Dijkstra’s Algorithm

**Actual Output for wdGraphs.txt**

G1's shortest path from 0 to 4:

( 0, 2, 10.869) --> 10.869

( 2, 1, 12.700) --> 23.569

( 1, 4, 25.297) --> 48.866

G2's shortest path from 0 to 4:

( 0, 3, 46.188) --> 46.188

( 3, 4, 26.595) --> 72.783

G3's shortest path from 0 to 54:

\*\*\* There is no path.

G4's shortest path from 0 to 54:

( 0, 15, 27.856) --> 27.856

( 15, 40, 25.831) --> 53.687

( 40, 37, 17.425) --> 71.112

( 37, 45, 15.195) --> 86.307

( 45, 30, 35.203) --> 121.510

( 30, 49, 53.613) --> 175.123

( 49, 34, 44.151) --> 219.274

( 34, 51, 56.281) --> 275.555

( 51, 54, 71.991) --> 347.546

G5's shortest path from 0 to 104:

( 0, 6, 76.886) --> 76.886

( 6, 80, 13.975) --> 90.861

( 80, 84, 30.383) --> 121.244

( 84, 71, 30.171) --> 151.415

( 71, 55, 18.473) --> 169.888

( 55, 104, 61.660) --> 231.548

G6's shortest path from 0 to 104:

( 0, 16, 14.024) --> 14.024

( 16, 17, 32.340) --> 46.364

( 17, 84, 28.333) --> 74.697

( 84, 88, 25.092) --> 99.789

( 88, 104, 20.194) --> 119.983

G7's shortest path from 0 to 154:

( 0, 100, 29.059) --> 29.059

( 100, 111, 52.552) --> 81.611

( 111, 86, 43.486) --> 125.097

( 86, 154, 9.245) --> 134.342

G8's shortest path from 0 to 154:

( 0, 50, 2.222) --> 2.222

( 50, 51, 7.003) --> 9.225

( 51, 8, 18.882) --> 28.107

( 8, 75, 3.151) --> 31.258

( 75, 151, 32.595) --> 63.853

( 151, 86, 16.914) --> 80.767

( 86, 154, 14.775) --> 95.542

G9's shortest path from 0 to 204:

( 0, 159, 44.070) --> 44.070

( 159, 140, 0.788) --> 44.858

( 140, 178, 2.092) --> 46.950

( 178, 69, 22.797) --> 69.747

( 69, 50, 11.888) --> 81.635

( 50, 204, 18.217) --> 99.852

G10's shortest path from 0 to 204:

( 0, 90, 43.249) --> 43.249

( 90, 93, 25.783) --> 69.032

( 93, 161, 4.815) --> 73.847

( 161, 39, 6.461) --> 80.308

( 39, 3, 18.243) --> 98.551

( 3, 204, 39.510) --> 138.061

G11's shortest path from 0 to 254:

( 0, 11, 28.485) --> 28.485

( 11, 196, 9.382) --> 37.867

( 196, 30, 2.989) --> 40.856

( 30, 239, 56.440) --> 97.296

( 239, 207, 5.353) --> 102.649

( 207, 31, 6.666) --> 109.315

( 31, 254, 22.426) --> 131.741

G12's shortest path from 0 to 254:

( 0, 193, 14.045) --> 14.045

( 193, 170, 4.034) --> 18.079

( 170, 114, 5.573) --> 23.652

( 114, 137, 20.414) --> 44.066

( 137, 254, 72.134) --> 116.200

G13's shortest path from 0 to 304:

( 0, 248, 17.620) --> 17.620

( 248, 275, 1.743) --> 19.363

( 275, 111, 41.087) --> 60.450

( 111, 304, 39.444) --> 99.894

G14's shortest path from 0 to 304:

( 0, 29, 45.883) --> 45.883

( 29, 303, 13.295) --> 59.178

( 303, 113, 4.574) --> 63.752

( 113, 87, 28.974) --> 92.726

( 87, 304, 17.106) --> 109.832

G15's shortest path from 0 to 354:

( 0, 130, 35.549) --> 35.549

( 130, 184, 0.561) --> 36.110

( 184, 233, 37.550) --> 73.660

( 233, 354, 10.639) --> 84.299

G16's shortest path from 0 to 354:

( 0, 274, 18.123) --> 18.123

( 274, 260, 39.260) --> 57.383

( 260, 180, 4.874) --> 62.257

( 180, 258, 14.829) --> 77.086

( 258, 89, 1.393) --> 78.479

( 89, 273, 23.676) --> 102.155

( 273, 354, 5.321) --> 107.476

G17's shortest path from 0 to 404:

( 0, 236, 44.074) --> 44.074

( 236, 404, 25.655) --> 69.729

G18's shortest path from 0 to 404:

( 0, 282, 11.115) --> 11.115

( 282, 341, 1.710) --> 12.825

( 341, 135, 2.719) --> 15.544

( 135, 139, 7.848) --> 23.392

( 139, 73, 14.759) --> 38.151

( 73, 148, 5.643) --> 43.794

( 148, 183, 0.178) --> 43.972

( 183, 350, 0.530) --> 44.502

( 350, 404, 38.604) --> 83.106

G19's shortest path from 0 to 454:

( 0, 84, 28.758) --> 28.758

( 84, 244, 9.415) --> 38.173

( 244, 223, 2.650) --> 40.823

( 223, 111, 5.708) --> 46.531

( 111, 454, 6.807) --> 53.338

G20's shortest path from 0 to 454:

( 0, 146, 0.019) --> 0.019

( 146, 117, 10.144) --> 10.163

( 117, 169, 2.618) --> 12.781

( 169, 454, 43.630) --> 56.411

**Summary**

There are 4 helper methods in the program. They are addEdge, solve, displayResult and processNieghbours, topologicalSort. Purpose of the addEdge method is to add an edge between given 2 vertices with the weight. Solve method is used to find the shortest path from vertex 0 to last vertex. ProcessNeighbour method is used as a helper method for solve method. Purpose of that method is to process all the neighbors of the given node and determine whether there is a shorter path from that vertex to all other vertices except the vertices already processed. displayResult method is used to display the result based on the given status.

To determine the shortest distance between given 2 nodes Dijkstra’s algorithm is used.

* Initialize a set of unvisited nodes and a priority queue of nodes to be processed, where the priority of each node is the estimated distance from the start node to the target node. The start node is assigned zero and all other nodes are assigned a distance of infinity.
* Take the node with the smallest estimated distance from the priority queue, mark it as visited, and remove it from the priority queue.
* For each neighbor of the current node, calculate the tentative distance from the start node to the neighbor through the current node. If this distance is less than the current distance assigned to the neighbor, update the distance and set the current node as the previous node for the neighbor.
* Repeat steps 2 and 3 until the shortest path between given 2 vertices are obtained.

The graph is represented by an adjacency list. ArrayList of ArrayList is used for that. Array of Nodes were used to store the calculated shortest distances between source and the other vertices. Integer Set was used to store the already processed vertices. Priority Queue is used to store the vertices that will be processed in future in an ordered way.

Time complexity of the algorithm is O((|V| + |E|) log V) , where V = Vertices, E = Edges. Space complexity of the algorithm is O(|V| + |E|) .