Tyler Paulsen
CSCI-351
Project 3

1. **Describe, at a high level but completely, what your analysis program(s)' inputs are, what the program(s) do, and what the outputs are.**

There is one program that takes in an input graph and a choice of 4 different graph analysis algorithms: betweenness centrality and creation of a minimum spanning tree, along with two helper functions: creating a graph file, and outputting a graph on the XY plane.

Main:
Takes in a graph file and any evaluations wanted. The graph file is read in, and the internal state of the graph is created. After graph creation, the program will iterate through the different evaluations, and output the results for each.

Betweenness Centrality:
The betweenness centrality is calculated by iterating over all the possible vertex pairs, and finding the path between each given pair. The path is then used to count all the vertices that were used in order to get from the source to destination. An array with the size equal to the number of vertices is used. Every time a vertex is passed through, the counter goes up. This process continues until all the pairs have been visited. The array is then iterated through, and the final result is calculated as: total_times_used_as_path[V] / total_paths_visited, where V is the vertex number.

Minimum Spanning Tree:
The minimum Spanning tree uses a priority queue structure while completing an algorithm that looks like Dijkstra's Algorithm calls Prim’s Algorithm.In order to get a Minimum spanning tree, a complete graph is created from the current graph in memory. Prim’s algorithm then loops through the priority queue until there is nothing left in the queue. The main outer loop checks to make sure the queue is not empty, if it is not, the first entry is taken from the queue, and the neighbors are iterated through. Each neighbor calculates the distance between the two nodes, and if that distance is less than the current distance saved for that vertex and it is still enqueue, the priority is increase.

2. **State the exact command line(s) for running the analysis program(s), including the Java main program class name and a description of each command line argument.**

Commands:
There is one program, and it only requires the graph file, and the functions that are needed to be ran. The program will read the graph, and iterate over each function specified.

Standard command:
java RoadNetworkAnalysis <graphfile> <graph measurement>
<graphfile> = graph file name.
<graph measurement> = [minSpanningTree | plotGraph | createGraphFile | betweenCentrality]

Tyler Paulsen
CSCI-351
Project 3

Creating a minimum spanning tree, creating the graph file, and creating a visual of the results:
java RoadNetworkAnalysis egypt-cities.txt minSpanningTree plotGraph createGraphFile

Do a betweenness measurement on a graph after creating a minimum spanning tree.
java RoadNetworkAnalysis egypt-cities.txt minSpanningTree betweenCentrality

