CMPT 435 - Fall 2021 - Dr. Labouseur

Assignment Four

Nicholas Petrilli

November 20, 2021

1 Binary Search Tree

Listing 1: BST Class

```
1  public class BST {
2       TreeNode root;
3
4      static int totalComparisons = 0;
5      static int comparisons = 0;
6
7
8      public BST() {
9          root = new TreeNode();
10     }
11     //Doesn't work properly, returns an average of around 14 per search when
12     public void search(TreeNode treeRoot, String target) {
13         comparisons = 0;
14         TreeNode current = treeRoot;
15         TreeNode parent = null;
16         String path = "";
17         while (current != null && current.data.compareTo(target) != 0) {
18             parent = current;
19             if (target.compareTo(current.data) < 0 ) {
20                 current = current.left;
21                 path += "L, ";
22                 comparisons++;
23             }
24             else {
25                 current = current.right;
26                 path += "R, ";
27                 comparisons++;
28             }
29         }
30
```

```
31          if (parent == null) {
32              System.out.println("The node with target string " + target + " is
33          }
34          else if (target.compareTo(parent.data) < 0) {
35              path += "L";
36              comparisons++;
37          }
38          else {
39              path += "R";
40              comparisons++;
41          }
42          System.out.println("The path for searching for " + target + " was " +
43          totalComparisons += comparisons;
44
45      }
46
47
48      //Insert a node into the tree
49      public void insert(BST tree, TreeNode newNode) {
50          String path = "";
51          TreeNode trailing = null;
52          TreeNode current = tree.root;
53          while (current != null) {
54              trailing = current;
55              if (newNode.data.compareToIgnoreCase(current.data) < 0) {
56                  current = current.left;
57                  path += "L, ";
58              }
59
60              else { //must be >=
61                  current = current.right;
62                  path += "R, ";
63              }
64
65          }
66          newNode.parent = trailing;
67          if (trailing == null)
68              tree.root = newNode;
69          else if (newNode.data.compareToIgnoreCase(trailing.data) < 0) {
70              trailing.left = newNode;
71              path += "L";
72          }
73
74          else {
75              trailing.right = newNode;
76              path += "R";
```

```
77          }
78          System.out.println("The path for inserting " + newNode.data + " was "
79      }
80      public void inOrderTraversal(TreeNode node) {
81          //base case to know there are no more nodes left
82          if (node == null)
83              return;
84          inOrderTraversal(node.left);
85          System.out.println(node.data);
86          inOrderTraversal(node.right);
87      }
88      public int getTotalComparisons() {
89          return totalComparisons;
90      }
91      public int getComparisons() {
92          return comparisons;
93      }
94  }
```

This is my binary search tree class. Insert function properly works, but there is an issue with the search function. Due to this error in the search function, the average number of comparisons for searching was 14, higher than the expected log(n) which is 9-10.

2 Tree Node

Listing 2: TreeNode Class

```
1   public class TreeNode {
2       String data;
3       TreeNode left;
4       TreeNode right;
5       TreeNode parent;
6
7       public TreeNode() {
8           data = "";
9           left = right = parent = null;
10      }
11      public TreeNode(String data) {
12          this.data = data;
13          left = right = parent = null;
14      }
15
16  }
```

This class is my TreeNode class, which is used to populate the binary search tree.

The left, right and parent pointers are used in the BST class when searching and inserting.

3 GraphMatrix

Listing 3: GraphMatrix Class

```
1  public class GraphMatrix {
2      int graphMatrix[][];
3      int numVerticies;
4
5      //Construct matrix graph with number of verticies
6      public GraphMatrix(int numVerticies) {
7          this.numVerticies = numVerticies;
8          graphMatrix = new int[numVerticies][numVerticies];
9      }
10
11     public void addEdge(int vertex1, int vertex2) {
12         //Add edge for both directions for undirected graph
13         graphMatrix[vertex1][vertex2] = 1;
14
15         graphMatrix[vertex2][vertex1] = 1;
16     }
17     public void printGraphMatrix() {
18         for (int i = 0; i < numVerticies; i++) {
19             System.out.print(i + "␣|␣");
20             for (int j = 0; j < numVerticies; j++) {
21                 System.out.print(graphMatrix[i][j] + "␣");
22             }
23             System.out.println();
24         }
25     }
26
27 }
```

This is my GraphMatrix class, which is used to represent the matrix version of the undirected graphs. Add edge adds the edge in both directions in the matrix, represented by a 1 in the matrix. The function printGraphMatrix prints the properly formatted matrix with the number of verticies and the proper linked edges.

