CMPT 435 - Spring 2023 - Prof. Gormanly

Lab Four

Reece Schenck
Reece.Schenck@Marist.edu

May 1, 2023

1 Program

1.1 LinkedGraph

This class is very similar to the Graph class, and that is because it was based off of the graph class with the only difference being that it uses another class that is housed in this file, the vertex class. For this reason I separated it from the Graph class to make the code easier to read and understand. As far as creating and adding foes, it is almost identical to Graph except I use the vertex class to hold the data. The major differences come in with the depth and breadth search functions. For the depth function I used chat for some help as I could not get this code to work. I only tweaked a few things from what chat gave me for the code. I then used the depth search function as a guide to make the breath search function and made the necessary changes to make it the correct search.

See Lines 8-90 of the linkedGraph class.(Find under Appendix/Code/linkedGraph).

1.2 Matrix

This class is used for printing out the graphs as matrices in conjunction with the graph class. This class creates 2d arrays that can be set with their respective values. The arrays can then be traversed to print out the values.

See Lines 1-25 of the matrix class.(Find under Appendix/Code/matrix)

1.3 Node

This is used as the base building block of the binary search tree as it holds the value, left child, right child, and current path of each node added to the tree

See Lines 2-6 of the Node class.(Find under Appendix/Code/Node)

1.4 BinarySearchTree

For the binary search tree, I used my code from a previous SD1 project as an outline and then heavily modified it to fit into the requirements of the lab. I was not able to get everything working(see unresolved issues and errors section), but other than that one issue it is fully functional. To start

there is a root node that is compared to the new value being added. the new node is then moved along to the left or right based on if it is greater or less than the root. The path is updated and the node is compared again to the roots child. This continues until the node being added reaches a leaf(a node with no children) in which it will be inserted at the proper position. This class also searches the tree by traversing all the way down the left branch of the tree until it hits the bottom left most node. from there it works its way up the tree printing the nodes as it goes with the end result being a list of sorted nodes. The final part of this class is the search function. In this function the root node is compared to the value, after the target is found at the root, or the search is pulled to the left or the right of the tree depending on if it was greater or less than the root, this process is repeated with the next node until the target is found. This (ideally) significantly reduces the amount of comparisons required as there is a big chunk of nodes that can be ignored due to the structure of the tree, thus the comparison count is very low as well as the run time.

See Lines 10-114 of the BinarySearchTree class.(Find under Appendix/Code/BinarySearchTree)

1.5 Graph

I did not know how to start tackling this part so I had ChatGPT make a skeleton class for me to heavily edit and fill in with the functions I needed. This class is used for both printing the graphs as matrices and adjacency lists. It takes in the number of vertices to created a linked array list witch then holds the values of edges given from the add function. From here It can return the linked lists as is since they are already in the form of an adjacency list, or they can be printed out as a matrix instead thanks to some slight tweaking.

See Lines 4-34 of the Graph class.(Find under Appendix/Code/Graph)

1.6 Main

1.6.1 Imports

These are the imports I needed for my code in Main:
import java.util.ArrayList;
import java.util.Arrays;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Random;
import java.util.List;

These Are used for reading/writing from the text file into and Array, and basic array / array list manipulation.

See Lines 1-7 of the Main class.(Find under Appendix/Code/Main)

1.6.2 Read/Write Text Files

I used the same code from Lab 2 with slight changes as there are 3 files instead of 1

See Lines 9-101 of the Main class.(Find under Appendix/Code/Main)

1.6.3 BinarySearchTree Use

I first created a instance of the class and made the root node. I then traverse the magicitems array and insert each value into the tree, printing the path as I go. Once complete I do an inorder search.

After that traverse the magicitems-find array and search the BST for each target value, printing the comparisons and time as I go. Finally I print the average time and comparisons.

See Lines 105-130 of the Main class.(Find under Appendix/Code/Main)

1.6.4 Graph Forms

To start off I took the graph file and read it into an array. I took this array and split the array into a bunch of arrays of the individual graph information that I could use. These arrays were then stored into an arraylist. I then traverse the values in each array in the array list, one array at a time, and count the vertices. I then go through and add the edges to the Graph object. Once complete I create a matrix object and print out the graph as a matrix, I repeat for the rest of the arrays in the array list.