3. **Put in the report the complete source code of your analysis program(s).**

```
import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.plot.Dots;
import edu.rit.numeric.plot.Plot;
import edu.rit.util.PriorityQueue;
import edu.rit.util.PriorityQueue.Item;

import java.awt.*;
import java.io.*;
import java.util.*;


/**
* Created by Tyler Paulsen on 4/9/2016.
* Class to compute various network science algorithms for a given graph.
*/
public class RoadNetworkAnalysis {
  private HashMap<Integer, Vertex> graph;
  private String graphfile;
  private int V;

  public static void main(String args[]) throws Exception{
     if (args.length < 2 ) usage();
     RoadNetworkAnalysis ga = new RoadNetworkAnalysis(args[0]);

     for(int i = 1; i < args.length; ++i){
        String arg = args[i];
        switch (arg){
           case "minSpanningTree":
              ga.minSpanningTree();
              break;
           case "plotGraph":
              ga.plotGraph();
              break;
           case "createGraphFile":
                ga.createGraphFile();
                break;
            case "betweenCentrality":
                ga.betweenCentrality();
                break;
            default:
                usage();
        }
    }
}

RoadNetworkAnalysis(String f) throws IOException{
    graphfile = f;
    graph = new HashMap<>();
    construct_graph(new File(f));
    //System.out.println(graph);
}

/**
 * creates a graphfile called: graphFile.txt using the graph formatted specified in:
 * https://cs.rit.edu/~ark/351/analysis/graphfile.shtml
 */
public void createGraphFile(){
    try {
        BufferedWriter writer = new BufferedWriter(
                new OutputStreamWriter(
                        new FileOutputStream("graphFile.txt", false),
                        "utf-8")
        );
        int E = 0;
        int V = 0;
        LinkedList<String> lines = new LinkedList<>();

        HashSet<HashSet<Integer>> edge = new HashSet<>();
        String l;
        //write the g tag <V> <E>
        HashSet<Integer> hs;
        for(Vertex v: graph.values()) {
            V += 1;
            lines.addFirst("d " + v.n + " " + v.x + " " + v.y);
            for(Integer e: v.edges) {
                l = "e " + v.n + " " + e;
                hs = new HashSet<>();
                hs.add(v.n);
                hs.add(e);
                if(!edge.contains(hs)) {
                    E += 1;
                    lines.addLast(l);
                    edge.add(hs);
                }
            }
        }
        lines.addFirst("g " + V + " " + E);

        for(String line: lines) {
            writer.write(line);
            writer.newLine();
        }
        writer.close();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
        System.exit(1);
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(1);
    }
  }
/**
* Construct a graph from a file. Must conform to the spec:
https://cs.rit.edu/~ark/351/analysis/graphfile.shtml
*  If the spec is not follow, results may vary.
* @param fileName - name of the graph file
*/
private void construct_graph(File fileName)throws IOException{
  // The name of the file to open.
  try {
      FileReader fileReader = new FileReader(fileName);
      BufferedReader bufferedReader = new BufferedReader(fileReader);

      String splitLine[] = bufferedReader.readLine().split(" ");

      while(!splitLine[0].equals("g")) {
          splitLine = bufferedReader.readLine().split(" ");
      }

      V = Integer.parseInt(splitLine[1]);
```

```
        for (int i = 0; i < V; ++i)
            graph.put(i, new Vertex(i));

        int v1,v2;
        float x,y,w;
        Vertex v;

        String line;
        //parse the weights
        while((line = bufferedReader.readLine()) != null) {
            splitLine = line.split(" ");
            //System.out.println(Arrays.toString(splitLine));
            if (splitLine[0].equals("v")) {
                v1 = Integer.parseInt(splitLine[1]);
                w = Float.parseFloat(splitLine[2]);
                v = graph.get(v1);
                v.w = w;
            }else if (splitLine[0].equals("d")) {
                v1 = Integer.parseInt(splitLine[1]);
                x = Float.parseFloat(splitLine[2]);
                y = Float.parseFloat(splitLine[3]);
                v = graph.get(v1);
                v.x = x;
                v.y = y;
            }else if(splitLine[0].equals("e")) {
                v1 = Integer.parseInt(splitLine[1]);
                v2 = Integer.parseInt(splitLine[2]);
                graph.get(v1).addAdjacent(v2);
                graph.get(v2).addAdjacent(v1);
            }
        }
        bufferedReader.close();
    }
    catch(FileNotFoundException ex) {
        System.out.println("Unable to open file: " + fileName);
        throw new FileNotFoundException();
    }
    catch(Exception ex) {
        System.out.println("Error parsing file: " + fileName);
        throw new IOException();
    }
}
```