4 Vertex

Listing 4: Vertex Class

```
1  import java.util.*;
```

```
2
3  public class Vertex {
4      int id;
5      boolean processed;
6      ArrayList<Vertex> neighbors;
7
8      public Vertex(int id) {
9          this.id = id;
10         processed = false;
11         neighbors = new ArrayList<>();
12     }
13     public Vertex() {
14         id = 0;
15         processed = false;
16         neighbors = new ArrayList<>();
17     }
18
19 }
```

This is my Vertex class, which is used for the GraphLinkedObjects which contains an array list of vertices. Each vertex has an id number, a boolean value to check if it had been processed, and an array list of verticies that indicate its neighbors (where edges connect).

5 GraphLinkedObjects

Listing 5: GraphLinkedObjects Class

```
1  import java.util.*;
2
3  public class GraphLinkedObjects {
4
5      ArrayList<Vertex> graphLinkedObjects;
6
7
8      public GraphLinkedObjects(int numVerticies) {
9          graphLinkedObjects = new ArrayList<>(numVerticies);
10
11     }
12     //Add edge in both directions for undirected graph
13     public void addEdge(int vertex1, int vertex2) {
14         graphLinkedObjects.get(vertex1).neighbors.add(new Vertex(vertex2));
15         graphLinkedObjects.get(vertex2).neighbors.add(new Vertex(vertex1));
16     }
17     public void depthFirstTraversal(Vertex v) {
18         if (!(v.processed)) {
```

```
19              System.out.print(v.id + " ");
20              v.processed = true;
21          }
22          for (Vertex n: v.neighbors) {
23              if (!(n.processed)) {
24                  depthFirstTraversal(n);
25              }
26          }
27      }
28      public void breadthFirstTraversal(Vertex v) {
29          Queue2 queue = new Queue2();
30          queue.enqueue(v);
31          v.processed = true;
32          Vertex currentVertex = new Vertex();
33          while (!(queue.isEmpty())) {
34              currentVertex = queue.dequeue();
35              System.out.print(currentVertex.id + " ");
36
37              for (Vertex n: currentVertex.neighbors) {
38                  if (!(n.processed)) {
39                      queue.enqueue(n);
40                      n.processed = true;
41                  }//end if
42              }//end for
43
44          }//end while
45
46      }
47      public void reset() {
48          for (int i = 0; i < graphLinkedObjects.size(); i++) {
49              graphLinkedObjects.get(i).processed = false;
50          }
51      }
52      /*
53      public void printGraph() {
54          for (Vertex v: graphLinkedObjects) {
55              System.out.print("Vertex " + v.id);
56              for (int x = 0; x < v.neighbors.size(); x++) {
57                  System.out.println(" has neighbors " + v.neighbors.get(x));
58              }
59          }
60      }
61      */
62  }
```

This class is my GraphLinkedObjects. It includes breadth first traversal and

depth first traversal, which both have asympotic running times of O(—V— + —E—). This is so because in both traversals, every vertex and every edge will be checked once when traversing through the graph. I struggled implementing this class, as the add edge function still causes an error. I tried to implement it the way the slides show, however I wasn't successful in getting it to work.

6 GraphAdjacencyList

Listing 6: GraphLinkedObjects Class

```
1  import java.util.*;
2
3  public class GraphAdjacencyList {
4      ArrayList<ArrayList<Integer>> graphAdjacencyList;
5
6      public GraphAdjacencyList(int numVerticies) {
7          graphAdjacencyList = new ArrayList<>(numVerticies);
8
9      }
10     public void addEdge(int vertex1, int vertex2) {
11         graphAdjacencyList.get(vertex1).add(vertex1);
12         graphAdjacencyList.get(vertex2).add(vertex2);
13     }
14     public void printGraph() {
15         for (List newList: graphAdjacencyList) {
16             for (Object m: newList) {
17                 System.out.println(newList.get((int) m));
18
19             }
20         }
21     }
22
23 }
```

This is my GraphAdjacencyList class, which I also struggled implementing. Adding edges still results in an error as I am unsure if my representation using an array list of array lists was accurate. For these reasons, the code in main is commented out when calling this function in order to have the program compile without any errors.

7 Main

Listing 7: Graphs (Main) Class

```
1  import java.io.*;
```

```
2  import java.util.*;
3
4  public class Graphs {
5      final static int NUM_ITEMS = 666;
6      final static int NUM_ITEMS_2 = 42;
7      final static int NUM_LINES_IN_GRAPHS = 375;
8
9      static GraphMatrix graphMatrix;
10     static GraphLinkedObjects graphLinkedObjects;
11     static GraphAdjacencyList graphAdjacencyList;
12
13
14     public static void main(String[] args) {
15         BST bTree = new BST();
16         String[] magicItems = new String[NUM_ITEMS];
17         int i = 0;
18
19         File myFile = new File("magicitems.txt");
20
21         try {
22             Scanner fileScan = new Scanner(myFile);
23         //Populates the array with the lines from text file
24         while (fileScan.hasNext()) {
25             magicItems[i] = fileScan.nextLine().toLowerCase().replace(" ", ""
26             i++;
27         }
28         fileScan.close();
29         }
30         catch (FileNotFoundException e) {
31             e.printStackTrace();
32
33         }
34         //Creates nodes for each element in the array, and inserts them into
35         for (int j = 0; j < NUM_ITEMS; j++) {
36             TreeNode temp = new TreeNode(magicItems[j]);
37             bTree.insert(bTree, temp);
38         }
39
40         System.out.println("\n**In Order Traversal of Binary Search Tree**");
41
42         //Prints out in order traversal of binary search tree
43         bTree.inOrderTraversal(bTree.root);
44
45         //Create new array for elements to search for in binary search tree a
46         String[] magicItems2 = new String[NUM_ITEMS_2];
47         File myFile2 = new File("magicitems-find-in-bst.txt");
```

```
48          int k = 0;
49
50          try {
51              Scanner fileScan2 = new Scanner(myFile2);
52          //Populates the array with the lines from text file
53          while (fileScan2.hasNext()) {
54              magicItems2[k] = fileScan2.nextLine().toLowerCase();
55              k++;
56          }
57          fileScan2.close();
58          }
59          catch (FileNotFoundException e) {
60              e.printStackTrace();
61
62          }
63          //Search for each of the 42 magic items in the binary search tree
64
65          for (int m = 0; m < NUM_ITEMS_2; m++) {
66              System.out.println(bTree.root.data);
67              bTree.search(bTree.root, magicItems2[m]);
68              System.out.println("The number of comparisons was " + bTree.getCo
69          }
70          double average = bTree.getTotalComparisons() / NUM_ITEMS_2;
71          System.out.println("The average number of comparisons for a search in
72          System.out.println(bTree.root.data);
73
74          String[] graphInstructions = new String[NUM_LINES_IN_GRAPHS];
75
76          int n = 0;
77
78          File myFile3 = new File("graphs1.txt");
79
80          try {
81              Scanner fileScan3 = new Scanner(myFile3);
82          //Populates the array with the lines from text file
83          while (fileScan3.hasNext()) {
84              graphInstructions[n] = fileScan3.nextLine().toLowerCase();
85              n++;
86          }
87          fileScan3.close();
88          }
89          catch (FileNotFoundException e) {
90              e.printStackTrace();
91
92          }
93
```

```
94          int graphNum = 0;
95
96          ArrayList<Integer> graph1Verticies = new ArrayList<Integer>();
97          ArrayList<Integer> graph2Verticies = new ArrayList<Integer>();
98          ArrayList<Integer> graph3Verticies = new ArrayList<Integer>();
99          ArrayList<Integer> graph4Verticies = new ArrayList<Integer>();
100         ArrayList<Integer> graph5Verticies = new ArrayList<Integer>();
101         ArrayList<Integer> graph1Edges = new ArrayList<Integer>();
102         ArrayList<Integer> graph2Edges = new ArrayList<Integer>();
103         ArrayList<Integer> graph3Edges = new ArrayList<Integer>();
104         ArrayList<Integer> graph4Edges = new ArrayList<Integer>();
105         ArrayList<Integer> graph5Edges = new ArrayList<Integer>();
106
107
108         /*
109         GraphMatrix gm = new GraphMatrix(5);
110         gm.addEdge(1, 2);
111         gm.addEdge(1, 4);
112         gm.addEdge(3, 4);
113         gm.printGraphMatrix();
114         */
115
116         /*
117         GraphLinkedObjects glo = new GraphLinkedObjects(8);
118         glo.addEdge(1, 5);
119         glo.addEdge(2, 4);
120         glo.addEdge(6, 7);
121         for (int g = 0; g < 8; g++) {
122              Vertex v = new Vertex(g);
123              glo.breadthFirstTraversal(v);
124              glo.depthFirstTraversal(v);
125         }
126         */
127         /*
128         GraphAdjacencyList adj = new GraphAdjacencyList(6);
129         adj.addEdge(1, 2);
130         adj.addEdge(2, 4);
131         adj.addEdge(4, 5);
132         adj.printGraph();
133         */
134
135
136         for (int x = 0; x < NUM_LINES_IN_GRAPHS; x++) {
137              //System.out.println(graphInstructions[x]);
138              if (graphInstructions[x].contains("new")) {
139                   graphNum++;
```

```
140
141                 System.out.println("Creating Graph Number: " + graphNum);
142                 //need to wait to actually create graph because we need the n
143
144
145             }//if
146             else if (graphInstructions[x].contains("add vertex")) {
147                 //removes all non numeric text in line
148                 String num = graphInstructions[x].replaceAll("[^\\d.]", "");
149                 int vertex = Integer.parseInt(num);
150
151                 //determine which number graph list to add the vertex to
152                 if(graphNum == 1) {
153                     graph1Verticies.add(vertex);
154
155                 }
156                 else if(graphNum == 2) {
157                     graph2Verticies.add(vertex);
158                 }
159                 else if(graphNum == 3) {
160                     graph3Verticies.add(vertex);
161                 }
162                 else if(graphNum == 4) {
163                     graph4Verticies.add(vertex);
164                 }
165                 else if(graphNum == 5) {
166                     graph5Verticies.add(vertex);
167                 }
168             }
169             else if (graphInstructions[x].contains("add edge")) {
170                 int edge1 = 0;
171                 int edge2 = 0;
172
173
174                 //Split the array at the spaces, and check if the new strings
175                 //if they are numbers, they are the edge numbers we need to a
176                 ArrayList<Integer> intList = new ArrayList<Integer>();
177                 for (String splitString: graphInstructions[x].split(" ")) {
178                     if(isNumber(splitString)) {
179                         intList.add(Integer.parseInt(splitString));
180                     }//if
181
182                 }//for
183                 //there are only two numbers in the line, so they will be the
184                 edge1 = intList.get(0);
185                 edge2 = intList.get(1);
```

```
186
187                 //System.out.println(edge1 + " and " + edge2);
188
189                 if (graphNum == 1) {
190                     graph1Edges.add(edge1);
191                     graph1Edges.add(edge2);
192                 }
193                 else if (graphNum == 2) {
194                     graph2Edges.add(edge1);
195                     graph2Edges.add(edge2);
196                 }
197                 else if (graphNum == 3) {
198                     graph3Edges.add(edge1);
199                     graph3Edges.add(edge2);
200                 }
201                 else if (graphNum == 4) {
202                     graph4Edges.add(edge1);
203                     graph4Edges.add(edge2);
204                 }
205                 else if (graphNum == 5) {
206                     graph5Edges.add(edge1);
207                     graph5Edges.add(edge2);
208                 }//else if
209
210
211                 //need to check that the list of edges for the graph isn't em
212                 if (graphNum == 1 && !(graph1Edges.isEmpty())) {
213                     //create the graphs
214                     graphMatrix = new GraphMatrix(graph1Verticies.size() + 1)
215                     graphLinkedObjects = new GraphLinkedObjects(graph1Vertici
216                     graphAdjacencyList = new GraphAdjacencyList(graph1Vertici
217
218                     while (!(graph1Edges.isEmpty())) {
219                         //add the edges to the graphs
220                         System.out.println("adding edges " + graph1Edges.get(
221                         graphMatrix.addEdge(graph1Edges.get(0), graph1Edges.g
222                         //commented out because causing errors
223                         //graphLinkedObjects.addEdge(graph1Edges.get(0), graph
224                         //graphAdjacencyList.addEdge(graph1Edges.get(0), graph
225
226                         //remove the two edges that were just added each time
227                         graph1Edges.remove(0);
228                         graph1Edges.remove(0);
229                     }//while
230
231                     System.out.println("Graph Matrix:");
```

```
232                 graphMatrix.printGraphMatrix();
233                 //Needed to do separately in order to print out correctly
234                 //Only included for the first graph because the output wo
235
236                 for (int y = 0; y < graph1Verticies.size(); y++) {
237                     Vertex v = new Vertex(graph1Verticies.get(y));
238                     graphLinkedObjects.breadthFirstTraversal(v);
239                 }
240                 System.out.println();
241                 graphLinkedObjects.reset();
242                 for (int z = 0; z < graph1Verticies.size(); z++) {
243                     Vertex v = new Vertex(graph1Verticies.get(z));
244                     graphLinkedObjects.depthFirstTraversal(v);
245                 }
246                 System.out.println();
247
248
249             }//if
250             else if (graphNum == 2 && !(graph2Edges.isEmpty())) {
251                 graphMatrix = new GraphMatrix(graph2Verticies.size() + 1)
252                 graphLinkedObjects = new GraphLinkedObjects(graph2Vertici
253                 graphAdjacencyList = new GraphAdjacencyList(graph2Vertici
254
255                 while(!(graph2Edges.isEmpty())) {
256                     graphMatrix.addEdge(graph2Edges.get(0), graph2Edges.g
257                     //graphLinkedObjects.addEdge(graph2Edges.get(0), grapi
258                     //graphAdjacencyList.addEdge(graph2Edges.get(0), grapi
259
260                     graph2Edges.remove(0);
261                     graph2Edges.remove(0);
262                 }//while
263
264                 //System.out.println("Graph Matrix:");
265                 //graphMatrix.printGraphMatrix();
266                 /*
267                 for (int y = 0; y < graph2Verticies.size(); y++) {
268                     Vertex v = new Vertex(graph2Verticies.get(y));
269                     graphLinkedObjects.depthFirstTraversal(v);
270                     graphLinkedObjects.breadthFirstTraversal(v);
271                 }
272                 */
273             }//else if
274
275             else if (graphNum == 3 && !(graph3Edges.isEmpty())) {
276                 graphMatrix = new GraphMatrix(graph3Verticies.size() + 1)
277                 graphLinkedObjects = new GraphLinkedObjects(graph3Vertici
```

```
278                 graphAdjacencyList = new GraphAdjacencyList(graph3Vertici
279
280                 while(!(graph3Edges.isEmpty())) {
281                     graphMatrix.addEdge(graph3Edges.get(0), graph3Edges.g
282                     //graphLinkedObjects.addEdge(graph3Edges.get(0), grap
283                     //graphAdjacencyList.addEdge(graph3Edges.get(0), grap
284
285                     graph3Edges.remove(0);
286                     graph3Edges.remove(0);
287                 }//while
288
289                 //System.out.println("Graph Matrix:");
290                 //graphMatrix.printGraphMatrix();
291                 /*
292                 for (int y = 0; y < graph3Verticies.size(); y++) {
293                     Vertex v = new Vertex(graph3Verticies.get(y));
294                     graphLinkedObjects.depthFirstTraversal(v);
295                     graphLinkedObjects.breadthFirstTraversal(v);
296                 }
297                 */
298
299             }//else if
300
301             else if (graphNum == 4 && !(graph4Edges.isEmpty())) {
302                 graphMatrix = new GraphMatrix(graph4Verticies.size() + 1)
303                 graphLinkedObjects = new GraphLinkedObjects(graph4Vertici
304                 graphAdjacencyList = new GraphAdjacencyList(graph4Vertici
305
306                 while(!(graph4Edges.isEmpty())) {
307                     graphMatrix.addEdge(graph4Edges.get(0), graph4Edges.g
308                     //graphLinkedObjects.addEdge(graph4Edges.get(0), grap
309                     //graphAdjacencyList.addEdge(graph4Edges.get(0), grap
310
311                     graph4Edges.remove(0);
312                     graph4Edges.remove(0);
313                 }//while
314
315                 //System.out.println("Graph Matrix:");
316                 //graphMatrix.printGraphMatrix();
317                 /*
318                 for (int y = 0; y < graph4Verticies.size(); y++) {
319                     Vertex v = new Vertex(graph4Verticies.get(y));
320                     graphLinkedObjects.depthFirstTraversal(v);
321                     graphLinkedObjects.breadthFirstTraversal(v);
322                 }
323                 */
```

```
324                 }//else if
325
326                 else if (graphNum == 5 && !(graph5Edges.isEmpty())) {
327                     graphMatrix = new GraphMatrix(graph5Verticies.size() + 1)
328                     graphLinkedObjects = new GraphLinkedObjects(graph5Vertici
329                     graphAdjacencyList = new GraphAdjacencyList(graph5Vertici
330
331                     while(!(graph5Edges.isEmpty())) {
332                         graphMatrix.addEdge(graph5Edges.get(0), graph5Edges.g
333                         //graphLinkedObjects.addEdge(graph5Edges.get(0), grap
334                         //graphAdjacencyList.addEdge(graph5Edges.get(0), grap
335
336                         graph5Edges.remove(0);
337                         graph5Edges.remove(0);
338                     }//while
339
340                     //System.out.println("Graph Matrix:");
341                     //graphMatrix.printGraphMatrix();
342                     /*
343                     for (int y = 0; y < graph5Verticies.size(); y++) {
344                         Vertex v = new Vertex(graph5Verticies.get(y));
345                         graphLinkedObjects.depthFirstTraversal(v);
346                         graphLinkedObjects.breadthFirstTraversal(v);
347                     }
348                     */
349                 }//else if
350
351
352
353             }
354         }//for
355
356
357
358
359     }
360     public static boolean isNumber(String str) {
361         if (str == null || str.length() == 0) {
362             return false;
363         }
364         for (char c: str.toCharArray()) {
365             if (!(Character.isDigit(c))) {
366                 return false;
367             }
368         }
369         return true;
```

```
370         }
371     }
```

Here is my main method. First a binary search tree is created and populated with all 666 magic items. Then, the random 42 magic items were searched for in the binary search tree. As for the graphs, parsing the file took longer than expected in order to properly read each line. Vertices and edges were added into array lists which was necessary for the total number of vertices, along with using the lists to add the proper edges in the graph. This is where I failed to correctly implementing adding to each graph. Blocks of code that are commented out either resulted in errors, or would make the output way too long like printing the graphMatrix after every pass through. I plan to continue to work on this in order to properly and fully understand the nature of these graphs.