Date: 27/08/2025

Lab Practical #13:

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

Practical Assignment #13:

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

```java
// 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: 27/08/2025

```
        // 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;
```

Date: 27/08/2025

```
        // Edge 0-2
        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;

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

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

```
import java.util.Scanner;
public class DijkstraRouting {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count, srcRouter, i, j, k, w, v = 0, min;
        int[][] costMatrix = new int[100][100];
        int[] dist = new int[100];
        int[] last = new int[100];
        boolean[] flag = new boolean[100];

        System.out.print("\nEnter the number of routers: ");
        count = sc.nextInt();

        System.out.println("\nEnter the cost matrix values:");
        for (i = 0; i < count; i++) {
            for (j = 0; j < count; j++) {
                System.out.print(i + "->" + j + ": ");
                costMatrix[i][j] = sc.nextInt();
                if (costMatrix[i][j] < 0) {
                    costMatrix[i][j] = 1000; // Treat negative as infinity
                }
            }
        }
```

Date: 27/08/2025

```
        System.out.print("\nEnter the source router: ");
        srcRouter = sc.nextInt();

        for (v = 0; v < count; v++) {
            flag[v] = false;
            last[v] = srcRouter;
            dist[v] = costMatrix[srcRouter][v];
        }
        flag[srcRouter] = true;

        for (i = 0; i < count; i++) {
            min = 1000;
            for (w = 0; w < count; w++) {
                if (!flag[w] && dist[w] < min) {
                    v = w;
                    min = dist[w];
                }
            }
            flag[v] = true;

            for (w = 0; w < count; w++) {
                if (!flag[w] && (min + costMatrix[v][w] < dist[w])) {
                    dist[w] = min + costMatrix[v][w];
                    last[w] = v;
                }
            }
        }

        // Print result
        for (i = 0; i < count; i++) {
            System.out.print("\n" + srcRouter + " ==> " + i + ": Path taken: " + i);
            w = i;
            while (w != srcRouter) {
                System.out.print(" <-- " + last[w]);
                w = last[w];
            }
            System.out.println("\nShortest path cost: " + dist[i]);
        }

        sc.close();
    }
}
```