Date: / /

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.io.*;
public class DistanceVectorRouting
{
 static int graph[][];
 static int via[][];
 static int rt[][];
 static int v;
 static int e;

 public static void main(String args[]) throws IOException
 {
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

  System.out.println("Please enter the number of Vertices: ");
  v = Integer.parseInt(br.readLine());

  System.out.println("Please enter the number of Edges: ");
  e = Integer.parseInt(br.readLine());

  graph = new int[v][v];
  via = new int[v][v];
  rt = new int[v][v];
  for(int i = 0; i < v; i++)
   for(int j = 0; j < v; j++)
   {
    if(i == j)
     graph[i][j] = 0;
    else
     graph[i][j] = 9999;
   }
  for(int i = 0; i < e; i++)
  {
   System.out.println("Please enter data for Edge " + (i + 1) + ":");
   System.out.print("Source: ");
   int s = Integer.parseInt(br.readLine());
   s--;
   System.out.print("Destination: ");
   int d = Integer.parseInt(br.readLine());
   d--;
   System.out.print("Cost: ");
   int c = Integer.parseInt(br.readLine());
   graph[s][d] = c;
   graph[d][s] = c;
  }

  dvr_calc_disp("The initial Routing Tables are: ");
  System.out.print("Please enter the Source Node for the edge whose cost has changed:
");
  int s = Integer.parseInt(br.readLine());
  s--;
  System.out.print("Please enter the Destination Node for the edge whose cost has
changed: ");
  int d = Integer.parseInt(br.readLine());
  d--;
  System.out.print("Please enter the new cost: ");
  int c = Integer.parseInt(br.readLine());
  graph[s][d] = c;
  graph[d][s] = c;

  dvr_calc_disp("The new Routing Tables are: ");
 }

 static void dvr_calc_disp(String message)
 {
  System.out.println();
  init_tables();
  update_tables();
  System.out.println(message);
  print_tables();
  System.out.println();
 }
 static void update_table(int source)
 {
  for(int i = 0; i < v; i++)
  {
   if(graph[source][i] != 9999)
   {
    int dist = graph[source][i];
    for(int j = 0; j < v; j++)
    {
     int inter_dist = rt[i][j];
     if(via[i][j] == source)
      inter_dist = 9999;
    if(dist + inter_dist < rt[source][j])
    {
     rt[source][j] = dist + inter_dist;
     via[source][j] = i;
    }
   }
  }
 }
}
static void update_tables()
{
 int k = 0;
 for(int i = 0; i < 4*v; i++)
 {
  update_table(k);
  k++;
  if(k == v)
   k = 0;
 }
}
static void init_tables()
{
 for(int i = 0; i < v; i++)
 {
  for(int j = 0; j < v; j++)
  {
   if(i == j)
   {
    rt[i][j] = 0;
    via[i][j] = i;
   }
   else
   {
    rt[i][j] = 9999;
    via[i][j] = 100;
   }
  }
 }
}
static void print_tables()
{
 for(int i = 0; i < v; i++)
 {
  for(int j = 0; j < v; j++)
  {
    System.out.print("Dist: " + rt[i][j] + "    ");
   }
   System.out.println();
  }
 }
}
```

Output :

```
Please enter the number of Vertices:
4
Please enter the number of Edges:
5
Please enter data for Edge 1:
Source: 1
Destination: 2
Cost: 1
Please enter data for Edge 2:
Source: 1
Destination: 3
Cost: 3
Please enter data for Edge 3:
Source: 2
Destination: 3
Cost: 1
Please enter data for Edge 4:
Source: 2
Destination: 4
Cost: 1
Please enter data for Edge 5:
Source: 3
Destination: 4
Cost: 4

The initial Routing Tables are:
Dist: 0    Dist: 1    Dist: 2    Dist: 2
Dist: 1    Dist: 0    Dist: 1    Dist: 1
Dist: 2    Dist: 1    Dist: 0    Dist: 2
Dist: 2    Dist: 1    Dist: 2    Dist: 0

Please enter the Source Node for the edge whose cost has changed: 2
Please enter the Destination Node for the edge whose cost has changed: 4
Please enter the new cost: 10
```

