Algorithm Analysis and Data Structures: Assignment 4 – Dijkstra’s Algorithm

1. Make a graph.  you can use any representation. The graph must have at least 10 nodes and 15 edges.  undirected graph.

2. Run Dijkstra's algorithm.

Code:

**package** Graphs;

**import** java.util.\*;

**public** **class** DijkstrasUsingAdjacencyList {

**public** **class** Edge {

**public** **final** Vertex dest;

**public** **final** **int** weight;

**public** Edge(Vertex dest, **int** weight){

**this**.dest = dest;

**this**.weight = weight;

}

}

**public** **class** Vertex **implements** Comparable<Vertex> {

**public** **final** String name;

**public** ArrayList<Edge> neighbours;

**public** LinkedList<Vertex> path;

**public** **int** minDistance = Integer.***MAX\_VALUE***;

**public** Vertex previous;

**public** **int** compareTo(Vertex other){

**return** Integer.*compare*(minDistance,other.minDistance);

}

**public** Vertex(String name){

**this**.name = name;

neighbours = **new** ArrayList<Edge>();

path = **new** LinkedList<Vertex>();

}

**public** String toString(){

**return** name;

}

}

**public** **class** Graph {

**private** ArrayList<Vertex> vertices;

**public** Graph(**int** numberVertices){

vertices = **new** ArrayList<Vertex>(numberVertices);

**for**(**int** i=0;i<numberVertices;i++){

vertices.add(**new** Vertex("n"+Integer.*toString*(i)));

}

}

**public** **void** addEdge(**int** src, **int** dest, **int** weight){

Vertex v = vertices.get(src);

Edge newEdge = **new** Edge(vertices.get(dest),weight);

v.neighbours.add(newEdge);

}

**public** ArrayList<Vertex> getVertices() {

**return** vertices;

}

**public** Vertex getVertex(**int** vert){

**return** vertices.get(vert);

}

**public** **void** DisplayGraph() {

HashMap<String, ArrayList<String>> adjlst = **new** HashMap<>();

**for**(Vertex v:**this**.getVertices()){

ArrayList<String> allEdges = **new** ArrayList<>();

**for**(Edge e: v.neighbours)

{

allEdges.add(e.dest.toString());

}

adjlst.put(v.name,allEdges);

}

Set<String> set = adjlst.keySet();

Iterator<String> i = set.iterator();

**while**(i.hasNext())

{

Object key = i.next();

Object value = adjlst.get(key);

System.***out***.println(key + " -> " +value);

}

}

}

**public** **static** **void** main(String[] arg){

DijkstrasUsingAdjacencyList obj = **new** DijkstrasUsingAdjacencyList();

// Creating a new graph.

Graph g = obj.**new** Graph(10);

//Adding the new edges

g.addEdge(0, 1, 9);

g.addEdge(0, 5, 14);

g.addEdge(0, 6, 15);

g.addEdge(1, 2, 24);

g.addEdge(2, 4, 2);

g.addEdge(2, 7, 19);

g.addEdge(3, 2, 6);

g.addEdge(3, 7, 6);

g.addEdge(4, 3, 11);

g.addEdge(4, 7, 16);

g.addEdge(5, 2, 18);

g.addEdge(5, 4, 30);

g.addEdge(5, 6, 5);

g.addEdge(6, 4, 20);

g.addEdge(6, 7, 44);

g.addEdge(7, 8, 10);

g.addEdge(8, 9, 5);

//as it is an undirected graph, we will have the same edge between the same

g.addEdge(1, 0, 9);

g.addEdge(5, 0, 14);

g.addEdge(6, 0, 15);

g.addEdge(2, 1, 24);

g.addEdge(4, 2, 2);

g.addEdge(7, 2, 19);

g.addEdge(2, 3, 6);

g.addEdge(7, 3, 6);

g.addEdge(3, 4, 11);

g.addEdge(7, 4, 16);

g.addEdge(2, 5, 18);

g.addEdge(4, 5, 30);

g.addEdge(6, 5, 5);

g.addEdge(4, 6, 20);

g.addEdge(7, 6, 44);

g.addEdge(8, 7, 10);

g.addEdge(9, 8, 5);

System.***out***.println("Undirected Graph using Adjacency List:\n");

g.DisplayGraph();

// Calculating Dijkstra

obj.calculateDijkstra(g.getVertex(0));

System.***out***.println("\nDijkstra's Algorithm: \n");

**for**(Vertex v:g.getVertices()){

System.***out***.print("Vertex - "+v+" , Distance - "+ v.minDistance+" , Path - ");

**for**(Vertex pathvert:v.path) {

System.***out***.print(pathvert+" ");

}

System.***out***.println(""+v);

}

}

**public** **void** calculateDijkstra(Vertex source){

//Takes the unvisited nodes with minimum weight and visits all the neighbours and updates the respective distances

source.minDistance = 0;

PriorityQueue<Vertex> queue = **new** PriorityQueue<Vertex>();

queue.add(source);

**while**(!queue.isEmpty()){

Vertex u = queue.poll();

**for**(Edge neighbour:u.neighbours){

**int** newDist = u.minDistance+neighbour.weight;

**if**(neighbour.dest.minDistance>newDist){

// Remove the node from the queue to update the distance value

queue.remove(neighbour.dest);

neighbour.dest.minDistance = newDist;

// Take the path visited till now and add the new nodes

neighbour.dest.path = **new** LinkedList<Vertex>(u.path);

neighbour.dest.path.add(u);

//Reenter the node with new distance

queue.add(neighbour.dest);

}

}

}

}

}

Instructions to compile:

1. Create a new java project in eclipse named Graphs
2. Create a new package in the source folder named Graphs
3. Create a java class in the same package with the name DijkstrasUsingAdjacencyList
4. Place the code and save
5. Run the code

Screen Shot of the output

