CSC 226: Lab 4

Summer 2018

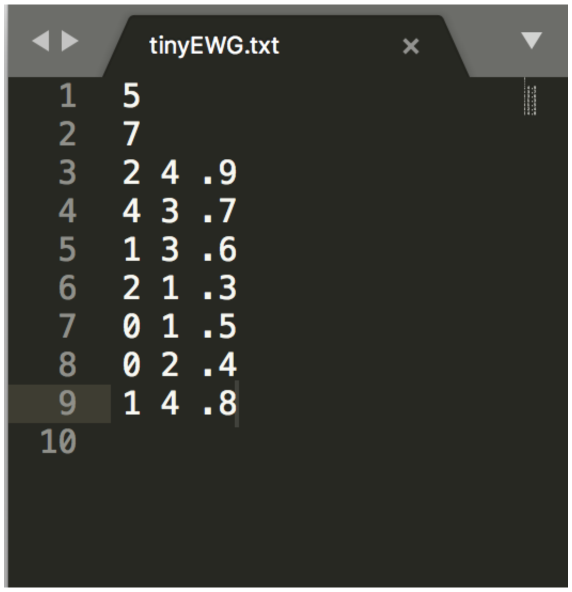
1) Create a new Java Project in Eclipse (in the default workspace) and name it MST.
2) Follow the steps described in algs4.pdf to include algs4.jar for this project.
3) Create a new Java class and name it Main.
4) Import all classes in the package edu.princeton.cs.algs4.
5) Write the main function definition. Main.java should now look like as follows:

```
import edu.princeton.cs.algs4.*;

public class Main {
    public static void main(String[] args) {

    }
}
```

6) The contents of tinyEWG.txt which defines an edge-weighted graph is as follows:



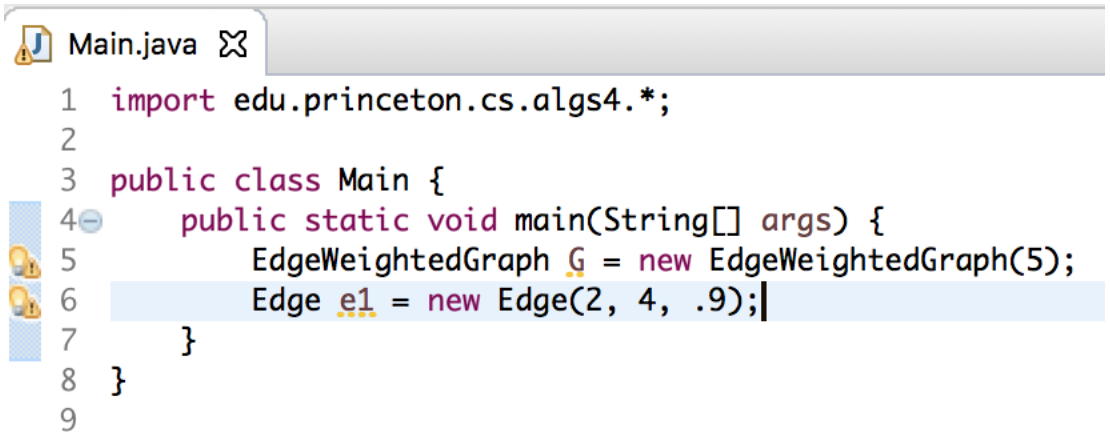
```
 1  5
 2  7
 3  2 4 .9
 4  4 3 .7
 5  1 3 .6
 6  2 1 .3
 7  0 1 .5
 8  0 2 .4
 9  1 4 .8
10
```

Note:
Line 1: the number of vertices of the graph. In this graph there are 5 vertices and they are named/numbered as 0, 1, 2, 3, and 4.

Line 2: the number of weighted edges of the graph. In this graph there are 7 weighted edges.

Lines 3-9: each line contains the description of a weighted edge. For example, Line 3 describes an undirected weighted edge from Vertex 2 to Vertex 4 with a weight of .9.

7. Create an EdgeWeightedGraph of 5 vertices in the main function of Main.java.
8. Create an Edge between Vertex 2 and Vertex 4 with a weight of .9. At this point Main.java should look like as follows:



```
import edu.princeton.cs.algs4.*;

public class Main {
    public static void main(String[] args) {
        EdgeWeightedGraph G = new EdgeWeightedGraph(5);
        Edge e1 = new Edge(2, 4, .9);
    }
}

```

9. Create six other edges as described in Lines 4-9 of tinyEWG.txt.
10. Insert all seven edges in the graph G. For example, edge e1 is added to the graph G as follows:

```
G.addEdge(e1);
```

11. Draw the graph on a paper.

We will now compute the minimal spanning tree for the graph G using Prim's algorithm.