Tyler Paulsen
CSCI-351
Project 3

```
/**
*  shortest path
* @param src node to start from
* @param dest node to find shortest path too
*/
private Path BFS(int src, int dest ){
   HashSet<Integer> seen = new HashSet<>();
   LinkedList<Path> queue = new LinkedList<>();
   Path A = new Path(src);
   queue.addLast(A);
   seen.add(A.id);
   while (!queue.isEmpty()) {
       A = queue.poll();
       for (Integer B : graph.get(A.id).edges) {
           if(B == dest) return new Path(B,A);
           if (!seen.contains(B)) {
               seen.add(B);
               queue.addLast(new Path(B, A));
           }
       }
   }
   return null;
}

// Class used in the breadth first search.
private class Path {
   int dist, id;
   Path parent;

   //constructor, keeps tract of the distance by adding one to the previous.
   Path(int _id, Path _parent) {
       id = _id;
       parent = _parent;
       dist = parent.dist + 1;
   }
   //initial constructor
   Path(int _id) {
       id = _id;
       dist = 1;
   }
   public HashSet<Integer> getPath(int dest){
       HashSet<Integer> path = new HashSet<>();
```

```
        if(dest != parent.id) {
            getPath(path, parent, dest);
        }
        return path;
    }
    private void getPath(HashSet<Integer> path, Path vertex, int dest){
        path.add(vertex.id);
        if(vertex.parent.id == dest)
            return;
        getPath(path, vertex.parent, dest);
    }
}

/**
* plots the graph.
*/
public void plotGraph(){

    ListXYSeries results = new ListXYSeries();
    Vertex src,dest;
    for (int i = 0; i < graph.size(); ++i) {
        src = graph.get(i);
        for(Integer A: src.edges){
            dest = graph.get(A);
            results.add(src.x, src.y);
            results.add(dest.x, dest.y);
        }
    }

    Plot plot = new Plot()
            .plotTitle("Graph Plot, graphfile = " + graphfile)
            .xAxisLength(650)
            .yAxisLength(650)
            .seriesDots(Dots.circle(5))
            .seriesColor(Color.RED)
            .segmentedSeries(results);
    plot.getFrame().setVisible(true);
}

/**
* Creates a complete graph of the current graph
*/
private void createCompleteGraph(){
    for(int i = 0; i < V; ++i){
        for( int k = i; k < V; ++k){
            graph.get(i).edges.add(k);
            graph.get(k).edges.add(i);
        }
    }
}
/**
     * uses Prim's algorithm to create a minimum spanning tree on the current graph.
     */
    public void minSpanningTree(){
        createCompleteGraph();
        PriorityQueue queue = new PriorityQueue();
        Distance D[] = new Distance[V];
        for(int i = 0; i<V; ++i){
            D[i] = new Distance(i, -1, Double.MAX_VALUE);
            queue.add(D[i]);
        }
        D[0].distance = 0.0f;
        Distance V;
        while( !queue.isEmpty()){
            V = (Distance)queue.remove();
            for(Integer W: graph.get(V.n).edges){
                if(D[W].enqueued() && distance(graph.get(W),graph.get(V.n)) < D[W].distance){
                    D[W].predecessor = V.n;
                    D[W].predecessor = V.n;
                    D[W].distance = distance(graph.get(W),graph.get(V.n));
                    D[W].increasePriority();
                }
            }
        }
        double total_distance = 0;
        //clear all the edges from the current graph, so we can add new ones.
        for(int i = 0; i < this.V; ++i) {
            graph.get(i).edges.clear();
            total_distance += D[i].distance;
        }
        System.out.printf("Total Distance: %f%n",total_distance);
        Vertex v1,v2;
        for(int i=0; i < this.V; ++i){
            if (D[i].predecessor == -1) continue;
            v1 = graph.get(i);
            v2 = graph.get(D[i].predecessor);
            v1.edges.add(D[i].predecessor);
            v2.edges.add(i);
        }

    }

    /**
     * calculated the betweenness centrality
     */
    public void betweenCentrality(){
        TreeMap<Double,LinkedList<Integer>> results = new
TreeMap<>(Collections.reverseOrder());
        int between[] = new int[V];
        for( int i = 0; i < V; ++i){
            between[i] = 0;
        }
        Path path;
        int total = 0;
        for( int src = 0; src < V; ++src){
            for(int dest = src+1; dest < V; ++dest) {
                if ( ( path = BFS(src, dest) ) == null) continue;

                HashSet<Integer> p = path.getPath(src);
                for(Integer i: p){
                    between[i] += 1;
                }
                total += 1;
            }
        }
        double result;
        LinkedList<Integer> ll;
        for(int i = 0; i < V; ++i){
            result = between[i] / (double)total;
            if(results.containsKey(result)){
                results.get(result).add(i);
            }else{
                ll = new LinkedList<>();
                ll.add(i);
                results.put(result, ll);
            }
        }
        System.out.println("Rank\tVertex\tbetweenness");
        int rank = 1;
        for(Map.Entry<Double,LinkedList<Integer>> entry : results.entrySet()) {
            for ( Integer vertex : entry.getValue()) {
                System.out.printf("%d\t%d\t%f%n", rank++, vertex, entry.getKey());
                graph.get(vertex).betweeness = entry.getKey();
                total += entry.getKey();
            }
        }
    }
    /**
     * used in prim's algo to store a vertex and edge.
     */
    private class Distance extends Item{
        int n,predecessor;
        double distance;
        Distance(int n, int predecessor, double distance){
            this.n = n;
            this.predecessor = predecessor;
            this.distance = distance;
        }
        public boolean equals(Object obj){
            return (Integer)obj == n;
        }

        @Override
        public boolean comesBefore(Item item) {
            return this.distance < ((Distance)item).distance;
        }
        public String toString(){
            return "N="+n + " p=" + predecessor + " D=" + distance;
        }
    }

    /**
     * get the euclidean distance for two points
     * @param a x,y cords of vertex
     * @param b x,y cords of vertex
     * @return distance
     */
    private double distance(Vertex a, Vertex b){
        return Math.sqrt(Math.pow(a.x - b.x, 2) + Math.pow(a.y - b.y, 2));
    }

    private class Vertex{
        int n;
        float x,y,w;
        double betweeness = -1;
        HashSet<Integer> edges;
        Vertex(int n){
            this.n = n;
            edges = new HashSet<>();
            x = y = 0.0f;
            w = 1.0f;
        }
        void addAdjacent(int v){
            edges.add(v);
        }

        public boolean equals(Object obj){
            return (Integer)obj == n;
        }
        public String toString(){
            return n + " : " + x + " , " + y;
        }
    }
    // Print a usage message and exit.
    private static void usage()
    {
        System.err.println ("Usage: java RoadNetworkAnalysis <graphfile> <graph
measurement>");
        System.err.println ("<graphfile> = graph file name");
        System.err.println ("<graph measurement> = [ minSpanningTree | plotGraph | " +
                "createGraphFile | betweenCentrality");
        throw new IllegalArgumentException();
    }

}
```

4. **Put in the report the contents of a file in Graph File Format containing the cities and roads in the Egyptian Empire, like the one for the Roman Empire. Also put in the report a plot showing the cities and roads in the Egyptian Empire, like the one for the Roman Empire. Include the exact command line(s) for your program(s) that you ran to answer this question.**

Command:

```
java RoadNetworkAnalysis egypt-cities.txt minSpanningTree plotGraph createGraphFile
```

Tyler Paulsen
CSCI-351
Project 3

```
g 200 199
d 199 -18.0236 -7.66248
d 198 8.57445 11.3847
d 197 -4.46946 17.4123
d 196 -12.1026 -7.63616
d 195 15.432 1.87333
d 194 2.17103 -5.92323
d 193 -3.48147 1.78585
d 192 13.8686 4.1823
d 191 17.5488 -4.0449
d 190 -8.28797 -17.1005
d 189 4.21659 10.639
d 188 -2.40383 -0.545177
d 187 -10.31 5.1892
d 186 8.52456 -6.63629
d 185 -5.56531 -14.1923
d 184 6.12812 -5.24011
d 183 5.0814 -6.18191
d 182 -10.5999 3.04321
d 181 -1.71185 2.07373
d 180 -14.1181 -12.7439
d 179 -4.62979 -12.3565
d 178 -1.80288 -14.3547
d 177 12.1946 15.4791
d 176 6.1876 -1.25214
d 175 0.501583 -2.27078
d 174 3.35115 -2.98725
d 173 -19.5887 -2.51334
d 172 -5.29642 2.12011
d 171 1.54197 2.65836
d 170 -5.18591 -12.1136
d 169 -3.85068 1.51167
d 168 6.23165 -2.53043
d 167 -15.4295 3.73886
d 166 -16.1293 2.33408
d 165 18.0659 -2.63652
d 164 0.80637 14.743
d 163 0.098278 -3.2304
d 162 5.69578 10.7648
d 161 6.31726 -13.3258
d 160 -8.92361 -17.5122
d 159 -2.23335 1.82041
```

Tyler Paulsen
CSCI-351
Project 3

```
d 158 -0.301406 1.47213
d 157 9.17678 -10.9941
d 156 -12.7361 7.82844
d 155 1.7099 0.444944
d 154 1.26783 1.30894
d 153 6.96846 -13.1173
d 152 9.31525 -2.25507
d 151 8.97852 3.69585
d 150 2.85016 2.13093
d 149 -0.0624049 4.58278
d 148 -4.36141 -10.7994
d 147 -3.97144 -10.7368
d 146 8.50867 -15.7433
d 145 -4.61095 1.3976
d 144 6.01202 -16.9944
d 143 -15.3703 11.7326
d 142 5.62832 2.56412
d 141 1.75211 3.49084
d 140 -7.17332 4.75287
d 139 -0.464344 3.59867
d 138 -3.53462 -10.1812
d 137 -2.45669 -18.4102
d 136 -2.44695 -2.16922
d 135 8.4039 -2.85103
d 134 11.4924 -12.5771
d 133 -16.9364 -1.80101
d 132 -5.80827 2.27374
d 131 -12.9876 2.49946
d 130 6.887 18.4488
d 129 3.33742 4.19354
d 128 0.249301 -12.2895
d 127 -9.82508 -6.62832
d 126 1.03019 -7.57558
d 125 1.63197 -2.60365
d 124 -13.3208 -13.9853
d 123 4.29406 1.28505
d 122 -15.5554 4.11732
d 121 -5.08337 1.48833
d 120 7.43752 -3.28053
d 119 -0.586094 14.5984
d 118 6.86351 -0.0141145
d 117 6.61391 -8.83125
d 116 4.69043 5.4568
```

Tyler Paulsen
CSCI-351
Project 3

```
d 115 8.585 -13.8326
d 114 10.9545 10.3073
d 113 16.4059 -2.1323
d 112 -11.1645 9.42646
d 111 7.71581 7.50644
d 110 -1.26872 0.342563
d 109 9.09669 3.34045
d 108 13.0399 1.47389
d 107 -0.496561 19.9859
d 106 -18.5279 2.60341
d 105 8.65539 11.0432
d 104 2.62175 6.8104
d 103 3.27979 -2.20611
d 102 17.3866 -2.60072
d 101 14.4618 -7.75749
d 100 -9.14798 -10.6501
d 99 -1.23767 -5.0712
d 98 2.05319 -12.0404
d 97 -0.506278 2.04269
d 96 -15.9231 -9.93038
d 95 7.90191 12.0468
d 94 -13.918 2.55105
d 93 -2.63311 8.69288
d 92 2.54218 1.22418
d 91 -1.45574 7.10204
d 90 -9.31903 -4.00654
d 89 0.701146 -10.1129
d 88 12.6706 -10.1302
d 87 -1.88569 11.3618
d 86 5.81203 11.7787
d 85 -4.1891 -11.7352
d 84 -3.34227 -7.47182
d 83 2.3186 11.3358
d 82 0.916972 0.61041
d 81 4.14253 12.6557
d 80 14.0095 3.5828
d 79 1.01957 0.145833
d 78 -1.18423 -0.839497
d 77 -3.12578 -1.25554
d 76 11.2213 6.43399
d 75 -17.5724 6.76661
d 74 4.31528 4.57171
d 73 -1.85063 -0.797475
```

