COMPUTER NETWORK LAB

NAME: SPRIHA ANVI

REG. NO: 21BPS1191

IMPLEMENTATION OF LINK STATE ROUTING ALGORITHM

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

#define MAX_NODES 100

// Struct to represent a node in the network
typedef struct {
    int cost;
    bool visited;
    int prev_node;
} Node;

// Function to find the node with the minimum cost
int findMinCostNode(Node nodes[], int num_nodes) {
    int min_cost = INT_MAX;
```

```
    int min_node = -1;

    for (int i = 0; i < num_nodes; i++) {
        if (!nodes[i].visited && nodes[i].cost < min_cost) {
            min_cost = nodes[i].cost;
            min_node = i;
        }
    }

    return min_node;
}

// Function to print the shortest path from source to destination
void printShortestPath(Node nodes[], int dest) {
    if (nodes[dest].prev_node != -1) {
        printShortestPath(nodes, nodes[dest].prev_node);
    }

    printf("%d ", dest);
}

// Dijkstra's algorithm for link state routing
void dijkstra(int graph[MAX_NODES][MAX_NODES], int
num_nodes, int source) {
    Node nodes[MAX_NODES];

    for (int i = 0; i < num_nodes; i++) {
        nodes[i].cost = INT_MAX;
        nodes[i].visited = false;
        nodes[i].prev_node = -1;
    }

    nodes[source].cost = 0;

    for (int count = 0; count < num_nodes - 1; count++) {
        int current_node = findMinCostNode(nodes, num_nodes);
```

```
        if (current_node == -1) {
            break;
        }

        nodes[current_node].visited = true;

        for (int i = 0; i < num_nodes; i++) {
            if (!nodes[i].visited && graph[current_node][i] &&
nodes[current_node].cost != INT_MAX &&
                nodes[current_node].cost + graph[current_node][i] <
nodes[i].cost) {
                nodes[i].cost = nodes[current_node].cost +
graph[current_node][i];
                nodes[i].prev_node = current_node;
            }
        }
    }

    // Print shortest paths to all nodes
    printf("Shortest paths from node %d:\n", source);
    for (int i = 0; i < num_nodes; i++) {
        if (i != source) {
            printf("Node %d: Cost = %d, Path: ", i, nodes[i].cost);
            printShortestPath(nodes, i);
            printf("\n");
        }
    }
}

int main() {
    int num_nodes, source;
    int graph[MAX_NODES][MAX_NODES];

    printf("Enter the number of nodes: ");
    scanf("%d", &num_nodes);

    printf("Enter the adjacency matrix:\n");
```

```
    for (int i = 0; i < num_nodes; i++) {
        for (int j = 0; j < num_nodes; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    printf("Enter the source node: ");
    scanf("%d", &source);

    dijkstra(graph, num_nodes, source);

    return 0;
}
```

OUTPUT: