CN LAB 5

Name: Rachit Nimje
Class: TY-CS-D
Roll no: 12

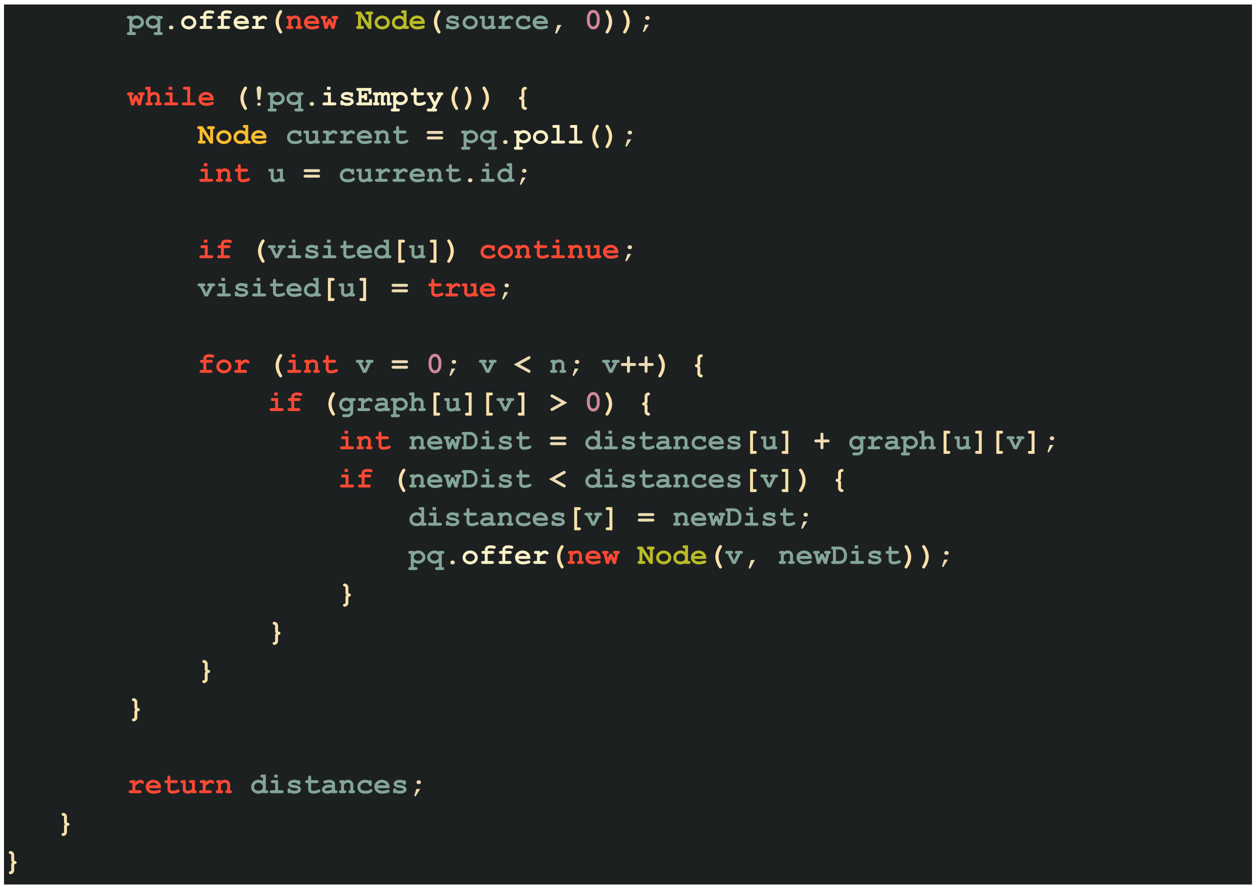
Batch: 1
PRN: 12210952

Title: Write a program to find the shortest path using Dijkstra Equation for Link State Routing Protocol which is used by Open Shortest Path First Protocol (OSPF) on the Internet for the network flow provided by the instructor.

Code:

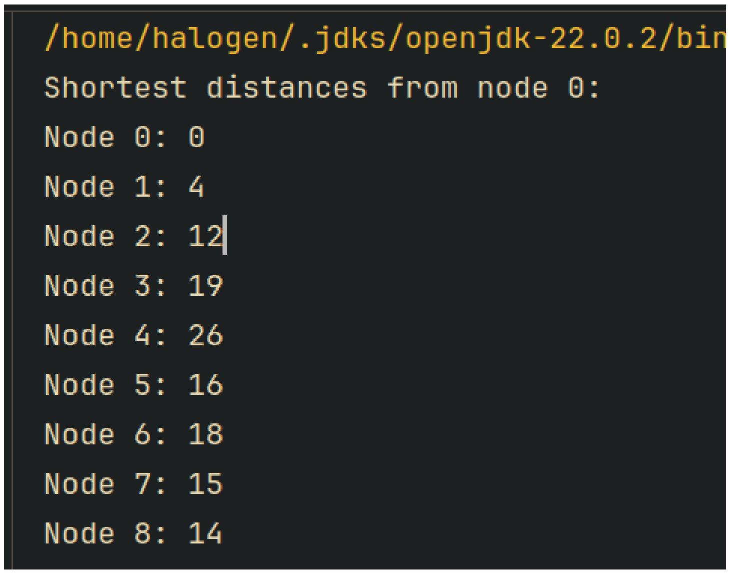
```
import javax.swing.*;
import java.awt.*;
import java.util.*;

public class DijkstraOSPF extends JFrame {
   private static final int INF = Integer.MAX_VALUE;
   private static final int WIDTH = 800;
   private static final int HEIGHT = 600;
   private static final int NODE_RADIUS = 20;

   private int[][] graph;
   private int[] distances;
   private int nodeCount;

   static class Node implements Comparable<Node> {
       int id;
       int distance;

       Node(int id, int distance) {
           this.id = id;
           this.distance = distance;
       }

       @Override
       public int compareTo(Node other) {
           return Integer.compare(this.distance, other.distance);
       }
   }

   public DijkstraOSPF(int[][] graph, int[] distances) {
       this.graph = graph;
       this.distances = distances;
       this.nodeCount = graph.length;
```

```java
        setTitle("Network Visualization");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        int centerX = WIDTH / 2;
        int centerY = HEIGHT / 2;
        int radius = Math.min(WIDTH, HEIGHT) / 3;

        for (int i = 0; i < nodeCount; i++) {
            for (int j = i + 1; j < nodeCount; j++) {
                if (graph[i][j] > 0) {
                    Point p1 = getNodePosition(i, nodeCount, centerX,
centerY, radius);
                    Point p2 = getNodePosition(j, nodeCount, centerX,
centerY, radius);
                    g2d.setColor(Color.GRAY);
                    g2d.drawLine(p1.x, p1.y, p2.x, p2.y);
                    // Draw edge weight
                    int midX = (p1.x + p2.x) / 2;
                    int midY = (p1.y + p2.y) / 2;
                    g2d.setColor(Color.BLACK);
                    g2d.drawString(String.valueOf(graph[i][j]), midX,
midY);
                }
            }
        }

        for (int i = 0; i < nodeCount; i++) {
            Point p = getNodePosition(i, nodeCount, centerX, centerY,
radius);
            g2d.setColor(Color.WHITE);
            g2d.fillOval(p.x - NODE_RADIUS, p.y - NODE_RADIUS, 2 *
NODE_RADIUS, 2 * NODE_RADIUS);
            g2d.setColor(Color.BLACK);
            g2d.drawOval(p.x - NODE_RADIUS, p.y - NODE_RADIUS, 2 *
NODE_RADIUS, 2 * NODE_RADIUS);
            g2d.drawString(i + " (" + distances[i] + ")", p.x - 15, p.y
+ 5);
        }
    }

    private Point getNodePosition(int nodeIndex, int totalNodes, int
centerX, int centerY, int radius) {
        double angle = 2 * Math.PI * nodeIndex / totalNodes - Math.PI /
2;
        int x = (int) (centerX + radius * Math.cos(angle));
        int y = (int) (centerY + radius * Math.sin(angle));
        return new Point(x, y);
    }

    public static void main(String[] args) {
        int[][] graph = {
                {0, 4, 0, 0, 0, 0, 0, 18, 0},
                {4, 0, 8, 0, 0, 0, 0, 11, 0},
                {0, 8, 0, 7, 0, 4, 0, 0, 2},
                {0, 0, 7, 0, 9, 14, 0, 0, 0},
                {0, 0, 0, 9, 0, 10, 0, 0, 0},
                {0, 0, 4, 14, 10, 0, 2, 0, 0},
                {0, 0, 0, 0, 0, 2, 0, 10, 6},
                {18, 11, 0, 0, 0, 0, 10, 0, 7},
                {0, 0, 2, 0, 0, 0, 6, 7, 0}
        };

        int source = 0;
        int[] distances = dijkstra(graph, source);

        System.out.println("Shortest distances from node " + source +
":");
        for (int i = 0; i < distances.length; i++) {
            System.out.println("Node " + i + ": " + distances[i]);
        }

        SwingUtilities.invokeLater(() -> {
            DijkstraOSPF visualization = new DijkstraOSPF(graph,
distances);
            visualization.setVisible(true);
        });
    }

    public static int[] dijkstra(int[][] graph, int source) {
        int n = graph.length;
        int[] distances = new int[n];
        boolean[] visited = new boolean[n];

        Arrays.fill(distances, INF);
        distances[source] = 0;

        PriorityQueue<Node> pq = new PriorityQueue<>();
```