For the adjacency list The process starts the same with counting the vertices and adding the edges to a graph object. The difference is I then use the adjList function of graph to be able to traverse and print the graph as an adjacency list

For the object form I traversed each array in the array list. I set the default start value to 1 and have and if statement change it to 0 if the first vertex is a 0 instead of a 1. I then get the number of vertices. After that I create the linkedGraph object with the array and vertex number (plus 1) before adding the edges. Once complete for all arrays I call the depthFirstTraversal and print the time, then the breadthFirstTraversal and print the time for all graphs. Once done I print the average time for both traversals.

See Lines 140-318 of the Main class.(Find under Appendix/Code/Main)

2 Unresolved Issues and Errors

The only real error that I was not able to figure out in this lab was printing the path to nodes in the BST. As of right now the path finding system I have only half works. I have tried so many different solutions and looked up numerous resources and I could not figure out how to fix it.

3 Results

results may vary

Algorithm	Number of Comparisons	Time in Nanoseconds
BST Lookup	9.00	28530.95
Depth Traversal	-	2574240.00
Breadth Traversal	-	2111260.00

For the BST, the comparisons are low as the set of data being traversed is (ideally) cut in half each comparison, or at least reduced by a significant amount. This in turn makes the time to compute much shorter.

For the two searches, most of the time Breadth First should be the faster option. This is because it finds vertices as it goes down to the end node instead of traversing down a single path, which may be in-optimal, for each node.

4 Appendix

4.1 Code

4.1.1 linkedGraph

```
/*
  This class is similar to the graph class as it does similar things
  however, it requires some different functions that can't be easlity
  implemented into the graph class like I did with the matrix and adjacency list
  and so I am making this new class as to not make the other overly cluttered
*/

import java.util.LinkedList;
import java.util.Queue;

// everything up to and including addEdge is almost identical to the graph class
// the only major differences are the addition and use of the vertex class
// I had a lot of errors implementing this that I used ChatGPT for help quite a bit
public class linkedGraph {
    private int numVertices;
    private Vertex[] vertices;

    public linkedGraph(String[] array, int numVertices){
        this.numVertices = numVertices;
        vertices = new Vertex[numVertices];
        for(int i = 0; i < numVertices; i++){
            vertices[i] = new Vertex(i);
        }
    }

    public void addEdge(int u, int v){
        vertices[u].addEdge(vertices[v]);
        vertices[v].addEdge(vertices[u]);
    }

    /*
      ChatGPT helped me with the public and private searches
      as I was having numerous issues implementing them
      I edited them slightly but the bulk of the code was given by ChatGPT for this part
    */
    public void depthFirstTraversal(int start){
        boolean[] visited = new boolean[numVertices];
        depthFirstTraversal(vertices[start], visited);
        System.out.println("");
    }

    private void depthFirstTraversal(Vertex vertex, boolean[] visited){
        visited[vertex.id] = true;
        System.out.print(vertex.id + " ");
        for(Vertex neighbor : vertex.getNeighbors()){
            if(!visited[neighbor.id]){
                depthFirstTraversal(neighbor, visited);
            }
        }
    }

    //
    public void breadthFirstTraversal(int start){
        // stores if a vertex has been visited
        boolean[] visited = new boolean[numVertices];
        Queue<Vertex> queue = new LinkedList<>();
        queue.add(vertices[start]);
        visited[start] = true;
```

```
59
60          while(!queue.isEmpty()){
61              // Retrieves and removes the queue head, breaks loop if empty
62              Vertex vertex = queue.poll();
63              System.out.print(vertex.id + " ");
64              for(Vertex neighbor : vertex.getNeighbors()){
65                  if(!visited[neighbor.id]){
66                      queue.add(neighbor);
67                      visited[neighbor.id] = true;
68                  }
69              }
70          }
71      }
72  }
73
74  // I used ChatGPT to help me trouble shoot my code and
75  // it had the vertex class outside of linkedGraph on the same file
76  class Vertex {
77      public int id;
78      public LinkedList<Vertex> neighbors;
79
80      public Vertex(int id){
81          this.id = id;
82          neighbors = new LinkedList<>();
83      }
84
85      public void addEdge(Vertex v){
86          neighbors.add(v);
87      }
88
89      public LinkedList<Vertex> getNeighbors(){
90          return neighbors;
91      }
92  }
```

