Date: 10 / 09/ 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.**

```
import java.util.*;

class Router {
   int id;
   int[] distance;
   int[] nextHop;
   List<Integer> neighbors;

   public Router(int id, int numRouters) {
      this.id = id;
      this.distance = new int[numRouters];
      this.nextHop = new int[numRouters];
      this.neighbors = new ArrayList<>();
      for (int i = 0; i < numRouters; i++) {
         if (i == id) {
            distance[i] = 0;
         } else {
            distance[i] = Integer.MAX_VALUE;
         }
         nextHop[i] = -1;
      }
   }
```

Date: 10 / 09/ 2025

```
    public void addNeighbor(int neighbor) {
        neighbors.add(neighbor);
    }
}

public class DistanceVectorRouting {
    private static final int INFINITY = Integer.MAX_VALUE;
    private int numRouters;
    private Router[] routers;
    private int[][] costMatrix;

    public DistanceVectorRouting(int numRouters) {
          this.numRouters = numRouters;
        routers = new Router[numRouters];
        costMatrix = new int[numRouters][numRouters];
        for (int i = 0; i < numRouters; i++) {
            routers[i] = new Router(i, numRouters);
            for (int j = 0; j < numRouters; j++) {
                costMatrix[i][j] = (i == j) ? 0 : INFINITY;
            }
        }
    }

    public void addLink(int from, int to, int cost) {
        costMatrix[from][to] = cost;
        costMatrix[to][from] = cost;
        routers[from].addNeighbor(to);
        routers[to].addNeighbor(from);
```

Date: 10 / 09/ 2025

```
    }

    public void bellmanFord() {
        boolean updated;
        for (int step = 0; step < numRouters - 1; step++) {
            updated = false;
            for (int i = 0; i < numRouters; i++) {
                Router router = routers[i];

                for (int neighbor : router.neighbors) {
                    for (int dest = 0; dest < numRouters; dest++) {
                        if (router.distance[dest] > routers[neighbor].distance[dest]
+ costMatrix[i][neighbor]) {
                            router.distance[dest] = routers[neighbor].distance[dest]
+ costMatrix[i][neighbor];
                            router.nextHop[dest] = neighbor;
                            updated = true;
                        }
                    }
                }
            if (!updated) {
                break;
            }
        }

    }
}
```

Date: 10 / 09/ 2025

```
    public void printRoutingTable() {
        System.out.println("Routing Tables:");
        for (Router router : routers) {
            System.out.println("\nRouter " + router.id + ":");
             System.out.println("Destination\tDistance\tNext Hop");
            for (int i = 0; i < numRouters; i++) {
                System.out.println(i + "\t\t" + (router.distance[i] == INFINITY ?
"Inf" : router.distance[i]) + "\t\t" + (router.nextHop[i] == -1 ? "-" :
router.nextHop[i]));
            }
        }
    }

    public static void main(String[] args) {
        int numRouters = 4;
        DistanceVectorRouting dvr = new DistanceVectorRouting(numRouters);
        dvr.addLink(0, 1, 1);
        dvr.addLink(0, 2, 4);
        dvr.addLink(1, 2, 2);
        dvr.addLink(1, 3, 6);
        dvr.addLink(2, 3, 3);
        dvr.bellmanFord();
        dvr.printRoutingTable();
    }
}
```

Date: 10 / 09/ 2025

```
D:\Programer\BTech\Sem-5\CN\LAB-13>java ./DistanceVectorRouting.java
Routing Tables:

Router 0:
Destination     Distance        Next Hop
0               -2147483648             1
1               -2147483646             1
2               -2147483648             1
3               -2147483648             1

Router 1:
Destination     Distance        Next Hop
0               -2147483647             0
1               -2147483647             2
2               -2147483647             0
3               -2147483647             0

Router 2:
Destination     Distance        Next Hop
0               -2147483646             3
1               -2147483646             3
2               -2147483646             3
3               -2147483645             1

Router 3:
Destination     Distance        Next Hop
0               -2147483643             2
1               -2147483643             2
2               -2147483643             2
3               -2147483642             2
```

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

```
import java.util.Arrays;

public class LinkStateRouting {

   static final int V = 6;
   static final int INF = Integer.MAX_VALUE;

   int minDistance(int dist[], boolean visited[]) {
      int min = INF, min_index = -1;

      for (int v = 0; v < V; v++) {
```

Date: 10 / 09/ 2025

```
        if (!visited[v] && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    }

    return min_index;
}

void printSolution(int dist[]) {
    System.out.println("Vertex \t Distance from Source");
    for (int i = 0; i < V; i++) {
        System.out.println(i + " \t\t " + dist[i]);
    }
}

void dijkstra(int graph[][], int src) {
    int dist[] = new int[V];
    boolean visited[] = new boolean[V];

    Arrays.fill(dist, INF);
    Arrays.fill(visited, false);
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, visited);
        visited[u] = true;

        for (int v = 0; v < V; v++) {
```

Date: 10 / 09/ 2025

```
            if (!visited[v] && graph[u][v] != 0 && dist[u] != INF && dist[u] +
                graph[u][v] < dist[v]) {
               dist[v] = dist[u] + graph[u][v];
            }
         }
      }
      printSolution(dist);
   }

   public static void main(String[] args) {
      int graph[][] = {
            { 0, 2, INF, 1, INF, INF },
            { 2, 0, 3, 2, INF, INF },
            { INF, 3, 0, INF, 7, 4 },
            { 1, 2, INF, 0, 5, INF },
            { INF, INF, 7, 5, 0, 6 },
            { INF, INF, 4, INF, 6, 0 }
      };
      LinkStateRouting lsr = new LinkStateRouting();
      lsr.dijkstra(graph, 0);
   }
}
```

```
D:\Programer\BTech\Sem-5\CN\LAB-13>java ./LinkStateRouting.java
Vertex   Distance from Source
0                0
1                -2147483645
2                -2147483648
3                1
4                -2147483642
5                -2147483648
```