTeX Sorts

Daniel Bilotto
danielbilotto1@marist.edu

November 20, 2021

1 Compute Class

This class is the main class for the lab. It includes methods for both linear and binary search. As well as the method ”main”.

2 Main Method

So this project was hard, and some of it doesn't work. But I did as much as I could and will run through the issues and the good things. So starting with the bad. The biggest issue I had was reading in the file. that was the number one issue. Lines 11 to 94 is my attempt to read in a file which did not work so I commented it out and moved on. But quickly trying to explain what I did. It will read through the file line by line and if the line contains ”new graph” then it will initialize a new graph, if it contains ”add vertex” it will find the int of that line and increment a value called vcount (vertex count) and if it contains ”add edge” it will find the two ints in that line and set edge 1 and edge 2 to those numbers respectfully. Because it didn't work I manually added the first graph only to show that my matrix and adjacent list works. You can see that in lines 99 to 123. On to better news my binary search trees kinda work. I can verify that every item is being added. But my issue is that some items aren't going to the right spot and I know I don't think I'm printing the path out correctly. More on that later. Overall this assignment was tough and I thought I was figuring out but the pieces never really fell into place.

```
1 import java.io.File;
2 import java.util.Scanner;
3
4 public class Compute {
```

```
// Create a new keyboard Scanner object.
  static Scanner keyboard = new Scanner(System.in);


public static void main(String[] args)
{
  /*
  int graphSize = 375;
  String fileName = null;

    try
    {
      fileName = "graphs.txt";
      String gline = null;
      File theFile = new File(fileName);
      Scanner ginput = new Scanner(theFile);

    for (int i = 0; i < graphSize; i++)
    {
      if (gline.contains("--"))
      {
        //donothing
      }

      else if (gline.contains("new graph"))
      {
        int vernum = 0;
        int firstver = 0;
        boolean first = true;
        int vcount = 0;
        int edge1 = 0;
        int edge2 = 0;

        i++;


        while (gline.contains("add vertex"))
        {
          String vid = gline.replaceAll("[^\\d-]", "");
          vernum = Integer.parseInt(vid);


        if (first)
        {
          firstver = vernum;
          first = false;
        }


        vcount++;
        i++;
        }//while

        if (firstver == 1)
        {
          vcount++;
        }//if
```

```

          GraphMatrix g = new GraphMatrix(vcount);

          while (gline.contains("add edge"))
          {
            String[] vstring;
            vstring = gline.replaceAll("[^\\d-]", "").split("-");

            edge1 = Integer.parseInt(vstring[0]);
            edge2 = Integer.parseInt(vstring[1]);

            g.addEdge(edge1 - 1, edge2 - 1);

            if(i < graphSize - 1) {
              i++;
            } else {
              break;
            }

          }//while
        }
      }



        ginput.close();
        keyboard.close();
      }//try

      catch(Exception ex)
      {
        System.out.println("Oops, something went wrong!");
      }//catch

      */


    GraphMatrix matrix = new GraphMatrix(7);
    AdjList adj = new AdjList(7);
    adj.addEdge(0, 1);
    adj.addEdge(0, 4);
    adj.addEdge(0, 5);
    adj.addEdge(1, 2);
    adj.addEdge(1, 4);
    adj.addEdge(1, 5);
    adj.addEdge(2, 3);
    adj.addEdge(3, 4);
    adj.addEdge(4, 5);
    adj.addEdge(4, 6);
    adj.addEdge(5, 6);

    matrix.addEdge(0, 1);
    matrix.addEdge(0, 4);
    matrix.addEdge(0, 5);
    matrix.addEdge(1, 2);
    matrix.addEdge(1, 4);
    matrix.addEdge(1, 5);
```

```
  matrix.addEdge(2, 3);
  matrix.addEdge(3, 4);
  matrix.addEdge(4, 5);
  matrix.addEdge(4, 6);
  matrix.addEdge(5, 6);

  adj.print();
  matrix.print();
  searchTree();



}//main


public static void searchTree()
  {
  String fileName = null;
    String line = null;
    BST tree = new BST();
    int size = 666;
    int size2 = 42;
    String fileline = null;

    try
    {
      fileName = "magicitems.txt";
      File theFile = new File(fileName);
      Scanner input = new Scanner(theFile);

    for (int i = 0; i<size; i++)
      {
      NodeTree item = new NodeTree();
      line = input.nextLine();
      item.setKey(line);
      System.out.println(item.getKey());
      tree.insert(tree, item);


      }//for


    //tree.printTree();
    //System.out.println("----------------------------------------------");

    Scanner fileinput = new Scanner(new File("src/text.txt"));
    for (int k = 0; k < size2; k++)
    {
      fileline = fileinput.nextLine();
      tree.search(fileline);
    }

    //tree.search("");


      input.close();
      keyboard.close();
    }//try



    catch(Exception ex)
    {
      System.out.println("Oops, something went wrong!");
    }//catch


   }

 }//Compute
```