Tyler Paulsen
CSCI-351
Project 3

```
d 72 -1.57614 -0.904914
d 71 3.711 2.93542
d 70 -5.20615 14.0361
d 69 13.8373 14.3964
d 68 6.73032 -1.33189
d 67 -0.28493 11.1226
d 66 -1.30629 -5.71431
d 65 1.5781 2.56362
d 64 7.44207 5.4476
d 63 -2.13447 3.5999
d 62 1.25402 -4.32587
d 61 -14.0285 13.0744
d 60 -12.4406 6.10013
d 59 1.77975 10.1789
d 58 17.8182 -0.085807
d 57 2.544 -1.93802
d 56 10.2053 -5.33742
d 55 4.94612 5.97579
d 54 2.52237 -6.32603
d 53 -4.98767 -9.95043
d 52 7.10874 9.88424
d 51 -18.1448 -7.73453
d 50 6.62231 -12.7654
d 49 2.0442 3.05175
d 48 6.12416 -6.64726
d 47 -8.11588 14.9199
d 46 -9.62969 6.61208
d 45 11.1081 -12.1008
d 44 2.08053 -0.709623
d 43 -5.03105 -10.6151
d 42 13.1986 11.0576
d 41 0.288905 2.59327
d 40 10.6034 9.1064
d 39 -14.7397 7.65741
d 38 -11.312 3.24297
d 37 14.3652 -11.2637
d 36 -1.03871 1.34506
d 35 -6.79134 -1.94784
d 34 5.8812 -11.5094
d 33 4.6887 10.3691
d 32 12.0237 -8.46813
d 31 10.7061 7.85033
d 30 -13.3078 -12.035
```

Tyler Paulsen
CSCI-351
Project 3

```
d 29 -7.84394 8.40742
d 28 10.262 -12.3961
d 27 -5.2002 -3.05843
d 26 11.1158 -8.405
d 25 7.02632 -5.50792
d 24 -1.88856 2.28969
d 23 -2.48731 2.74142
d 22 8.78035 16.9592
d 21 16.1915 -7.49015
d 20 16.7302 2.1948
d 19 -5.20532 -2.2513
d 18 10.0259 -4.27128
d 17 4.95673 -4.78604
d 16 0.372065 10.4306
d 15 -0.310026 4.84681
d 14 -4.1214 1.89049
d 13 -5.71363 12.156
d 12 -7.88723 -3.79586
d 11 -19.1375 -1.30398
d 10 -1.26193 -0.389156
d 9 -2.07661 -0.996556
d 8 -1.84183 1.41522
d 7 -10.5291 -1.45591
d 6 -7.9367 15.8426
d 5 4.40599 -4.48003
d 4 6.5639 -2.66122
d 3 -15.9191 -2.80382
d 2 -4.19318 -10.1176
d 1 0.374516 11.8515
d 0 0.0 0.0
e 0 79
e 1 67
e 1 119
e 2 138
e 2 147
e 3 133
e 4 120
e 4 168
e 5 17
e 5 174
e 6 47
e 7 90
e 8 36
```

```
e 8 159
e 9 73
e 9 188
e 10 78
e 10 110
e 11 106
e 11 133
e 11 173
e 12 35
e 12 90
e 13 70
e 13 87
e 14 145
e 14 169
e 15 91
e 15 149
e 16 59
e 16 67
e 17 183
e 17 184
e 18 56
e 18 152
e 19 27
e 19 35
e 19 77
e 20 58
e 20 195
e 21 101
e 22 130
e 22 177
e 23 24
e 23 63
e 24 181
e 25 184
e 25 186
e 26 32
e 28 45
e 28 115
e 28 157
e 29 46
e 30 100
e 30 180
e 31 40
```

Tyler Paulsen
CSCI-351
Project 3

```
e 31 76
e 32 88
e 32 101
e 33 162
e 33 189
e 34 50
e 34 117
e 36 110
e 36 158
e 37 88
e 38 131
e 38 182
e 39 75
e 39 143
e 39 156
e 40 114
e 41 97
e 41 139
e 41 171
e 42 69
e 42 114
e 43 53
e 43 148
e 44 57
e 44 155
e 45 88
e 45 134
e 46 140
e 46 187
e 47 70
e 48 117
e 48 183
e 49 141
e 49 150
e 49 171
e 50 153
e 50 161
e 51 199
e 52 105
e 52 111
e 52 162
e 54 126
e 54 194
```

```
e 55 64
e 55 104
e 55 116
e 56 186
e 57 103
e 57 125
e 58 113
e 59 83
e 60 156
e 60 187
e 61 143
e 62 163
e 62 194
e 64 111
e 64 151
e 65 171
e 66 84
e 66 99
e 67 87
e 68 118
e 68 176
e 69 177
e 70 197
e 71 129
e 71 150
e 72 73
e 72 78
e 74 116
e 74 129
e 76 192
e 77 136
e 77 188
e 79 82
e 79 155
e 80 108
e 80 192
e 80 195
e 81 86
e 82 154
e 83 189
e 84 138
e 85 148
e 85 179
```

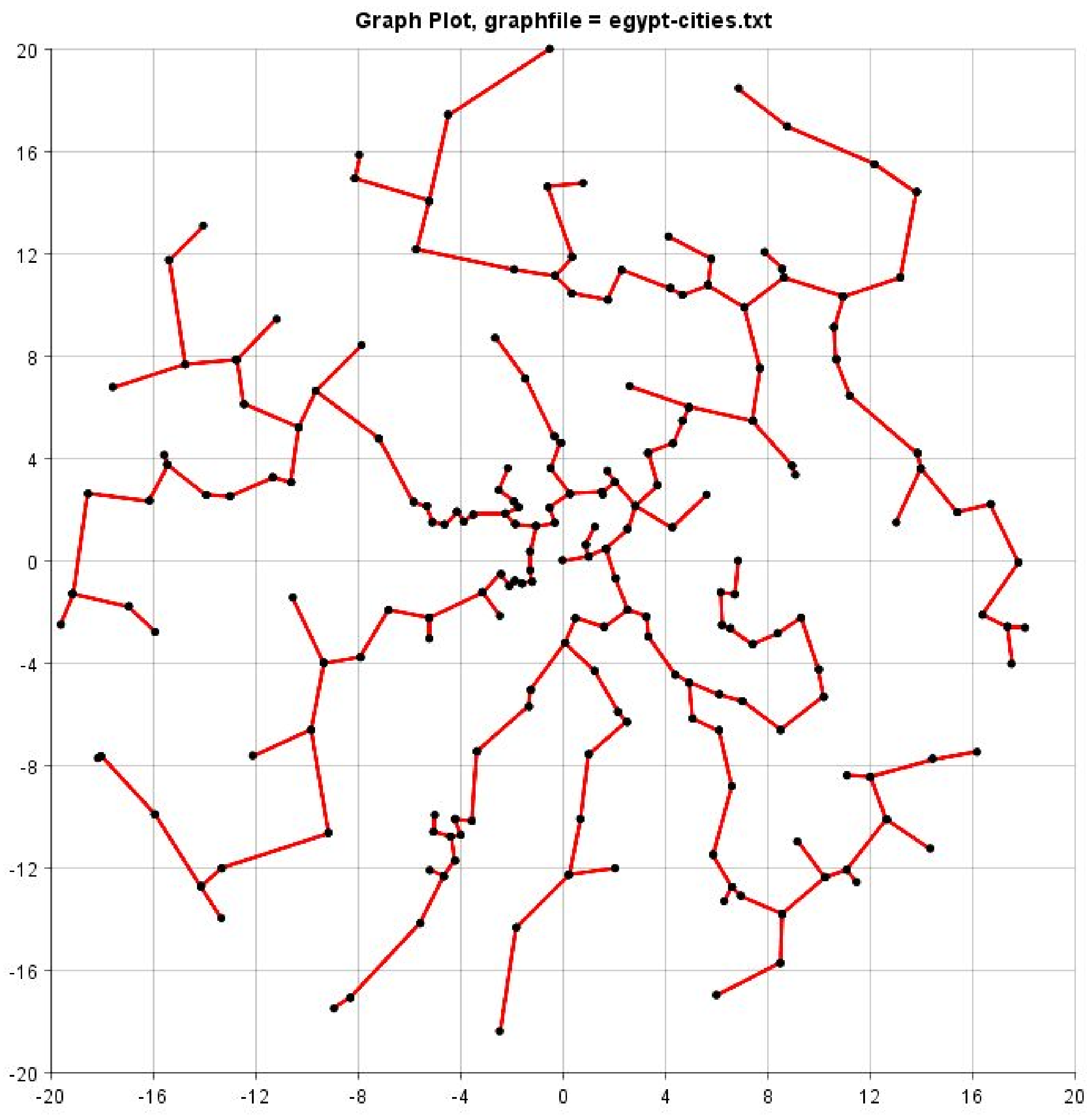
Tyler Paulsen
CSCI-351
Project 3

```
e 86 162
e 89 126
e 89 128
e 90 127
e 91 93
e 92 150
e 92 155
e 94 131
e 94 167
e 95 198
e 96 180
e 96 199
e 97 158
e 98 128
e 99 163
e 100 127
e 102 113
e 102 165
e 102 191
e 103 174
e 105 114
e 105 198
e 106 166
e 107 197
e 109 151
e 112 156
e 115 146
e 115 153
e 119 164
e 120 135
e 121 145
e 121 172
e 122 167
e 123 142
e 123 150
e 124 180
e 125 175
e 127 196
e 128 178
e 132 140
e 132 172
e 135 152
e 137 178
```