```
The new Routing Tables are:
Dist: 0    Dist: 1    Dist: 2    Dist: 6
Dist: 1    Dist: 0    Dist: 1    Dist: 5
Dist: 2    Dist: 1    Dist: 0    Dist: 4
Dist: 6    Dist: 5    Dist: 4    Dist: 0
```

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

```
import java.util.*;
import java.lang.*;

class LinkStateRouting {
    static final int V = 9;
    int minDistance(int dist[], Boolean sptSet[]) {
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++) {
            if (!sptSet[v] && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }
        }
        return min_index;
    }

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

    void linkStateRoute(int graph[][], int src) {
        int dist[] = new int[V];
        Boolean sptSet[] = new Boolean[V];
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        dist[src] = 0;
        for (int count = 0; count < V - 1; count++) {
            int u = minDistance(dist, sptSet);
            sptSet[u] = true;

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

Date: / /

```
                if (!sptSet[v] && graph[u][v] != 0 &&
                    dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] < dist[v]) {
                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }
        printSolution(dist);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkStateRouting sp = new LinkStateRouting();
        int graph[][] = new int[V][V];

        System.out.println("Enter the adjacency matrix for the graph (size 9x9):");
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                graph[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter the source vertex (0 to 8): ");
        int src = scanner.nextInt();
        sp.linkStateRoute(graph, src);
    }
}
```

Output:

```
Enter the number of routers: 6
Enter the adjacency matrix (use 0 for no direct connection):
0 7 9 0 0 14
7 0 10 15 0 0
9 10 0 11 0 2
0 15 11 0 6 0
0 0 0 6 0 9
14 0 2 0 9 0
Router 0:
To Router 1: Distance = 7, Path = 1 <- 0
To Router 2: Distance = -2147483640, Path = 2 <- 5 <- 1 <- 0
To Router 3: Distance = -2147483636, Path = 3 <- 4 <- 1 <- 0
To Router 4: Distance = -2147483642, Path = 4 <- 1 <- 0
To Router 5: Distance = -2147483642, Path = 5 <- 1 <- 0

Router 1:
To Router 0: Distance = 7, Path = 0 <- 1
To Router 2: Distance = -2147483631, Path = 2 <- 3 <- 0 <- 1
To Router 3: Distance = -2147483642, Path = 3 <- 0 <- 1
To Router 4: Distance = -2147483642, Path = 4 <- 0 <- 1
To Router 5: Distance = -2147483633, Path = 5 <- 4 <- 0 <- 1

Router 2:
To Router 0: Distance = -2147483640, Path = 0 <- 1 <- 5 <- 2
To Router 1: Distance = -2147483647, Path = 1 <- 5 <- 2
To Router 3: Distance = -2147483647, Path = 3 <- 5 <- 2
To Router 4: Distance = -2147483641, Path = 4 <- 3 <- 5 <- 2
To Router 5: Distance = 2, Path = 5 <- 2

Router 3:
To Router 0: Distance = -2147483643, Path = 0 <- 4 <- 3
To Router 1: Distance = -2147483643, Path = 1 <- 4 <- 3
To Router 2: Distance = -2147483643, Path = 2 <- 4 <- 3
To Router 4: Distance = 6, Path = 4 <- 3
To Router 5: Distance = -2147483641, Path = 5 <- 2 <- 4 <- 3

Router 4:
To Router 0: Distance = -2147483643, Path = 0 <- 3 <- 4
To Router 1: Distance = -2147483636, Path = 1 <- 0 <- 3 <- 4
To Router 2: Distance = -2147483641, Path = 2 <- 5 <- 3 <- 4
To Router 3: Distance = 6, Path = 3 <- 4
To Router 5: Distance = -2147483643, Path = 5 <- 3 <- 4

Router 5:
To Router 0: Distance = -2147483619, Path = 0 <- 1 <- 3 <- 4 <- 2 <- 5
To Router 1: Distance = -2147483626, Path = 1 <- 3 <- 4 <- 2 <- 5
To Router 2: Distance = 2, Path = 2 <- 5
To Router 3: Distance = -2147483641, Path = 3 <- 4 <- 2 <- 5
To Router 4: Distance = -2147483647, Path = 4 <- 2 <- 5
```