```java
import java.util.*;

public class Dijkstra {
    private int distances[];
    private Set<Integer> settled;
    private PriorityQueue<Node> priorityQueue;
    private int numNodes;
    private int matrix[][]; //adjacency matrix

    public Dijkstra(int number_of_nodes) {
        this.numNodes = number_of_nodes;
        distances = new int[numNodes + 1];
        settled = new HashSet<Integer>();
        priorityQueue = new PriorityQueue<Node>(numNodes,new
Node());
        matrix = new int[numNodes + 1][numNodes + 1];
    }

    public void dijkstra_algorithm(int adjacency_matrix[][], int
source) {
        int evaluationNode;
        for (int i = 1; i <= numNodes; i++)
            for (int j = 1; j <= numNodes; j++)
                matrix[i][j] = adjacency_matrix[i][j];

        for (int i = 1; i <= numNodes; i++) {
            distances[i] = Integer.MAX_VALUE;
        }

        priorityQueue.add(new Node(source, 0));
        distances[source] = 0;
        while (!priorityQueue.isEmpty()) {
            evaluationNode =
getNodeWithMinimumDistanceFromPriorityQueue();
            settled.add(evaluationNode);
            evaluateNeighbours(evaluationNode);
        }
    }

    private int getNodeWithMinimumDistanceFromPriorityQueue() {
        int node = priorityQueue.remove().node;
```

```
39          return node;
40      }
41
42      private void evaluateNeighbours(int eNode) { //evaluationNode
43          int edgeDistance = -1;
44          int newDistance = -1;
45
46          for (int dNode = 1; dNode <= numNodes; dNode++) {
47              if (!settled.contains(dNode)) {
48                  if (matrix[eNode][dNode] != Integer.MAX_VALUE) {
49                      edgeDistance = matrix[eNode][dNode];
50                      newDistance = distances[eNode] + edgeDistance;
51                      if (newDistance < distances[dNode]) {
52                          distances[dNode] = newDistance;
53                      }
54                      priorityQueue.add(new
  Node(dNode,distances[dNode]));
55                  }
56              }
57          }
58      }
59
60      public static void main(String... arg) {
61          int adjacency_matrix[][];
62          int numVertices;
63          int source = 0;
64          Scanner scan = new Scanner(System.in);
65          try {
66              System.out.println("Enter the number of vertices");
67              numVertices = scan.nextInt();
68              adjacency_matrix = new int[numVertices + 1][numVertices
  + 1];
69
70              System.out.println("Enter the Weighted Matrix for the
  graph");
71              for (int i = 1; i <= numVertices; i++) {
72                  for (int j = 1; j <= numVertices; j++) {
73                      adjacency_matrix[i][j] = scan.nextInt();
74                      if (i == j) {
75                          adjacency_matrix[i][j] = 0;
76                          continue;
```

```
77                      }
78                      if (adjacency_matrix[i][j] == 0) {
79                          adjacency_matrix[i][j] =
  Integer.MAX_VALUE;
80                      }
81                  }
82              }
83              System.out.println("Enter the source ");
84              source = scan.nextInt();
85              Dijkstra dPQueue = new Dijkstra(numVertices);
86              dPQueue.dijkstra_algorithm(adjacency_matrix, source);
87              System.out.println("Shortest paths:");
88              for (int i = 1; i <= dPQueue.distances.length - 1; i++)
  {
89                  System.out.println(source + " to " + i + " is " +
  dPQueue.distances[i]);
90              }
91          } catch (InputMismatchException inputMismatch) {
92              System.out.println("Wrong Input Format");
93          }
94      }
95  }
96
97  class Node implements Comparator<Node> {
98      public int node;
99      public int cost;
100     public Node() {}
101     public Node(int node, int cost) {
102         this.node = node;
103         this.cost = cost;
104     }
105
106     public int compare(Node node1, Node node2) {
107         if (node1.cost < node2.cost)
108             return -1;
109         if (node1.cost > node2.cost)
110             return 1;
111         return 0;
112     }
113 }
```