Tyler Paulsen
CSCI-351
Project 3

```
e 139 149
e 144 146
e 147 148
e 159 181
e 159 193
e 160 190
e 163 175
e 166 167
e 168 176
e 169 193
e 170 179
e 179 185
e 182 187
e 185 190
```

Tyler Paulsen
CSCI-351
Project 3



5. **What is the total distance of the roads in the Egyptian Empire, such that all the cities are linked together and the total distance is as small as possible? Include the exact command line(s) for your program(s) that you ran to answer this question.**

Command:

```
java RoadNetworkAnalysis egypt-cities.txt minSpanningTree
```

The total distance of all the roads in the Egyptian Emire is: 314.057627

Tyler Paulsen
CSCI-351
Project 3

6. **What are the top-40-ranked cities in the Egyptian Empire with respect to betweenness centrality? Include a table of data to justify your answer. Include the exact command line(s) for your program(s) that you ran to answer this question.**

Command:

```
java RoadNetworkAnalysis egypt-cities.txt minSpanningTree betweenCentrality
```

Output:

```
Rank  Vertex betweenness
1     150    0.662814
2     57     0.484372
3     155    0.467035
4     41     0.463467
5     49     0.463065
6     36     0.462513
7     171    0.457236
8     92     0.456683
9     44     0.437688
10    97     0.426834
11    158    0.423015
12    71     0.379296
13    129    0.374372
14    74     0.369347
15    116    0.364221
16    55     0.361307
17    64     0.352563
18    52     0.346533
19    111    0.331357
20    103    0.294874
21    8      0.288442
22    174    0.288442
23    17     0.288342
24    159    0.287940
25    5      0.281910
26    193    0.247739
27    169    0.240603
28    14     0.233367
29    125    0.226030
30    145    0.226030
31    121    0.218593
32    175    0.218593
33    163    0.217839
34    172    0.211055
35    110    0.203417
```

36	132	0.203417
37	10	0.195678
38	140	0.195678
39	105	0.189749
40	46	0.188844

7. **What is the betweenness centrality value and the rank of the Egyptian capital city, Alexandria? Will Pharaoh be separating you from your head based on your answer?**

The centrality value of the Egyptian capital city is: 0.0, and the rank of the city is tied for last place at rank: 157. The pharaoh will most certainly be separating my head because the capitol is not easily accessible.

8. **Write a paragraph describing what you learned from this project.**

When looking at the output of the graph, you can see that the program has zero bias. The graph could have 30 vertices all around the middle node, but just have enough distance between it to not make it a central point. This algorithm would create odd looking road layouts if used in real life due to the fact that it only cares about the shortest distance, and now how popular a city is.