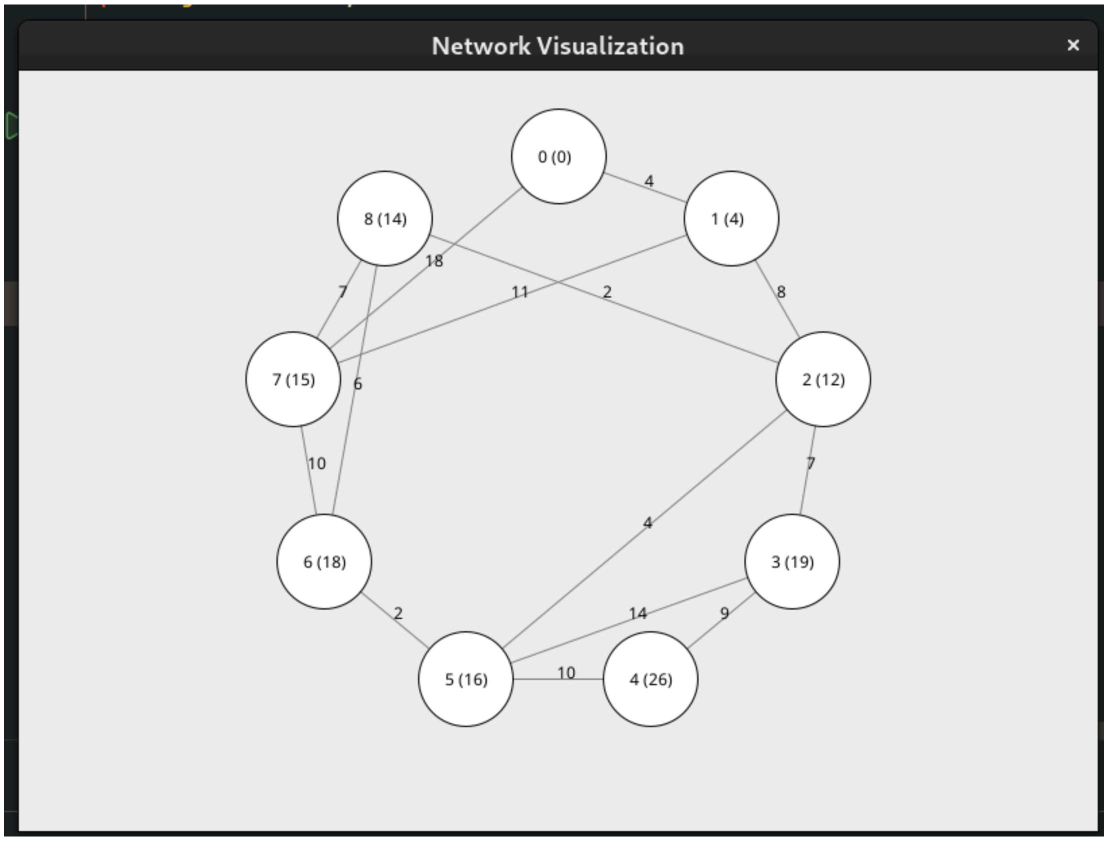
```
        pq.offer(new Node(source, 0));

        while (!pq.isEmpty()) {
            Node current = pq.poll();
            int u = current.id;

            if (visited[u]) continue;
            visited[u] = true;

            for (int v = 0; v < n; v++) {
                if (graph[u][v] > 0) {
                    int newDist = distances[u] + graph[u][v];
                    if (newDist < distances[v]) {
                        distances[v] = newDist;
                        pq.offer(new Node(v, newDist));
                    }
                }
            }
        }

        return distances;
    }
}
```

Output:



```
/home/halogen/.jdks/openjdk-22.0.2/bin
Shortest distances from node 0:
Node 0: 0
Node 1: 4
Node 2: 12
Node 3: 19
Node 4: 26
Node 5: 16
Node 6: 18
Node 7: 15
Node 8: 14
```

Visualization:
Node format: Node_id (shortest distance from Node 0)
Edge: distance/cost
Next page->


Network Visualization
0 (0)
1 (4)
2 (12)
3 (19)
4 (26)
5 (16)
6 (18)
7 (15)
8 (14)
4
8
7
9
10
14
4
2
10
6
7
18
11
2