**ASSIGNMENT NO.3**

**Single-Source shortest path algorithm:-**

import java.util.\*;

class Edge

{

    int source, dest, weight;

    public Edge(int source, int dest, int weight)

    {

        this.source = source;

        this.dest = dest;

        this.weight = weight;

    }

}

class Node

{

    int vertex, weight;

    public Node(int vertex, int weight)

    {

        this.vertex = vertex;

        this.weight = weight;

    }

}

class Graph

{

    List<List<Edge>> adjList = null;

    Graph(List<Edge> edges, int n)

    {

        adjList = new ArrayList<>();

        for (int i = 0; i < n; i++) {

            adjList.add(new ArrayList<>());

        }

        for (Edge edge: edges) {

            adjList.get(edge.source).add(edge);

        }

    }

}

class Main

{

    private static void getRoute(int[] prev, int i, List<Integer> route)

    {

        if (i >= 0)

        {

            getRoute(prev, prev[i], route);

            route.add(i);

        }

    }

    public static void findShortestPaths(Graph graph, int source, int n)

    {

        PriorityQueue<Node> minHeap;

        minHeap = new PriorityQueue<>(Comparator.comparingInt(node -> node.weight));

        minHeap.add(new Node(source, 0));

        List<Integer> dist;

        dist = new ArrayList<>(Collections.nCopies(n, Integer.MAX\_VALUE));

         dist.set(source, 0);

         boolean[] done = new boolean[n];

        done[source] = true;

         int[] prev = new int[n];

        prev[source] = -1;

    while (!minHeap.isEmpty())

        {

 Node node = minHeap.poll();

            int u = node.vertex;

            for (Edge edge: graph.adjList.get(u))

            {

                int v = edge.dest;

                int weight = edge.weight;

                if (!done[v] && (dist.get(u) + weight) < dist.get(v))

                {

                    dist.set(v, dist.get(u) + weight);

                    prev[v] = u;

                    minHeap.add(new Node(v, dist.get(v)));

                }

            }

            done[u] = true;

        }

        List<Integer> route = new ArrayList<>();

        for (int i = 0; i < n; i++)

        {

            if (i != source && dist.get(i) != Integer.MAX\_VALUE)

            {

                getRoute(prev, i, route);

                System.out.printf("Path (%d —> %d): Minimum cost = %d, Route = %s\n",

                                source, i, dist.get(i), route);

                route.clear();

            }

        }

    }

    public static void main(String[] args)

    {

        List<Edge> edges = Arrays.asList(

                new Edge(0, 1, 10), new Edge(0, 4, 3), new Edge(1, 2, 2),

                new Edge(1, 4, 4), new Edge(2, 3, 9), new Edge(3, 2, 7),

                new Edge(4, 1, 1), new Edge(4, 2, 8), new Edge(4, 3, 2)

        );

        int n = 5;

        Graph graph = new Graph(edges, n);

        for (int source = 0; source < n; source++) {

            findShortestPaths(graph, source, n);

        }

    }

}