**Algoritma Djikstra Penerapan Di Java Netbeans**

Nama : Imam Teguh

NPM / Kelas : 1414052 / SI

Source Code :

import java.util.PriorityQueue;

import java.util.List;

import java.util.ArrayList;

import java.util.Collections;

class Vertex implements Comparable<Vertex>

{

    public final String name;

    public Edge[] adjacencies;

    public double minDistance = Double.POSITIVE\_INFINITY;

    public Vertex previous;

    public Vertex(String argName) { name = argName; }

    public String toString() { return name; }

    public int compareTo(Vertex other)

    {

        return Double.compare(minDistance, other.minDistance);

    }

}

class Edge

{

    public final Vertex target;

    public final double weight;

    public Edge(Vertex argTarget, double argWeight)

    { target = argTarget; weight = argWeight; }

}

public class Dijkstra

{

    public static void computePaths(Vertex source)

    {

        source.minDistance = 0.;

        PriorityQueue<Vertex> vertexQueue = new PriorityQueue<Vertex>();

      vertexQueue.add(source);

       while (!vertexQueue.isEmpty()) {

           Vertex u = vertexQueue.poll();

            // Visit each edge exiting u

            for (Edge e : u.adjacencies)

            {

                Vertex v = e.target;

                double weight = e.weight;

                double distanceThroughU = u.minDistance + weight;

              if (distanceThroughU < v.minDistance) {

                  vertexQueue.remove(v);

                  v.minDistance = distanceThroughU ;

                  v.previous = u;

                  vertexQueue.add(v);

              }

            }

        }

    }

    public static List<Vertex> getShortestPathTo(Vertex target)

    {

        List<Vertex> path = new ArrayList<Vertex>();

        for (Vertex vertex = target; vertex != null; vertex = vertex.previous)

            path.add(vertex);

        Collections.reverse(path);

        return path;

    }

    public static void main(String[] args)

    {

       Vertex v0 = new Vertex("Redvile");

       Vertex v1 = new Vertex("Blueville");

       Vertex v2 = new Vertex("Greenville");

       Vertex v3 = new Vertex("Orangeville");

       Vertex v4 = new Vertex("Purpleville");

       v0.adjacencies = new Edge[]{ new Edge(v1, 5),

                                    new Edge(v2, 10),

                               new Edge(v3, 8) };

       v1.adjacencies = new Edge[]{ new Edge(v0, 5),

                                    new Edge(v2, 3),

                                    new Edge(v4, 7) };

       v2.adjacencies = new Edge[]{ new Edge(v0, 10),

                               new Edge(v1, 3) };

       v3.adjacencies = new Edge[]{ new Edge(v0, 8),

                                    new Edge(v4, 2) };

       v4.adjacencies = new Edge[]{ new Edge(v1, 7),

                               new Edge(v3, 2) };

       Vertex[] vertices = { v0, v1, v2, v3, v4 };

        computePaths(v0);

        for (Vertex v : vertices)

       {

           System.out.println("Distance to " + v + ": " + v.minDistance);

           List<Vertex> path = getShortestPathTo(v);

           System.out.println("Path: " + path);

       }

    }

}