3 Matrix Method

This is my Matrix class, which makes a matrix (I know shocking). This was also pretty difficult. But basically you have a array list of integers and a number of vertexes. You use the int ver in this case to tell the method how many vertices there are and then the addEdge method adds them by setting the two numbers given to it a connection. For example if one is connected to 3 that mean 3 has to be connected to 1 and you can see that in lines 17 to 21. the last method is a print method which simply has a nested for loop that loops for the size of the arraylist and prints out the values. This was the first time I have used array lists so the print statement was heavily drawn from the internet (more on this below).

```
 1 import java.util.ArrayList;
 2 public class GraphMatrix
 3 {
 4  ArrayList<ArrayList<Integer>> adjace;
 5   int ver;
 6 
 7  //Initialize
 8 
 9  public GraphMatrix(int vertex)
10  {
11   ver = vertex;
12   adjace = new ArrayList<ArrayList<Integer>>(ver);
13     for (int i = 0; i < ver; i++)
14      adjace.add(new ArrayList<Integer>());
15  }
16 
17  public void addEdge(int source, int dest)
18  {
19   adjace.get(source).add(dest);
20   adjace.get(dest).add(source);
21  }
22 
23 
24 
25  public void print()
```

```
    {
        for (int i = 0; i < adjace.size(); i++) {
            System.out.println("Adjacency list of " + i);
            for (int j = 0; j < adjace.get(i).size(); j++) {
                System.out.print(adjace.get(i).get(j) + " ");
            }
            System.out.println();
        }
    }
}
```

4 Adj List Method

This is the adjacency list method basically what it does is pretty similar to the matrix method. But it uses java's own linked list (I had no idea this even existed until i looked it up). And then it uses that link list to connect the edges.

```
import java.util.LinkedList;
public class AdjList
{
  int ver;
  LinkedList<Integer> list[];


  public AdjList(int vertex)
  {
     ver = vertex;
     list = new LinkedList[vertex];
     for (int i = 0; i <vertex ; i++) {
     list[i] = new LinkedList<>();
     }
  }


  public void addEdge(int source, int dest)
  {
    list[source].addFirst(dest);
  }

  public void print(){
     for (int i = 0; i < ver ; i++)
     {
       if(list[i].size()>0)
       {
         System.out.print("Vertex " + i + " is connected to: ");
         for (int j = 0; j < list[i].size(); j++)
         {
           System.out.print(list[i].get(j) + " ");
         }
         System.out.println();
       }
     }
  }
}
```