12. Declare a Boolean array "marked" to keep track of the vertices that are in the tree.

```
boolean[] marked = new boolean[G.V()];
```

13. Declare a Queue "mst" to store the edges of the tree.

```
Queue<Edge> mst = new Queue<Edge>();
```

14. Declare a minimum priority queue to store the "crossing" edges. A crossing edge is one whose one end point/vertex is marked, and the other end point/vertex is not marked.

```
MinPQ<Edge> pq = new MinPQ<Edge>();
```

Proposition. (Cut Property, see text book, page 606) Let A and B be two disjoint subsets of the set of vertices V form a partition of V such that the vertices of A are marked (i.e., currently in the tree) and that the vertices of B are not marked. Then the crossing edge of minimum weight is in MST of the graph.

Note that for graph of V vertices, the spanning tree will contain V-1 edges.

15. We will start to grow a tree from Vertex 0. That is, Vertex 0 is in the tree now. So, we will mark it.

```
marked[0] = true;
```

16. Insert all edges of the graph that are incident to Vertex 0 in the minimum priority queue "pq".

```
for (Edge e : G.adj(0)) {
    pq.insert(e);
}
```

17. Extract the minimum edge from "pq".

```
Edge se1 = pq.delMin();
```

18. Check whether this edge is a crossing edge.

```
int u = se1.either();
int v = se1.other(u);
if ( (!marked[u] && marked[v]) || (marked[u] && !marked[v]) ) {
```

The "if" condition checked whether the edge is a crossing edge or not. A simpler "if" condition that might work is as follows:

```
if (!marked[u] || !marked[v]) {
```

Why do you think that this if condition will work?
According to the "Cut Property" proposition this must be in the minimum spanning tree. So, we will store it in the queue "mst".

```
mst.enqueue(se1);
```

19. Now one of the end vertices of this edge isn't marked. So, we will mark it. That means the most recently marked vertex is in the spanning tree. We will then add all of

its incident edges that are crossing (the other end vertex being unmarked) to the minimum priority queue “pq”.

20. Code for Steps 18 and 19 will now look like as follows:

```
int u = se1.either();
int v = se1.other(u);
if (!marked[u] || !marked[v]) {
     mst.enqueue(se1);
     if (!marked[u]) {
          marked[u] = true;
          for (Edge e : G.adj(u)) {
               if (!marked[e.other(u)]) pq.insert(e);
          }
     }
     if (!marked[v]) {
          marked[v] = true;
          for (Edge e : G.adj(v)) {
               if (!marked[e.other(v)]) pq.insert(e);
          }
     }
}
```

21. On a paper write down the contents of “marked”, “mst”, and “pq” (after performing Step 20).
22. Now repeat Steps 17, 18, 19, 20. The code for this step is as follows:

```
Edge se2 = pq.delMin();
u = se2.either();
v = se2.other(u);
if (!marked[u] || !marked[v]) {
    mst.enqueue(se2);
    if (!marked[u]) {
        marked[u] = true;
        for (Edge e : G.adj(u)) {
            if (!marked[e.other(u)]) pq.insert(e);
        }
    }

    if (!marked[v]) {
        marked[v] = true;
        for (Edge e : G.adj(v)) {
            if (!marked[e.other(v)]) pq.insert(e);
        }
    }
}
```

23. On a paper write down the contents of “marked”, “mst”, and “pq” (after performing Step 22).
24. Now repeat Steps 17, 18, 19, 20 like Step 22.

25. On a paper write down the contents of “marked”, “mst”, and “pq” (after performing Step 22).
Question: Was the minimum weight edge of pq extracted on this step a valid crossing edge?
The queue “pq” may contain invalid edges. This is because as new vertices are added to the spanning tree some of the edges in the queue can become invalid because the unmarked vertices of these edge get marked (added to the spanning tree).

25. Repeat Steps 24, and 25 two more times.
26. Now print the edges of the minimum spanning tree and the weight.

```
double weight = 0;
for (Edge e : mst) {
    System.out.println(e.toString());
    weight += e.weight();
}
System.out.println("weight: " + weight);
```

27. Compute the spanning tree manually on a paper. Do you get the same result?
28. We have lots of repeated code. So, we comment out all the code from Step 17 to Step 25. And add the following code above the code of 26.

```
while(!pq.isEmpty()) {
    Edge e = pq.delMin();
    int s = e.either();
    int t = e.other(s);
    if (marked[s] && marked[t]) continue;
    mst.enqueue(e);
    if (!marked[s]) {
        marked[s] = true;
        for (Edge f : G.adj(s)) {
            if (!marked[f.other(s)]) pq.insert(f);
        }
    }
    if (!marked[t]) {
        marked[t] = true;
        for (Edge f : G.adj(t)) {
            if (!marked[f.other(t)]) pq.insert(f);
        }
    }
}
```

29. Once you know what’s going on in Prim’s algorithm, here’s how you can compute the minimum spanning tree and the weight of a graph defined in a text file in the format described above.

```java
public static void main(String[] args) {
    In in = new In(args[0]);
    EdgeWeightedGraph G = new EdgeWeightedGraph(in);
    LazyPrimMST mst = new LazyPrimMST(G);
    for (Edge e : mst.edges()) {
        System.out.println(e);
    }
    System.out.println("weight: " + mst.weight());
}
```