4.1.2 matrix

```
1  // Used ChatGPT to help with some troubleshooting
2  public class matrix {
3      private int numRows;
4      private int numCols;
5      private int[][] data;
6
7      public matrix(int numRows, int numCols){
8          this.numRows = numRows;
9          this.numCols = numCols;
10         data = new int[numRows][numCols];
11     }
12
13     public void set(int i, int j, int value){
14         data[i][j] = value;
15     }
16
17     public void print(){
18         for(int i = 0; i < numRows; i++){
19             for(int j = 0; j < numCols; j++){
20                 System.out.print(data[i][j] + " ");
21             }
22             System.out.println();
23         }
24     }
25 }
```

4.1.3 Graph

```
import java.util.ArrayList;

// Used ChatGPT to make the base graph class and then edited to fit my code
public class Graph {
    private int numVertices;
    private ArrayList<ArrayList<Integer>> adjList;

    public Graph(int numVertices){
        this.numVertices = numVertices;
        adjList = new ArrayList<ArrayList<Integer>>();
        for(int i = 0; i < numVertices; i++){
            adjList.add(new ArrayList<Integer>());
        }
    }

    public void addEdge(int u, int v){
        adjList.get(u).add(v);
        adjList.get(v).add(u);
    }

    public matrix toMatrix(){
        matrix matrix = new matrix(numVertices, numVertices);
        for(int i = 0; i < numVertices; i++){
            for(int j = 0; j < adjList.get(i).size(); j++){
                int neighbor = adjList.get(i).get(j);
                matrix.set(i, neighbor, 1);
            }
        }
        return matrix;
    }

    public ArrayList<ArrayList<Integer>> toAdjList(){
        return adjList;
    }
}
```

4.1.4 BinarySearchTree

```
//Taken from previous SD1 project and heavily edited
import java.util.*;

public class BinarySearchTree{
    //String path = "";

    //public BinarySearchTree(){
        //this.path = path;
    //}
    int comparisonTotal = 0;
    double totalTime = 0;
    long startTime;
    long endTime;
    long elapsedTime;

    //inserts a new node to the proper position
    public Node insert(Node newNode, String val){
        if(newNode == null){
            //path = "";
            return NewNode(val);
        }

        //if(val < newNode.value)
```

```
24          if(val.compareToIgnoreCase(newNode.value) < 0){
25              newNode.path = newNode.path + "L ";
26              newNode.left = insert(newNode.left, val);
27          }
28          //else if(val > newNode.value)
29          else if(val.compareToIgnoreCase(newNode.value) > 0){
30              newNode.path = newNode.path + "R ";
31              newNode.right = insert(newNode.right, val);
32          }
33          //System.out.println(path);
34          return newNode;
35      }
36
37      //creates a new node
38      public Node NewNode(String value){
39          Node newNode = new Node();
40
41          newNode.value = value;
42          newNode.left = null;
43          newNode.right = null;
44
45          newNode.path = "";
46
47          return newNode;
48      }
49
50      public String getPath(Node newNode){
51          return newNode.path;
52      }
53
54      public void clearPath(Node newNode){
55          newNode.path = "";
56      }
57
58
59      //traverses the tree and prints the elements inorder
60      //elements are seperated by a '.'
61      public void inorderSort(Node inNode){
62          //immediatly stops in node has no value
63          if(inNode == null){
64              return;
65          }
66
67          inorderSort(inNode.left);
68          System.out.print(inNode.value + ". ");
69          inorderSort(inNode.right);
70      }
71
72      //add times + comparison
73      public void search(Node node, String target){
74          node.path = "";
75          int comparisons = 0;
76          startTime = System.nanoTime();
77          while(target.compareToIgnoreCase(node.value) != 0){
78              if(target.compareToIgnoreCase(node.value) < 0){
79                  node.path = node.path + "L ";
80                  node = node.left;
81              }else{
82                  node.path = node.path + "R ";
83                  node = node.right;
84              }
85              comparisons++;
86          }
87          endTime = System.nanoTime();
```

```
        elapsedTime = endTime - startTime;
        totalTime = totalTime + elapsedTime;

        System.out.println("Path of " + target + ":");
        System.out.println(node.path);

        System.out.println("Number of comparisons: ");
        System.out.println(comparisons);
        comparisonTotal = comparisonTotal + comparisons;

        System.out.println("Time in nanoseconds: ");
        System.out.println(elapsedTime);
    }

    public void averageComp(int length){
        double averageComp = comparisonTotal / length;
        System.out.println("Average Comparisons: ");
        System.out.format("%.2f%n", averageComp);

    }

    public void averageTime(int length){
        double averageTime = totalTime / length;
        System.out.println("Average time in nanoseconds:");
        System.out.format("%.2f%n", averageTime);
    }
}
```

4.1.5 Node

```
// taken from previous SD1 project
public class Node{
    String value;
    Node left;
    Node right;
    String path = "";
}
```

4.1.6 Main

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        //Read and Write text files into arrays
        //same code from other labs, changed slightly for reading more files

        //1st read file

        //This wasn't working for me:
        String filename = "magicitems.txt";

        //It only worked when hard coded, so this is what I used for testing:
        //String filename = "C:\\Users\\goldh\\OneDrive\\Documents\\GitHub\\RSchenck-435\\Lab 4\\magici
```

```
22          ArrayList<String> lines = new ArrayList<String>();
23
24          try {
25              BufferedReader reader = new BufferedReader(new FileReader(filename));
26              String line = reader.readLine();
27              while (line != null) {
28                  lines.add(line);
29                  line = reader.readLine();
30              }
31              reader.close();
32          } catch (IOException e) {
33              e.printStackTrace();
34          }
35
36          String[] linesArrayMagic = lines.toArray(new String[lines.size()]);
37          //prints out the read lines, used for testing
38          //System.out.println("Lines read from file:");
39          //for (String l : linesArrayMagic) {
40          //    System.out.println(l);
41          //}
42
43
44          //2nd read file
45
46          //This wasn't working for me:
47          String filename2 = "magicitems-find-in-bst.txt";
48
49          //It only worked when hard coded, so this is what I used for testing:
50          //String filename2 = "C:\\Users\\goldh\\OneDrive\\Documents\\GitHub\\RSchenck-435\\Lab 4\\magic
51
52          ArrayList<String> lines2 = new ArrayList<String>();
53
54          try {
55              BufferedReader reader = new BufferedReader(new FileReader(filename2));
56              String line = reader.readLine();
57              while (line != null) {
58                  lines2.add(line);
59                  line = reader.readLine();
60              }
61              reader.close();
62          } catch (IOException e) {
63              e.printStackTrace();
64          }
65
66          String[] linesArrayFind = lines2.toArray(new String[lines2.size()]);
67          //prints out the read lines, used for testing
68          //System.out.println("Lines read from file:");
69          //for (String l : linesArrayFind) {
70          //    System.out.println(l);
71          //}
72
73
74          //3rd read file
75
76          //This wasn't working for me:
77          String filename3 = "graphs1.txt";
78
79          //It only worked when hard coded, so this is what I used for testing:
80          //String filename3 = "C:\\Users\\goldh\\OneDrive\\Documents\\GitHub\\RSchenck-435\\Lab 4\\graph
81
82          ArrayList<String> lines3 = new ArrayList<String>();
83
84          try {
85              BufferedReader reader = new BufferedReader(new FileReader(filename3));
```

```
            String line = reader.readLine();
            while (line != null) {
                lines3.add(line);
                line = reader.readLine();
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        String[] linesArrayGraph = lines3.toArray(new String[lines3.size()]);
        //prints out the read lines, used for testing
        //System.out.println("Lines read from file:");
        //for (String l : linesArrayGraph) {
        //    System.out.println(l);
        //}


