Date: 13/09/2025

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.**

```
#include <stdio.h>
#define INF 999
int dist[50][50], temp[50][50], n;
void dvr() {
   int i, j, k;
   // Floyd-Warshall algorithm
   for (k = 0; k < n; k++) {
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            if (dist[i][k] + dist[k][j] < dist[i][j]) {
               dist[i][j] = dist[i][k] + dist[k][j];
               temp[i][j] = temp[i][k];
            }
         }
      }
   }
   // Print routing table
   for (i = 0; i < n; i++) {
      printf("\n\nState value for router %d is:\n", i + 1);
      for (j = 0; j < n; j++) {
         printf("Node %d via %d Distance %d\n", j + 1, temp[i][j] + 1, dist[i][j]);
      }
   }
   printf("\n");
```

Date: 13/09/2025

```
}
int main() {
    int i, j, x;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    printf("Enter the distance matrix (use 999 for no link):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &dist[i][j]);
            temp[i][j] = j;
        }
    }

    // Set diagonal elements to 0
    for (i = 0; i < n; i++)
        dist[i][i] = 0;

    // First computation
    dvr();

    // Update cost
    printf("Enter i and j for cost update: ");
    scanf("%d %d", &i, &j);
    printf("Enter new cost: ");
    scanf("%d", &x);

    dist[i][j] = x;
```

Date: 13/09/2025

```
     printf("After update:\n");
     dvr();

     return 0;
 }
```

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

```
#include <stdio.h>

#define INF 999


int n;

int cost[50][50];


// Function to find shortest path from source to all other vertices

void dijkstra(int source) {

    int dist[50], visited[50], nextHop[50];

    int i, j, count, minDist, u;


    // Initialization

    for (i = 0; i < n; i++) {

        dist[i] = cost[source][i];

        visited[i] = 0;

        if (cost[source][i] != INF && source != i)

            nextHop[i] = i;  // direct path
    else
        nextHop[i] = -1; // no direct path
}
dist[source] = 0;
visited[source] = 1;

// Dijkstra's Algorithm
for (count = 1; count < n - 1; count++) {
    minDist = INF;
    u = -1;
    for (i = 0; i < n; i++) {
        if (!visited[i] && dist[i] < minDist) {
            minDist = dist[i];
            u = i;
        }
    }
    if (u == -1) break; // No reachable vertex left
    visited[u] = 1;

    for (i = 0; i < n; i++) {
        if (!visited[i] && dist[u] + cost[u][i] < dist[i]) {
            dist[i] = dist[u] + cost[u][i];
```

```
                nextHop[i] = nextHop[u];
            }
        }
    }
    // Print routing table for this router
    printf("\nRouting Table for Router %d:\n", source + 1);
    printf("Dest\tNextHop\tCost\n");
    for (i = 0; i < n; i++) {
        if (i != source) {
            printf("%d\t", i + 1);
            if (nextHop[i] != -1)
                printf("%d\t", nextHop[i] + 1);
            else
                printf("-\t");
            if (dist[i] != INF)
                printf("%d\n", dist[i]);
            else
                printf("INF\n");
        }
    }
}
int main() {
```

Date: 13/09/2025

```
    int i, j;


    printf("Enter number of routers: ");
    scanf("%d", &n);


    printf("Enter the cost adjacency matrix (use 999 for no link):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }
    // Run Dijkstra for each router
    for (i = 0; i < n; i++) {
        dijkstra(i);
    }
    return 0;
}
```