CSC 226 : Summer 2019 Lab 5

1 Implementing Lazy Prim's Algorithm

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 *Prim.java*.
4. Import all classes in the package *edu.princeton.cs.algs4*.
5. Write the main function definition.

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

public class Prim{
        public static void main(String[] args){
        }
}
```

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

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

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 Prim.java.

8. Create an Edge between Vertex 2 and Vertex 4 with a weight of .9.

   ```
   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 using he *addEdge* method.

    ```
    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.

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. On a paper write down the contents of *marked*, *mst*, and *pq*.

21. Now repeat Steps 17, 18, 19.

22. On a paper write down the contents of *marked*, *mst*, and *pq* (after performing Step 21).

23. Now repeat Steps 17, 18, 19 like Step 21.

24. On a paper write down the contents of *marked*, *mst*, and *pq* (after performing Step 21).

    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 23, and 24 two more times.

26. Now print the edges of the minimum spanning tree and the weight.

27. Compute the spanning tree manually on a paper. Do you get the same result?

28. We have lots of repeated code. So, try this while lopp to remove the repetition.

```
while(!pq.isEmpty()){
        Edge e = pq.delMin();
        ....
        .....
}
```

29. Once you know what is going on in Prim's algorithm,try to use the "LazyPrimMST" class from algs4.jar.