        //creates a binary search tree
        BinarySearchTree tree = new BinarySearchTree();

        //Creates the root node
        Node root = null;

        //adds items to the tree and prints out the paths
        //finding path is broken but IDK whats wrong or how to fix it
        for(int i=0;i<linesArrayMagic.length;i++){
            root = tree.insert(root, linesArrayMagic[i]);
            System.out.println("Path to " + linesArrayMagic[i] + ":");
            System.out.println(tree.getPath(root));
            //doesn't fix the issues of path finding
            //tree.clearPath(root);
        }

        //prints out the elements of the tree through in-order traversal
        //items are printed in order and are seperated by a '.'
        tree.inorderSort(root);


        for(int i=0;i<linesArrayFind.length;i++){
            tree.search(root, linesArrayFind[i]);
            System.out.println("");
        }
        tree.averageComp(linesArrayFind.length);
        tree.averageTime(linesArrayFind.length);

        /*
           takes information from the strings in graphs1.txt
           and turns them into usable arrays of strings
           that are stored in the array list splitArrays
        */

        // This code was taken from ChatGPT and then edited to work with strings
        // as well as the arrays and array lists needed to function in my code
        String keyPhrase = "new graph";
        List<String[]> splitArrays = new ArrayList<>();
        List<String> currentArray = new ArrayList<>();
        for(String str : linesArrayGraph){
            if(str.equals(keyPhrase)){
                splitArrays.add(currentArray.toArray(new String[currentArray.size()]));
                currentArray.clear();
            }else{
                // gets rid of the comments above each new graph
                // also new graph line is not added to split array
```

```
150                 // found .startswith from https://docs.oracle.com/javase/tutorial/java/data/comparestri
151                 if(str.startsWith("--") == false){
152                     currentArray.add(str);
153                 }
154             }
155         }
156         // Adds the last array
157         splitArrays.add(currentArray.toArray(new String[currentArray.size()]));
158 
159         // for testing
160         //System.out.println("Original Array: " + Arrays.toString(originalArray));
161         //System.out.println("Original Array: " + Arrays.toString(linesArrayGraph));
162 
163         // first index will always be a useless blank array because
164         // of the removal of the comments and array splits at 'new graph'
165         splitArrays.remove(0);
166 
167         // for testing
168         //System.out.println("Split Arrays:");
169         //for(String[] splitArray : splitArrays){
170         //    System.out.println(Arrays.toString(splitArray));
171         //}
172         //System.out.println(Arrays.toString(splitArrays.get(1)));
173 
174         //prints out the graphs as matrixs
175         System.out.println("Matrix Form:");
176         for(String[] splitArray : splitArrays){
177             int vertexNum = 0;
178             for(String str : splitArray){
179                 if(str.startsWith("add vertex")){
180                     vertexNum++;
181                 }
182             }
183             // used for testing
184             System.out.println("Graph with " + vertexNum + " vertexs");
185             Graph graph = new Graph(vertexNum+1);
186             for(String str : splitArray){
187                 int num1;
188                 int num2;
189                 if(str.startsWith("add edge")){
190                     String[] split = str.split(" ");
191                     // .parseInt was found here:
192                     // https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-
193                     num1 = Integer.parseInt(split[2]);
194                     num2 = Integer.parseInt(split[4]);
195                     graph.addEdge(num1, num2);
196                 }
197             }
198             matrix matrix = graph.toMatrix();
199             matrix.print();
200             System.out.println("");
201         }
202 
203         // adjacency list and matrix could be combined but I seperated them to make it look nice when p
204 
205         // prints the graphs as adjacency lists
206         System.out.println("Adjacency List Form:");
207         for(String[] splitArray : splitArrays){
208             int vertexNum = 0;
209             for(String str : splitArray){
210                 if(str.startsWith("add vertex")){
211                     vertexNum++;
212                 }
213             }
```