5 BST List Method

Ok this is the Binary search tree method. it does has the methods, print, add and search. Starting with print; if the item you pass it is not null then execute the print statement of the items left and print it, then execute it again but with the item to the right. Next is the insert statement, it gets a tree and a node passed to it and basically what it does is has a trailing node and a current node and when curr is not null compare it with the item you are adding to the tree. If the item comes before it then look left if not look right. Then if the trail is not null then compare the item to that and like before if it comes before add it to the left side, if not add it to the right side. As for the search method, if the tree is null or there is only one item in the tree/is the root then just return the item passed. if it isnt then compare the item you are looking for with the tree and navigate correctly.

```

public class BST
{
  NodeTree root = null;


  public BST()
  {
    root = null;

  }//BST

  public void printTree()
  {
    printTree(root);
  }

  public static void printTree(NodeTree item)
  {
    if (item != null)
    {
      printTree(item.left);
      System.out.println(item.getKey());
      printTree(item.right);
    }

  }//print

  public void insert(BST tree, NodeTree item)
  {
    NodeTree trail = null;
    NodeTree curr = tree.root;



    while (curr != null)
    {

      trail = curr;
      if (item.getKey().compareToIgnoreCase(curr.getKey()) < 0)
    {
      curr = curr.left;
      System.out.println("L");
    }


    else
    {
      curr = curr.right;
      System.out.println("R");
    }

  }//while

  item.parent = trail;

  if (trail == null)
  {
    tree.root = item;
  }
  else if (item.getKey().compareToIgnoreCase(trail.getKey()) < 0)
  {
    trail.left = item;
    System.out.println("L");
  }
  else
  {
    trail.right = item;
    System.out.println("R");
  }


}//insert

public void search(String want)
{
  search(root, want);
}

public static NodeTree search (NodeTree item, String want)
{
  if (item == null || want == item.getKey())
    return item;


  if (want.compareTo(item.getKey()) < 0)
  {
    System.out.println("L");
    return search(item.left, want);

  }
  else
  {
    System.out.println("R");
    return search(item.right, want);
  }
}
```

```



}//BST
```

6 Node for tree List Method

This is the tree node class which as the name suggests is the node used for the BST. Super simple, it has a left, a right and a parent as well as a key(data). lines 10 - 16 are initializing the node and then you have a getter and setter.

```
 1
 2
 3 public class NodeTree
 4 {
 5   String myKey;
 6   NodeTree left;
 7   NodeTree right;
 8   NodeTree parent;
 9
10   NodeTree()
11   {
12     myKey = null;
13     left = null;
14     right = null;
15     parent = null;
16   }//nodeTree
17
18   NodeTree(String item)
19   {
20     myKey = item;
21     left = null;
22     right = null;
23     parent = null;
24   }
25
26
27   public String getKey()
28   {
29     return myKey;
30   }//get
31
32   public void setKey(String newKey)
33   {
34     myKey = newKey;
35   }//set
36 }//NodeTree
```

7 Final Thoughts

As you may notice I do not have depth first or breadth first triversal. And that is because I could not get the read file to work which means in turn I can not get those to work either. Below is how far I got with that, it's a modified node class (which looking back I think I could of just used this one for the tree node as well) But I was not able to get it to work over all. I think most of it was because I didn't know that much about array lists nor could I get my head around a smart was to read the file in. Which lead to me having to look a lot of things up wither that be the textbook or online resources.

```java


import java.util.ArrayList;

public class NodeBilotto
{
  public String myData;
  public NodeBilotto myNext;
  public NodeBilotto prev;

  public NodeBilotto ()
  {
    myData = null;
    myNext = null;
    prev = null;
  }//NodeBilotto

  public NodeBilotto (String newData)
  {
    myData = newData;
    myNext = null;
  }//NodeBilotto

  public String getData()
  {
    return myData;
  }//getData

  public void setData (String newData)
  {
    myData = newData;
  }//setData

  public NodeBilotto getNext()
  {
    return myNext;
  }//getNext

  public void setNext(NodeBilotto newNext)
  {
    myNext = newNext;
  }//setNext

  //graph stuff

  public int num;
  public boolean isProccessed;
  public boolean needsProccessed;
  public ArrayList<NodeBilotto> neighbors = new ArrayList<NodeBilotto>();


}//node
```