Date: 19 / 09 /24

Lab Practical #12:

To develop network using distance vector routing protocol and link state routing protocol.

Practical Assignment #12:

1. C/Java Program: Distance Vector Routing Algorithm using Bellman Ford's Algorithm.

```
// Java Program to implement
// Bellman For Algorithm
import java.util.Arrays;

// Bellman For Algorothm
public class BellmanFord {
      // Graph is Created Using Edge Class
    static class Edge {
        int source, destination, weight;

        Edge() {
            source = destination = weight = 0;
        }
    }

    int V, E;
    Edge edge[];

    // Constructor to initialize the graph
    BellmanFord(int v, int e) {
        V = v;
        E = e;
        edge = new Edge[e];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }

    // Bellman-Ford Algorithm to find shortest paths from source to all vertices
    void BellmanFordAlgo(BellmanFord graph, int source) {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];

        // Step 1: Initialize distances from source to all other vertices as
INFINITE
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[source] = 0;

```

Date: 19 / 09 /24

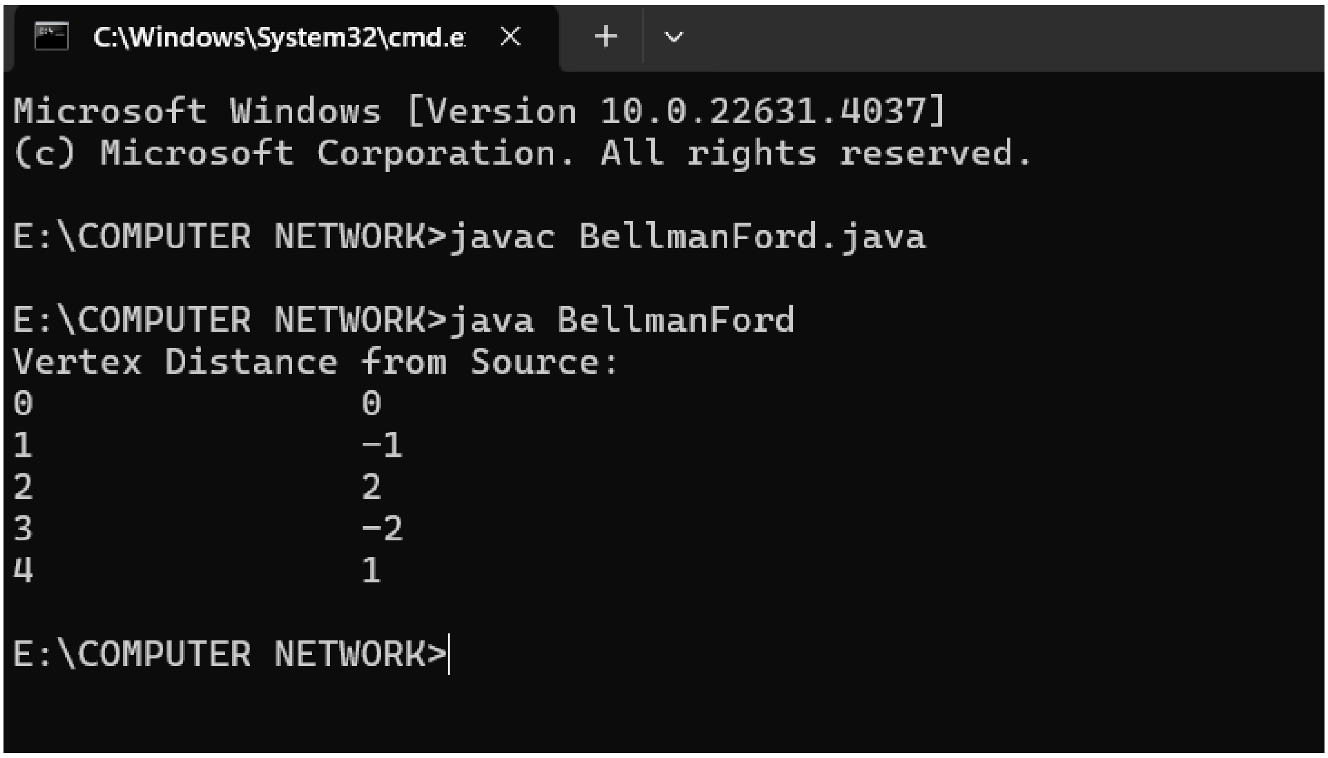
```
        // Step 2: Relax all edges |V| - 1 times.
        for (int i = 1; i < V; ++i) {
            for (int j = 0; j < E; ++j) {
                int u = graph.edge[j].source;
                int v = graph.edge[j].destination;
                int weight = graph.edge[j].weight;
                if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v])
                    dist[v] = dist[u] + weight;
            }
        }

        // Step 3: Check for negative-weight cycles
        for (int j = 0; j < E; ++j) {
            int u = graph.edge[j].source;
            int v = graph.edge[j].destination;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                System.out.println("Graph contains negative weight cycle");
                return;
            }
        }

        // Print distances from source to all vertices
        printDistances(dist, V);
    }

    // Print distances from source to all vertices
    void printDistances(int dist[], int V) {
        System.out.println("Vertex Distance from Source:");
        for (int i = 0; i < V; ++i)
            System.out.println(i + "\t\t" + dist[i]);
    }

    // Main method to test the Bellman-Ford algorithm
    public static void main(String[] args) {
        int V = 5;
        int E = 8;
        BellmanFord graph = new BellmanFord(V, E);

        // Define edges
        // Edge 0-1
        graph.edge[0].source = 0;
        graph.edge[0].destination = 1;
        graph.edge[0].weight = -1;

        // Edge 0-2
```

Date: 19 / 09 /24

```
        graph.edge[1].source = 0;
        graph.edge[1].destination = 2;
        graph.edge[1].weight = 4;

        // Edge 1-2
        graph.edge[2].source = 1;
        graph.edge[2].destination = 2;
        graph.edge[2].weight = 3;

        // Edge 1-3
        graph.edge[3].source = 1;
        graph.edge[3].destination = 3;
        graph.edge[3].weight = 2;

        // Edge 1-4
        graph.edge[4].source = 1;
        graph.edge[4].destination = 4;
        graph.edge[4].weight = 2;

        // Edge 3-2
        graph.edge[5].source = 3;
        graph.edge[5].destination = 2;
        graph.edge[5].weight = 5;

        // Edge 3-1
        graph.edge[6].source = 3;
        graph.edge[6].destination = 1;
        graph.edge[6].weight = 1;

        // Edge 4-3
        graph.edge[7].source = 4;
        graph.edge[7].destination = 3;
        graph.edge[7].weight = -3;

        // Execute Bellman-Ford algorithm
        graph.BellmanFordAlgo(graph, 0);
    }
}

```

Date: 19 / 09 /24

- **Output :-**



```
Microsoft Windows [Version 10.0.22631.4037]
(c) Microsoft Corporation. All rights reserved.

E:\COMPUTER NETWORK>javac BellmanFord.java

E:\COMPUTER NETWORK>java BellmanFord
Vertex Distance from Source:
0               0
1               -1
2               2
3               -2
4               1

E:\COMPUTER NETWORK>
```

2. C/Java Program: Link state routing algorithm.

```java
// Java Program to implement
import java.util.*;

public class LinkStateRouting {
    private Map<Integer, Node> nodes;
    private Map<Integer, Set<Integer>> links;

    public LinkStateRouting() {
        nodes = new HashMap<>();
        links = new HashMap<>();
    }

    public void addNode(int nodeId) {
        nodes.put(nodeId, new Node(nodeId));
    }

    public void addLink(int node1, int node2, int cost) {
        Node n1 = nodes.get(node1);
        Node n2 = nodes.get(node2);
```

Date: 19 / 09 /24

```
            n1.addNeighbor(n2, cost);
            n2.addNeighbor(n1, cost);
            links.putIfAbsent(node1, new HashSet<>());
            links.get(node1).add(node2);
            links.putIfAbsent(node2, new HashSet<>());
            links.get(node2).add(node1);
        }

        public void floodLinkState(int nodeId) {
            Node sourceNode = nodes.get(nodeId);
            Set<Integer> visited = new HashSet<>();
            Queue<Node> queue = new LinkedList<>();
            queue.add(sourceNode);
            visited.add(nodeId);

            System.out.println("Flooding Link-State from Node: " + nodeId);
            while (!queue.isEmpty()) {
                Node current = queue.poll();
                for (Node neighbor : current.getNeighbors().keySet()) {
                    if (!visited.contains(neighbor.getId())) {
                        System.out.println("Link from Node " + current.getId() + " to Node " + neighbor.getId());
                        queue.add(neighbor);
                        visited.add(neighbor.getId());
                    }
                }
            }
        }

        public void calculateShortestPaths(int sourceId) {
            Node sourceNode = nodes.get(sourceId);
            Map<Node, Integer> distances = new HashMap<>();
            PriorityQueue<NodeDistance> pq = new PriorityQueue<>(Comparator.comparingInt(nd -> nd.distance));

            for (Node node : nodes.values()) {
                if (node.equals(sourceNode)) {
                    distances.put(node, 0);
                } else {
                    distances.put(node, Integer.MAX_VALUE);
                }
            }

            pq.add(new NodeDistance(sourceNode, 0));

            while (!pq.isEmpty()) {
```

Date: 19 / 09 /24

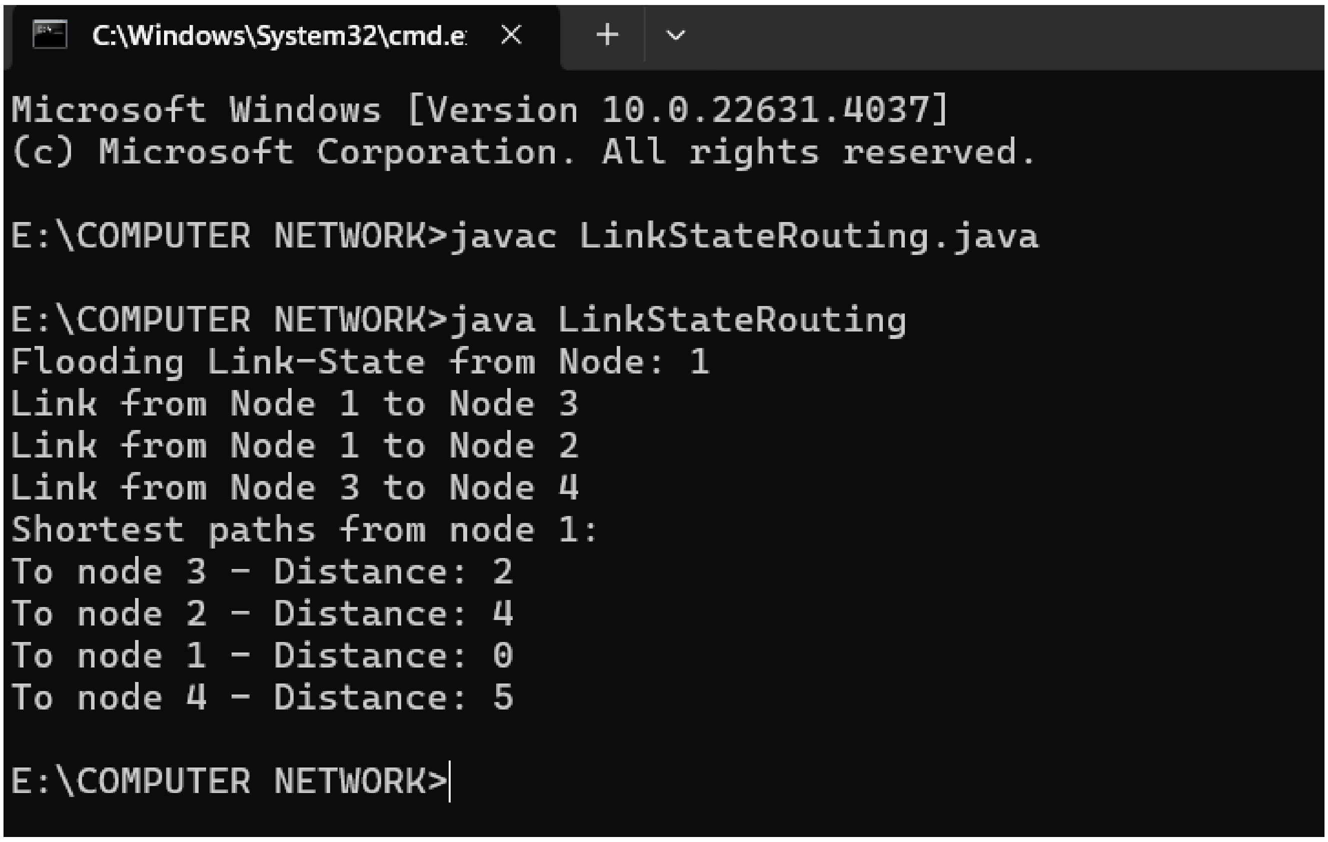
```java
                NodeDistance current = pq.poll();
                Node currentNode = current.node;

                for (Map.Entry<Node, Integer> neighborEntry : currentNode.getNeighbors().entrySet()) {
                    Node neighbor = neighborEntry.getKey();
                    int edgeWeight = neighborEntry.getValue();
                    int newDist = distances.get(currentNode) + edgeWeight;

                    if (newDist < distances.get(neighbor)) {
                        distances.put(neighbor, newDist);
                        pq.add(new NodeDistance(neighbor, newDist));
                    }
                }
            }

            // Display the shortest paths
            System.out.println("Shortest paths from node " + sourceId + ":");
            for (Map.Entry<Node, Integer> entry : distances.entrySet()) {
                System.out.println("To node " + entry.getKey().getId() + " - Distance: " + entry.getValue());
            }
        }

        public static void main(String[] args) {
            LinkStateRouting routing = new LinkStateRouting();

            // Adding nodes
            routing.addNode(1);
            routing.addNode(2);
            routing.addNode(3);
            routing.addNode(4);

            // Adding links between nodes with costs
            routing.addLink(1, 2, 4);
            routing.addLink(1, 3, 2);
            routing.addLink(2, 3, 5);
            routing.addLink(2, 4, 10);
            routing.addLink(3, 4, 3);

            // Flood link state starting from node 1
            routing.floodLinkState(1);

            // Calculate shortest paths from node 1
            routing.calculateShortestPaths(1);
        }
```

Date: 19 / 09 /24

```
}

// Helper class for shortest path calculation
class NodeDistance {
    Node node;
    int distance;

    public NodeDistance(Node node, int distance) {
        this.node = node;
        this.distance = distance;
    }
}

// Node class with neighbors and methods
class Node {
    private int id;
    private Map<Node, Integer> neighbors;

    public Node(int id) {
        this.id = id;
        neighbors = new HashMap<>();
    }

    public void addNeighbor(Node neighbor, int cost) {
        neighbors.put(neighbor, cost);
    }

    public int getId() {
        return id;
    }

    public Map<Node, Integer> getNeighbors() {
        return neighbors;
    }
}

```

Date: 19 / 09 /24

- **Output:-**



```
Microsoft Windows [Version 10.0.22631.4037]
(c) Microsoft Corporation. All rights reserved.

E:\COMPUTER NETWORK>javac LinkStateRouting.java

E:\COMPUTER NETWORK>java LinkStateRouting
Flooding Link-State from Node: 1
Link from Node 1 to Node 3
Link from Node 1 to Node 2
Link from Node 3 to Node 4
Shortest paths from node 1:
To node 3 - Distance: 2
To node 2 - Distance: 4
To node 1 - Distance: 0
To node 4 - Distance: 5

E:\COMPUTER NETWORK>
```