```
214             // used for testing
215             System.out.println("");
216             System.out.println("Graph with " + vertexNum + " vertexs");
217             Graph graph = new Graph(vertexNum+1);
218             for(String str : splitArray){
219                 int num1;
220                 int num2;
221                 if(str.startsWith("add edge")){
222                     String[] split = str.split(" ");
223                     // .parseInt was found here:
224                     // https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-
225                     num1 = Integer.parseInt(split[2]);
226                     num2 = Integer.parseInt(split[4]);
227                     graph.addEdge(num1, num2);
228                 }
229             }
230             // used ChatGPT for minor formatting help
231             ArrayList<ArrayList<Integer>> adjList = graph.toAdjList();
232             for(int i=0; i<adjList.size(); i++){
233                 System.out.print(i + ": ");
234                 for(int j=0; j<adjList.get(i).size(); j++){
235                     System.out.print(adjList.get(i).get(j) + " ");
236                 }
237                 System.out.println("");
238             }
239         }
240 
241         // the reason this is seperated instead of combined is that same as I stated before
242 
243         //makes the graph as a linked object
244         System.out.println("");
245         System.out.println("Linked Object Form:");
246         int start = 1;
247         double totalTimeDepth = 0;
248         double totalTimeBreadth = 0;
249         long startTime;
250         long endTime;
251         long elapsedTime;
252         int timesRun = 0;
253         for(String[] splitArray : splitArrays){
254             // start index defaults to 0
255             int vertexNum = 0;
256             for(String str : splitArray){
257                 // updates the start value for traversals based off the first index
258                 // this was the only way I could figure out to hanled having
259                 // starting vertecies of both 0 and 1 in the same file
260                 if(str.equals("add vertex 0")){
261                     start = 0;
262                 }
263                 // used for testing
264                 //System.out.println("start value: " + start);
265                 if(str.startsWith("add vertex")){
266                     vertexNum++;
267                 }
268             }
269             // used for testing
270             System.out.println("");
271             System.out.println("Graph with " + vertexNum + " vertexs");
272             linkedGraph linked = new linkedGraph(splitArray, vertexNum+1);
273             for(String str : splitArray){
274                 int num1;
275                 int num2;
276                 if(str.startsWith("add edge")){
277                     String[] split = str.split(" ");
```

```
                    // .parseInt was found here:
                    // https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-
                    num1 = Integer.parseInt(split[2]);
                    num2 = Integer.parseInt(split[4]);
                    linked.addEdge(num1, num2);
                }
            }
            System.out.println("Depth first traversal: ");
            startTime = System.nanoTime();
            linked.depthFirstTraversal(start);
            endTime = System.nanoTime();
            elapsedTime = endTime - startTime;
            totalTimeDepth = totalTimeDepth + elapsedTime;
            System.out.println("Time in nanoseconds: ");
            System.out.println(elapsedTime);

            System.out.println("");

            System.out.println("Breadth first traversal: ");
            startTime = System.nanoTime();
            linked.breadthFirstTraversal(start);
            endTime = System.nanoTime();
            elapsedTime = endTime - startTime;
            totalTimeBreadth = totalTimeBreadth + elapsedTime;

            System.out.println("");
            System.out.println("Time in nanoseconds: ");
            System.out.println(elapsedTime);
            System.out.println(" ");

            timesRun++;

            // breath first should almost always be faster, however it is close
        }
        double averageTimeDepth = totalTimeDepth / timesRun;
        System.out.println("Average depth first traversal time: ");
        System.out.format("%.2f%n", averageTimeDepth);

        double averageTimeBreadth = totalTimeBreadth / timesRun;
        System.out.println("Average breadth first traversal time: ");
        System.out.format("%.2f%n", averageTimeBreadth);
    }
}
```

4.1.7 Text File (magicitems)

I'm not putting all of the words here because it's 666 lines of text/code

4.1.8 Text File (magicitems-find-in-bst)

It is 42 words chosen from the magicitems text file that are searched for in the BST.

4.1.9 Text File (graphs1)

I'm not putting all of the words here because it's 375 lines of text/code that give the dimensions of the graphs I am printing and searching