Date: 19/09/2024

Lab Practical #12:

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

Practical Assignment #12:

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

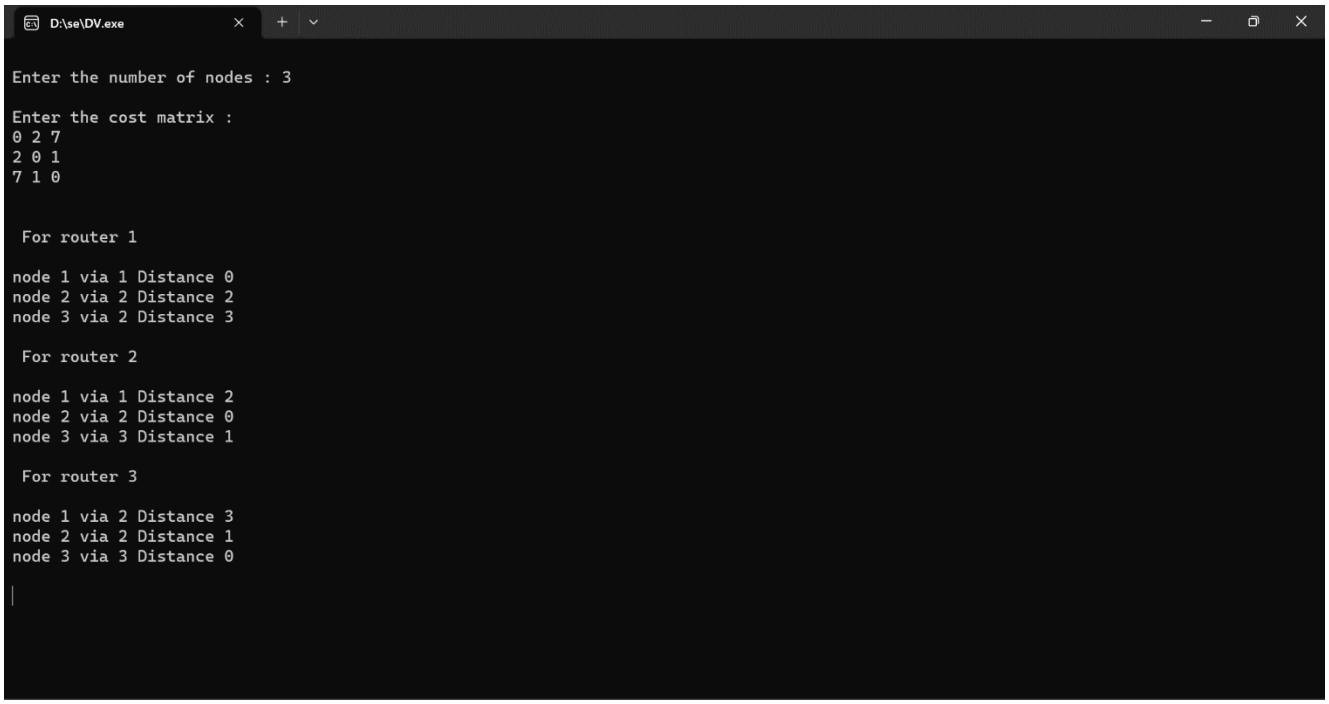
```
struct node
{
    unsigned dist[20];
    unsigned from[20];
} rt[10];
int main()
{
    int costmat[20][20];
    int nodes, i, j, k, count = 0;
    printf("\nEnter the number of nodes : ");
    scanf("%d", &nodes); // Enter the nodes
    printf("\nEnter the cost matrix :\n");
    for (i = 0; i < nodes; i++)
    {
        for (j = 0; j < nodes; j++)
        {
            scanf("%d", &costmat[i][j]);
            costmat[i][i] = 0;
            rt[i].dist[j] = costmat[i][j]; // initialise the distance equal to
cost matrix
            rt[i].from[j] = j;
        }
    }
    do
    {
        count = 0;
        for (i = 0; i < nodes; i++)
           for (j = 0; j < nodes; j++)
             for (k = 0; k < nodes; k++)
                    if (rt[i].dist[j] > costmat[i][k] + rt[k].dist[j])
                    { // We calculate the minimum distance
                        rt[i].dist[j] = rt[i].dist[k] + rt[k].dist[j];
                        rt[i].from[j] = k;
                        count++;
                    }
    } while (count != 0);
    for (i = 0; i < nodes; i++)
    {
        printf("\n\n For router %d\n", i + 1);
        for (j = 0; j < nodes; j++)
```

Date: 19/09/2024

```
            {
                printf("\t\nnode %d via %d Distance %d ", j + 1, rt[i].from[j] + 1,
rt[i].dist[j]);
            }
        }
        printf("\n\n");
        getch();
}
```

Date: 19/09/2024

Output:



Date: 19/09/2024

2. C Program: Link state routing algorithm.

```
#include "global.h"
#include <assert.h>
#include <limits.h>
#include <stdlib.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <string.h>

#define INFINITY INT_MAX
#define UNDEFINED (-1)
#define INDEX(x, y, nnodes) ((x) + (nnodes) * (y))

struct node_list
{
    char **nodes;
    int nnodes;
    int unsorted;
};

int nl_index(struct node_list *nl, char *node);

struct node_list *nl_create(void)
{
    return (struct node_list *)calloc(1, sizeof(struct node_list));
}

int nl_nsites(struct node_list *nl)
{
    return nl->nnodes;
}

void nl_add(struct node_list *nl, char *node)
{
    /* No duplicate nodes.
     */
    if (nl_index(nl, node) != -1)
    {
        return;
    }

    /* Create a copy of the site.
     */
    int len = strlen(node);
    char *copy = malloc(len + 1);
```

```
    strcpy(copy, node);

    /* Add this copy to the list.
     */
    nl->nodes = (char **)realloc(nl->nodes, sizeof(char *) * (nl->nnodes + 1));
    nl->nodes[nl->nnodes++] = copy;
    nl->unsorted = 1;
}

int nl_compare(const void *e1, const void *e2)
{
    const char **p1 = (const char **)e1, **p2 = (const char **)e2;
    return strcmp(*p1, *p2);
}

void nl_sort(struct node_list *nl)
{
    qsort(nl->nodes, nl->nnodes, sizeof(char *), nl_compare);
    nl->unsorted = 0;
}

/* Return the rank of the given site in the given site list.
 */
int nl_index(struct node_list *nl, char *node)
{
    /* Sort the list if not yet sorted.
     */
    if (nl->unsorted)
    {
        nl_sort(nl);
    }

    /* Binary search.
     */
    int lb = 0, ub = nl->nnodes;
    while (lb < ub)
    {
        int i = (lb + ub) / 2;
        int cmp = strcmp(node, nl->nodes[i]);
        if (cmp < 0)
        {
            ub = i;
        }
        else if (cmp > 0)
        {
            lb = i + 1;
```

```
        }
        else
        {
            return i;
        }
    }
    return -1;
}

char *nl_name(struct node_list *nl, int index)
{
    if (index < 0)
    {
        return "UNDEFINED";
    }
    return nl->nodes[index];
}

void nl_destroy(struct node_list *nl)
{
    int i;

    for (i = 0; i < nl->nnodes; i++)
    {
        free(nl->nodes[i]);
    }
    free(nl->nodes);
    free(nl);
}

/* Set the distance from src to dst.
 */
void set_dist(struct node_list *nl, int graph[], int nnodes, char *src, char *dst,
int dist)
{
    int x = nl_index(nl, src), y = nl_index(nl, dst);
    if (x < 0 || y < 0)
    {
        fprintf(stderr, "set_dist: bad source or destination\n");
        return;
    }
    graph[INDEX(x, y, nnodes)] = dist;
    // graph[INDEX(y, x, nnodes)] = dist;
}

char *addr_to_string(struct sockaddr_in addr)
```

Date: 19/09/2024

```
{
    char *addr_string = malloc(40);
    strcpy(addr_string, inet_ntoa(addr.sin_addr));
    strcat(addr_string, ":");
    char *port = malloc(12);
    sprintf(port, "%d", ntohs(addr.sin_port));
    strcat(addr_string, port);
    free(port);
    return addr_string;
}

struct sockaddr_in string_to_addr(char *string)
{
    char *port = index(string, ':');
    *port++ = '\0';

    struct sockaddr_in addr;
    memset((void *)&addr, 0, sizeof(addr));
    addr_get(&addr, string, atoi(port));
    *--port = ':';
    return addr;
}

/************************************************
    Dijkstra's algorithm
************************************************/
void dijkstra(int graph[], int nnodes, int src, int dist[], int prev[])
{
    int *visited = malloc(sizeof(int) * nnodes); // mark whether the node is visited
    int count, mindistance, nextnode, i, j;
    for (i = 0; i < nnodes; i++)
    {
        visited[i] = 0;
        if (graph[INDEX(src, i, nnodes)] == 1 || graph[INDEX(src, i, nnodes)] == 0)
        {
            dist[i] = graph[INDEX(src, i, nnodes)];
            prev[i] = src;
        }
        else
        {
            dist[i] = INFINITY;
        }
    }
    dist[src] = 0;
    visited[src] = 1;
    prev[src] = UNDEFINED; // src has no prev
```

Date: 19/09/2024

```
    for (count = 0; count < nnodes; count++)
    {
        mindistance = INFINITY;
        for (i = 0; i < nnodes; i++)
        {
            if (dist[i] < mindistance && !visited[i])
            {
                mindistance = dist[i];
                nextnode = i;
            }
        }
        visited[nextnode] = 1;
        for (j = 0; j < nnodes; j++)
        {
            if (!visited[j] && (graph[INDEX(nextnode, j, nnodes)] != INFINITY) &&
dist[nextnode] + graph[INDEX(nextnode, j, nnodes)] < dist[j])
            {

                dist[j] = dist[nextnode] + graph[INDEX(nextnode, j, nnodes)];
                prev[j] = nextnode;
            }
        }
    }
}
```

Date: 19/09/2024

Output: