Date: 03/09/2025

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.Scanner;

public class DistanceVectorRouting {
    private static final int INF = 9999;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of routers: ");
        int numRouters = scanner.nextInt();

        int[][] costMatrix = new int[numRouters][numRouters];
        System.out.println("Enter the cost matrix (use " + INF + " for infinity):");
        for (int i = 0; i < numRouters; i++) {
            for (int j = 0; j < numRouters; j++) {
                costMatrix[i][j] = scanner.nextInt();
            }
        }

        int[][] distanceVector = new int[numRouters][numRouters];
        int[][] nextHop = new int[numRouters][numRouters];

        for (int i = 0; i < numRouters; i++) {
            for (int j = 0; j < numRouters; j++) {
                distanceVector[i][j] = costMatrix[i][j];
                nextHop[i][j] = (costMatrix[i][j] != INF && i != j) ? j : -1;
            }
```

Date: 03/09/2025

```
        }

        boolean updated;
        do {
            updated = false;
            for (int i = 0; i < numRouters; i++) {
                for (int j = 0; j < numRouters; j++) {
                    for (int k = 0; k < numRouters; k++) {
                        if (distanceVector[i][k] + distanceVector[k][j] < distanceVector[i][j]) {
                            distanceVector[i][j] = distanceVector[i][k] + distanceVector[k][j];
                            nextHop[i][j] = nextHop[i][k];
                            updated = true;
                        }
                    }
                }
            }
        } while (updated);

        System.out.println("\nFinal Distance Vector Table:");
        for (int i = 0; i < numRouters; i++) {
            System.out.println("Router " + (i + 1) + ":");
            for (int j = 0; j < numRouters; j++) {
                if (distanceVector[i][j] == INF) {
                    System.out.print("INF ");
                } else {
                    System.out.print((distanceVector[i][j] + 1) + " ");
                }
            }
            System.out.println();
        }

        scanner.close();
    }
```

Date: 03/09/2025

```
}
```

Input:

```
Enter the number of routers: 3
Enter the cost matrix (use 9999 for infinity):
0 2 9999
2 0 4
9999 4 0
```

Output:

```
Final Distance Vector Table:
Router 1:
1 3 7
Router 2:
3 1 5
Router 3:
7 5 1
```

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

```
import java.util.*;
public class Dijkstra {
   static final int INF = Integer.MAX_VALUE;
   static int findKey(boolean[] visited, int[] distance, int V) {
      int min = INF;
      int key = -1;

      for (int i = 0; i < V; i++) {
         if (!visited[i] && distance[i] < min) {
            min = distance[i];
            key = i;
         }}
      return key;
```

Date: 03/09/2025

```
  }
static void dijkstra(int[][] graph, int src) {
    int V = graph.length;
    boolean[] visited = new boolean[V];
    int[] distance = new int[V];
    Arrays.fill(distance, INF);

    distance[src] = 0;

    for (int i = 0; i < V - 1; i++) {
      int u = findKey(visited, distance, V);
      if (u == -1) break;

      visited[u] = true;

      for (int v = 0; v < V; v++) {
        if (graph[u][v] != 0 && !visited[v] && distance[u] != INF
            && distance[v] > distance[u] + graph[u][v]) {
          distance[v] = distance[u] + graph[u][v];
        }}}

    System.out.println("\nShortest distances from node " + src + ":");
    for (int i = 0; i < V; i++) {
      if (distance[i] == INF)
        System.out.println("Node " + i + ": INF");
      else
        System.out.println("Node " + i + ": " + distance[i]);
    } }
    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter number of vertices: ");
    int V = sc.nextInt();
    int[][] graph = new int[V][V];
    System.out.println("Enter adjacency matrix (0 if no edge):");
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
        graph[i][j] = sc.nextInt();
      }}
    System.out.print("Enter source node (0 to " + (V - 1) + "): ");
    int src = sc.nextInt();

    System.out.println("\nGraph:");
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++)
        System.out.print(graph[i][j] + "\t");
      System.out.println();
    }
    dijkstra(graph, src);
  }
}
```

Input:

```
Enter number of vertices: 4
Enter adjacency matrix (0 if no edge):
0 5 9999 10
5 0 3 9999
9999 3 0 1
10 9999 1 0
Enter source node (0 to 3): 0
```

Output:

```
Shortest distances from node 0:

Node 0: 0 , Node 1: 5 , Node 2: 8 , Node 3: 9
```