**Lab 12 Report**

**CS303L-L3 Algorithms and Data Structures**

**Sam Lazrak**

**Spring Semester 2018**

**Objectives:**

* Implement weighted graph and Prim’s Algorithm for minimum spanning trees

**In-class Assignment:**

1. Implement a weighted graph class from the Graph.java used previously. Graph.java uses integer value for storing an edge. Instead of using integer value for storing edges, create an “Edge” class that holds information of edge. Edge class has following attributes (not limited to):

Vertex v1, v2

int edgeID

int edgeWeight

1. Change the adjacency list to hold the vertex and edge weight. 3. Write a driver program, which reads input files mediumGraph.txt, LargeGraph.txt and XtraLargeGraph.txt (on Canvas) and display the weighted graphs by printing adjacency list.

**Homework Assignment:**

1. Implement Prim’s algorithm (see below). Hint: for the following code, you need to use min-heap queue. In Java, you can use PriorityQueue API.
2. Write a driver program, which reads input files mediumGraph.txt, LargeGraph.txt and XtraLargeGraph.txt (on Canvas) and runs Prim’s algorithm on each of them to find a Minimum Spanning Tree within these graphs. Record the times required for each of these graphs
3. Write a report containing your observations running Prim’s algorithm for finding Minimum Spanning Trees.

The following source code was written:

public class Edge {

private int vertexOne;

private int vertexTwo;

private double edgeWeight;

public Edge(int u, int v, double weight) {

this.vertexOne = u;

this.vertexTwo = v;

this.edgeWeight = weight;

}

public int getVertexOne() {

return vertexOne;

}

public int getVertexTwo() {

return vertexTwo;

}

public double getWeight() {

return edgeWeight;

}

public String toString() {

return vertexOne + " <- " + edgeWeight + " -> " + vertexTwo + " ";

} }

public class Graph {

public static void main(String[] args) throws IOException {

BufferedReader read = new BufferedReader(new FileReader("tinyDG.txt"));

Graph g = new Graph(read);

System.out.println(g.tostring());

System.out.println("Minimum Spanning Tree with tiny input: ");

System.out.println(g.Prim(g).toString());

BufferedReader rea = new BufferedReader(new FileReader("mediumDG.txt"));

Graph grap = new Graph(rea);

long medTime = System.nanoTime();

grap.Prim(grap);

System.out.println("\n" + "Prim's Algorithm with medium input took: " + (System.nanoTime() - medTime) + " nanoseconds.");

BufferedReader r = new BufferedReader(new FileReader("largeDG.txt"));

Graph gr = new Graph(r);

long largeTime = System.nanoTime();

gr.Prim(gr);

System.out.println("\n" + "Prim's Algorithm with large input took: " + (System.nanoTime() - largeTime) + " nanoseconds.");

BufferedReader re = new BufferedReader(new FileReader("XtraLargeDG.txt"));

Graph graph = new Graph(re);

long xLarTime = System.nanoTime();

graph.Prim(graph);

System.out.println("\n" + "Prim's Algorithm with medium input took: " + (System.nanoTime() - xLarTime) + " nanoseconds.");

}

public static int V;

public static int E;

public LinkedList<Edge>[] adj;

public Graph()

{

V = 0;

E = 0;

}

@SuppressWarnings("unchecked")

public Graph(BufferedReader reader) throws IOException

{

String line;

line = reader.readLine();

V = Integer.parseInt(line);

line = reader.readLine();

E = Integer.parseInt(line);

adj = new LinkedList[V];

for (int v = 0; v < V; v++) {

adj[v] = new LinkedList<Edge>();

}

while ((line = reader.readLine()) != null) {

int tempV1, tempV2;

double weight;

StringTokenizer st = new StringTokenizer(line, " ");

tempV1 = Integer.parseInt(st.nextToken());

tempV2 = Integer.parseInt(st.nextToken());

weight = Double.parseDouble(st.nextToken());

addEdge(tempV1, tempV2, weight);

}

}

public void addEdge(int v, int w, double weight) {

Edge e1 = new Edge(v, w, weight);

adj[e1.getVertexOne()].add(e1);

adj[e1.getVertexTwo()].add(e1);

}

public String tostring()

{

String s = new String();

s = "There are "+V+" vertices and "+E+" edges\n\n";

for(int i=0;i<V;i++)

{

s = s+i+": ";

for(int j = 0; j<adj[i].size();j++)

{

s = s+adj[i].get(j).toString() +" ";

}

s = s+"\n";

}

return s;

}

public ArrayList<Edge> Prim(Graph G) {

ArrayList<Edge> mst = new ArrayList<Edge>();

PriorityQueue<Edge> pq = new PriorityQueue<Edge>((Edge o1, Edge o2) -> {

Edge first = o1;

Edge second = o2;

if(first.getWeight() < second.getWeight()) {

return -1;

}

if(first.getWeight() > second.getWeight()) {

return 1;

}

else return 0;

});

for(Edge e: G.adj[0]) {

pq.add(e);

}

boolean[] marked = new boolean [G.V];

marked[0] = true;

while(!pq.isEmpty()) {

Edge e = pq.peek();

pq.poll();

if(marked[e.getVertexOne()] && marked[e.getVertexTwo()]) continue;

marked[e.getVertexOne()] = true;

for( Edge edge: G.adj[e.getVertexTwo()]) {

if(!marked[edge.getVertexTwo()]) {

pq.add(edge);

}

}

marked[e.getVertexTwo()] = true;

mst.add(e);

}

return mst;

}

}

The output was:

There are 8 vertices and 15 edges

0: 0 <- 0.38 -> 4 0 <- 0.26 -> 2 6 <- 0.58 -> 0

1: 5 <- 0.32 -> 1 1 <- 0.29 -> 3

2: 0 <- 0.26 -> 2 2 <- 0.34 -> 7 6 <- 0.4 -> 2

3: 7 <- 0.39 -> 3 1 <- 0.29 -> 3 3 <- 0.52 -> 6

4: 4 <- 0.35 -> 5 5 <- 0.35 -> 4 4 <- 0.37 -> 7 0 <- 0.38 -> 4 6 <- 0.93 -> 4

5: 4 <- 0.35 -> 5 5 <- 0.35 -> 4 5 <- 0.28 -> 7 7 <- 0.28 -> 5 5 <- 0.32 -> 1

6: 6 <- 0.4 -> 2 3 <- 0.52 -> 6 6 <- 0.58 -> 0 6 <- 0.93 -> 4

7: 4 <- 0.37 -> 7 5 <- 0.28 -> 7 7 <- 0.28 -> 5 7 <- 0.39 -> 3 2 <- 0.34 -> 7

Minimum Spanning Tree with tiny input:

0 <- 0.26 -> 2

2 <- 0.34 -> 7

5 <- 0.28 -> 7

4 <- 0.37 -> 7

7 <- 0.39 -> 3

1 <- 0.29 -> 3

6 <- 0.4 -> 2

Prim's Algorithm with medium input took: 7959226 nanoseconds.

Prim's Algorithm with large input took: 34240770 nanoseconds.

Prim's Algorithm with extra large input took: 160279497 nanoseconds.

Analysis:

First, I print the vertices and the edges along with the edge weight.

Then, I run Prim’s algorithm to get a minimum spanning tree.

The running time for the different input files varies with how large the file is. The larger the file, the longer it takes to run Prim’s Algorithm.