



NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

School of Electrical Engineering and Computer Sciences

EE-357 Computer and Communication Networks

END SEMESTER PROJECT REPORT

**Analysis of routing of a 50-node network via Link-State Algorithm using
Dijkstra's Algorithm**

INSTRUCTOR: Dr. Huma Ghafoor

GROUP MEMBERS:

Muhammad Sohaib Ikram (283756)

Abdul Ahad Sheikh (282457)

Danyal Wali (300230)

Hammad Hassan (309328)

SECTION: BEE 11-A

DATE OF SUBMISSION: 24/05/2022

Contents

Problem Statement:.....	3
Introduction:	3
Dijkstra's Algorithm:.....	3
Objectives:	3
Modules:.....	4
1. Network topology:	4
Topology:	4
Adjacency/Cost Matrix:[2].....	5
2. Calculating Shortest Path:	7
Dijkstra's Algorithm:.....	7
Results for shortest path by tool:.....	7
Results for shortest path by our code:.....	8
3. Calculating Number of Hops:	9
Results for shortest path by tool:.....	9
Results for shortest path by our code:.....	10
4. Calculating total number of paths.....	11
Directional/Undirectional graphs:.....	11
Cyclic/Acyclic Graphs.....	13
Calculating total number of paths:.....	13
Complete Code:	14
Outputs Obtained:.....	30
Challenges:	33
50 Nodes Complexity:	33
Cyclic Topology	33
Prospects:	33
Conclusion:	33
References.....	34

Problem Statement:

Construct a 50 nodes topology connected in such a way that each node must have 4 interfaces and all the 4 interfaces are connected to 4 other nodes. However, the source and destination nodes have 3 interfaces, and the source should be at least 30 nodes away from destination. Each link contains random value from 1 to 15. We need to identify the total number of paths from source to destination and select the shortest path among them. The number of hops for each path would also be calculated.

Introduction:

A routing algorithm is a procedure that lays down the route or path to transfer data packets from source to the destination. They help in directing Internet traffic efficiently. There are two main types of routing of algorithms:

1. Global for example Link State algorithm
2. Local for example Distance Vector algorithm

Link State algorithm uses Dijkstra's algorithm as its working principal; however, Distance Vector uses Bellman-Ford equations. We will be using Link State algorithm therefore would be implementing Dijkstra's algorithm.

Dijkstra's Algorithm:

It computes the least-cost path from one node(source) to all other nodes in the network. In addition to this, Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs.

Objectives:

The objectives of this project are:

- To learn and implement suitable routing algorithm to find solution of the problem statement
- To calculate the shortest path from source to destination by use of a suitable algorithm (Dijkstra/Bellman-Ford Equations).
- To calculate the number of hops data would take on the shortest path to reach destination from source.
- To calculate the cost of our shortest path.
- To calculate total number of paths from source to destination

Modules:

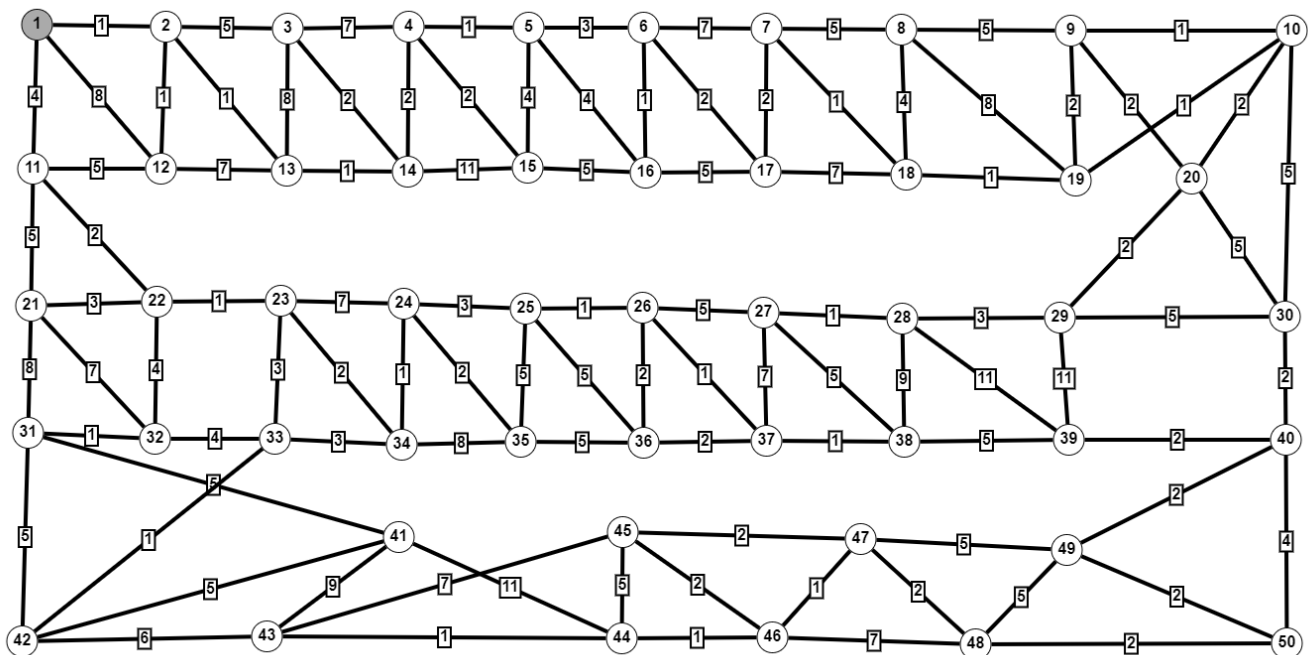
Our project implementation is divided into four major modules:

1. Making a network topology using the specifications mentioned in the problem statement
2. Calculating shortest path from the source node to destination node
3. Calculating the number of hops in the shortest path determined
4. Calculating total number of paths

1. Network topology:

Topology:

There are various online tools that can be used for construction of network topology. We have used packet tracer and eNSP software in the lab, however, constructing a 50-node network was extremely cumbersome on these software so we used an online tool. [1] The network topology we made is shown below



Network Topology with reference to the problem statement

Source Node: Node 1

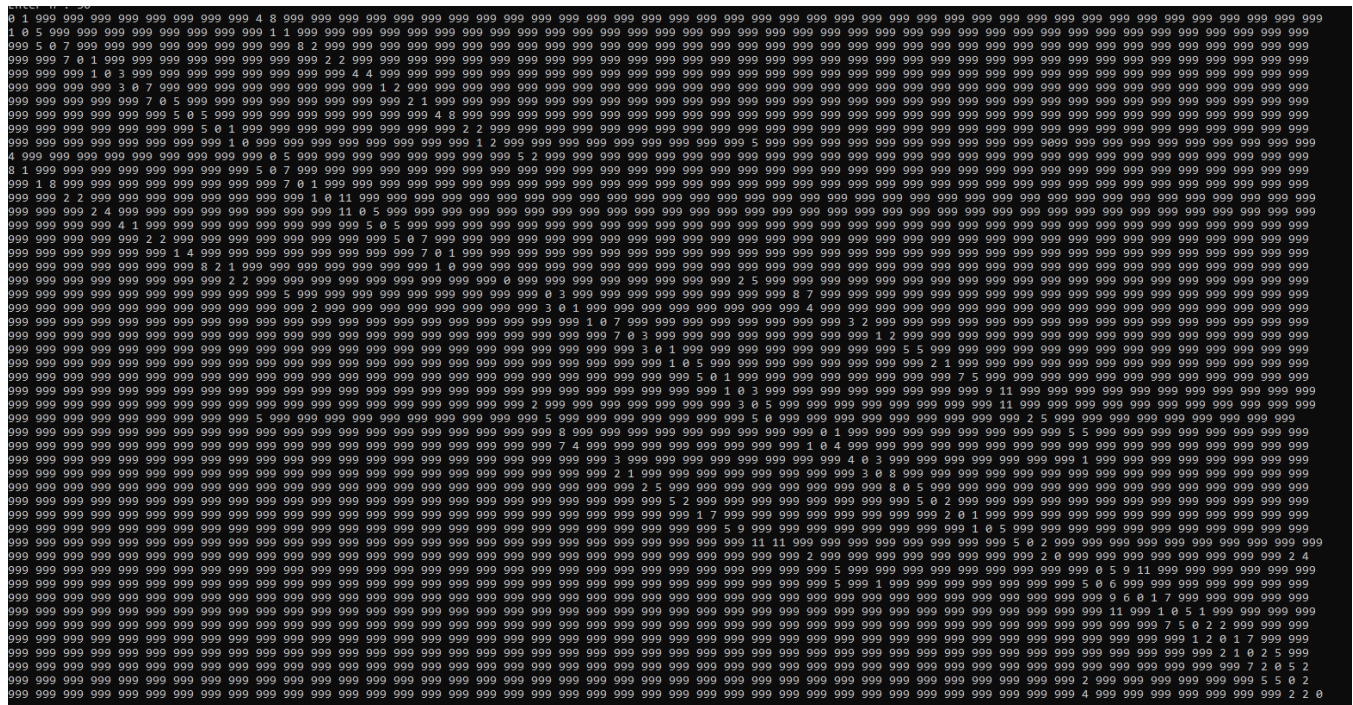
Destination Node: Node 50

Adjacency/Cost Matrix:[2]

An adjacency matrix is a way of representing a graph as a matrix of Booleans (0's and 1's). A finite graph can be represented in the form of a square matrix on a computer, where the Boolean value of the matrix indicates if there is a direct path between two vertices.

In case of multigraph representation, instead of entry 0 or 1, the entry will be between number of edges between two vertices. In case of weighted graph, the entries are weights of the edges between the vertices. The adjacency matrix for a weighted graph is called as cost adjacency matrix.

Attached is the screenshot of cost matrix and adjacency matrix for our topology



Cost Matrix for our topology

2. Calculating Shortest Path:

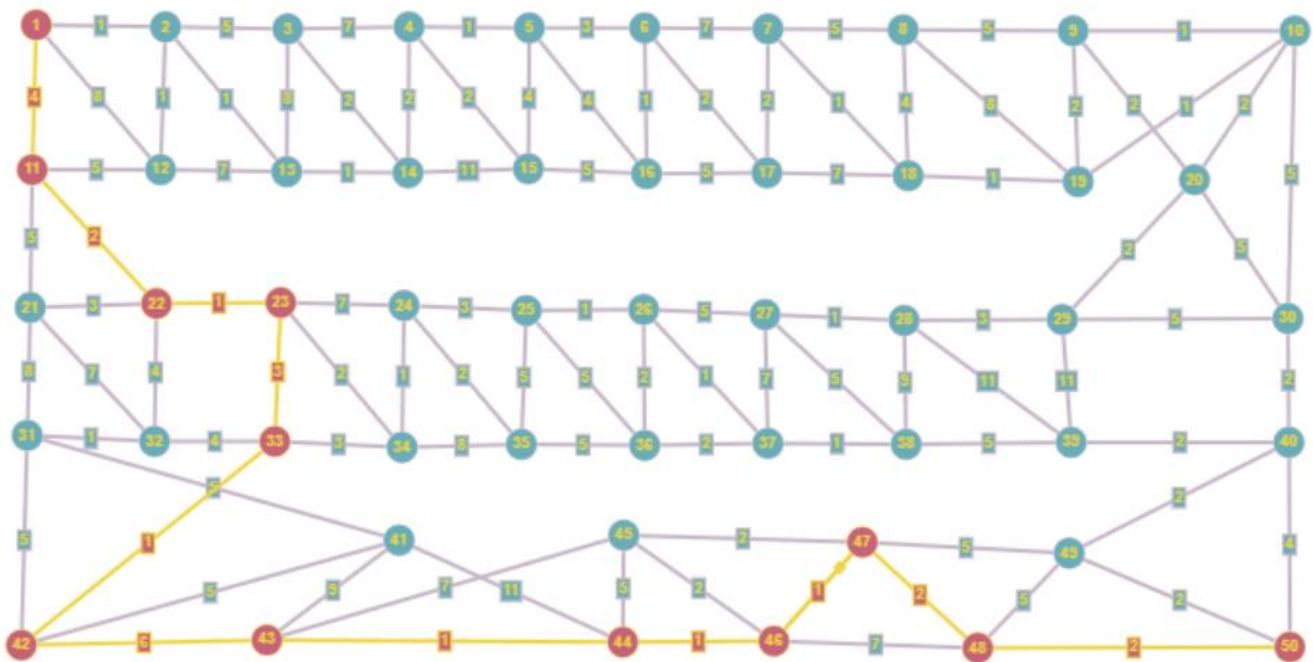
Shortest Path can be calculated using any suitable algorithm, however for Link State routing, Dijkstra's algorithm is used

Dijkstra's Algorithm:

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph. However, limitation of Dijkstra's algorithm is that it does not let us calculate the total number of paths from source to destination, but just only the shortest path. [3]

Results for shortest path by tool:

The same online tool we used for making the topology, was used to calculate the shortest path from source node to destination node. Attached is the result obtained



Shortest Path calculated by the online tool

Results for shortest path by our code:

Code for Dijkstra's algorithm was written in C++, the result matched with the result of the output shortest path gained by the online tool

```
::::Distance = 10
24 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 13
25 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 14
26 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 19
27 <- 27 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 20
28 <- 20 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
::::Distance = 20
29 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
::::Distance = 21
30 <- 32 <- 22 <- 11 <- 1 <-
::::Distance = 11
31 <- 22 <- 11 <- 1 <-
::::Distance = 10
32 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 10
33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 9
34 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 12
35 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 16
36 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 15
37 <- 37 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 16
38 <- 38 <- 37 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 21
39 <- 30 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
::::Distance = 23
40 <- 31 <- 32 <- 22 <- 11 <- 1 <-
::::Distance = 16
41 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 11
42 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 17
43 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 18
44 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 21
45 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 19
46 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 20
47 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 22
48 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 25
49 <- 48 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 24
```

Shortest Path calculated by our code

The path calculated by the online tool and the path calculated by our program is the same. However, arrays in C++ start from 0th position, therefore the topology may be referred to as starting node being 0 and ending node being 49.

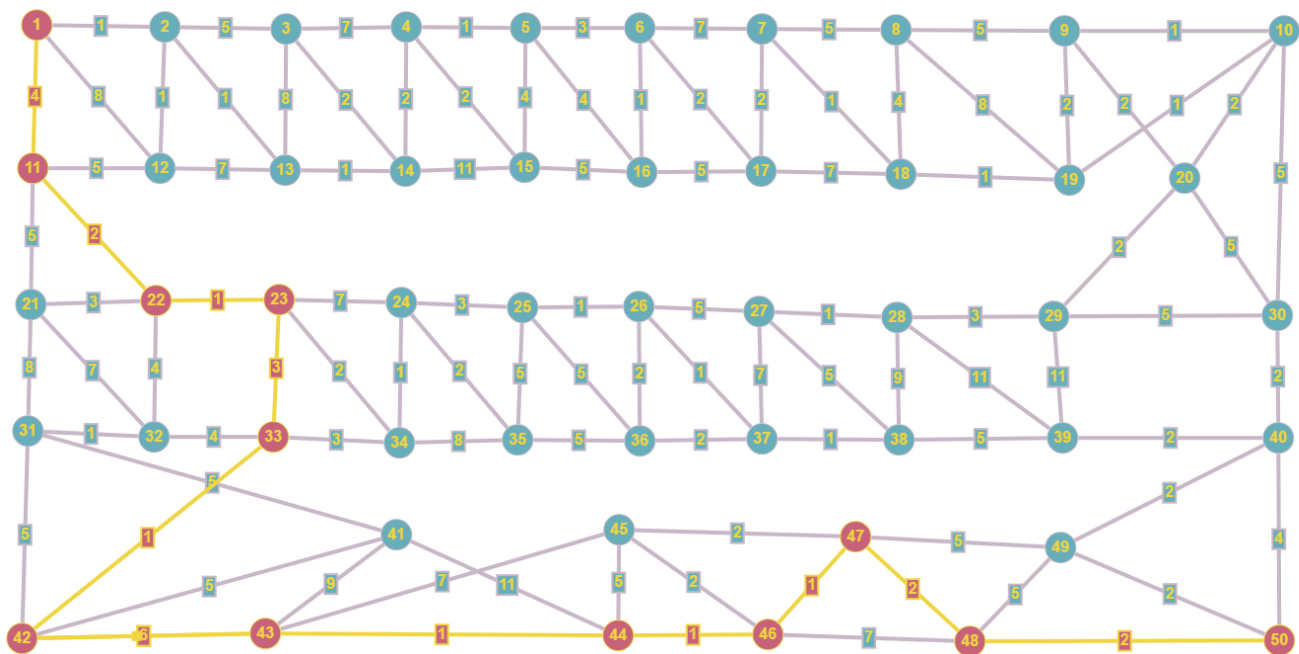
3. Calculating Number of Hops:

The number of hops in the shortest path can be calculated using the same algorithm that was used to calculate the shortest path, so again Dijkstra's algorithm is used

Results for shortest path by tool:

Results from the same online tool we used for making the topology, is attached

Shortest path length is 24: 1⇒11⇒22⇒23⇒33⇒42⇒43⇒44⇒46⇒47⇒48⇒50



Number of hops calculated by the online tool

Number of hops = 11

Results for shortest path by our code:

The same code was utilized to find out the number of hops and the output screenshot is attached

```
::::Distance = 10
24 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 13
25 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 14
26 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 19
27 <- 27 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 20
28 <- 20 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
::::Distance = 20
29 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
::::Distance = 21
30 <- 32 <- 22 <- 11 <- 1 <-
::::Distance = 11
31 <- 22 <- 11 <- 1 <-
::::Distance = 10
32 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 10
33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 9
34 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 12
35 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 16
36 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 15
37 <- 37 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 16
38 <- 38 <- 37 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 21
39 <- 30 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
::::Distance = 23
40 <- 31 <- 32 <- 22 <- 11 <- 1 <-
::::Distance = 16
41 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 11
42 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 17
43 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 18
44 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 21
45 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 19
46 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 20
47 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 22
48 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 25
49 <- 48 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
::::Distance = 24
```

Number of hops calculated by our code

Number of hops = 11

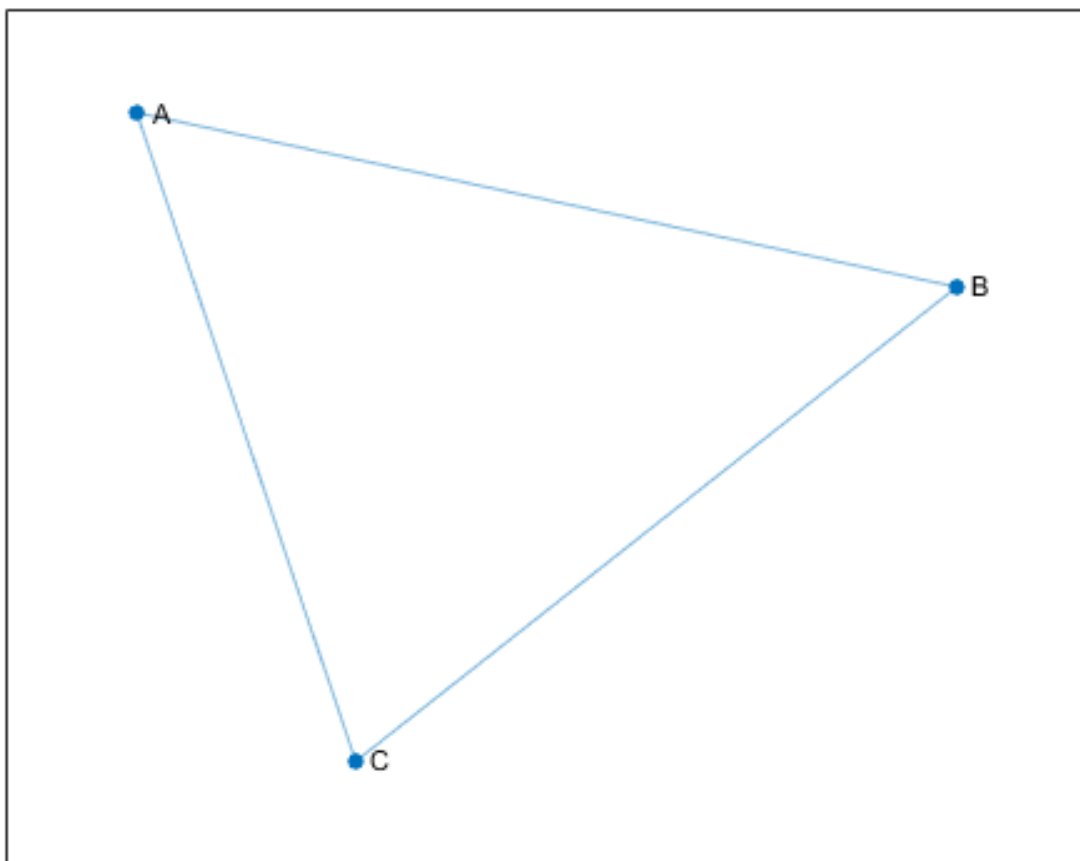
It can be seen that the number of hops calculated by our code and the online tool is same.

4. Calculating total number of paths

To understand how the total number of paths can be calculated for our topology, we need to understand the theory of directional/undirectional and cyclic/acyclic graphs.

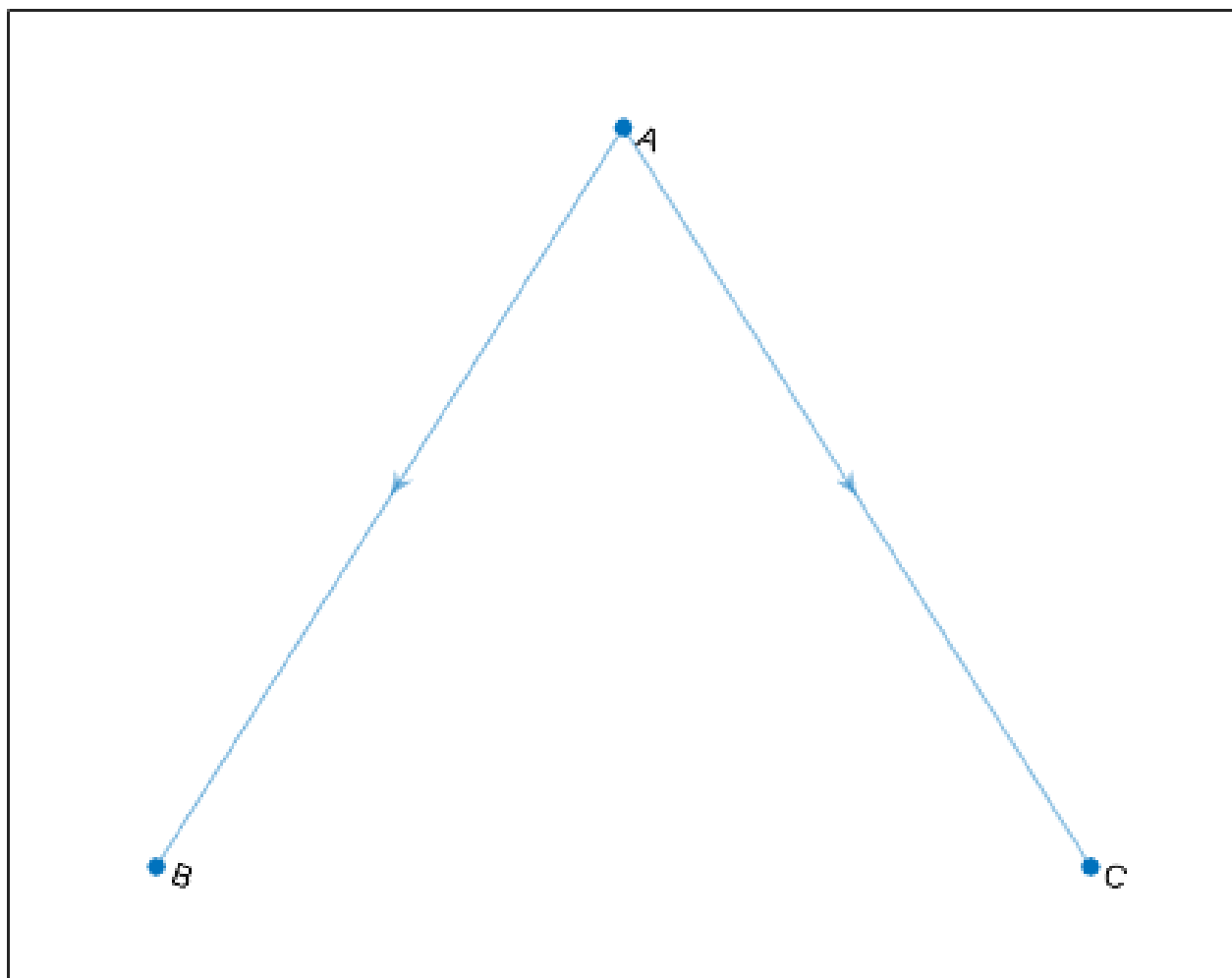
Directional/Undirectional graphs:

Undirected graphs have edges that do not have a direction. The edges indicate a two-way relationship, in that each edge can be traversed in both directions. This figure shows a simple undirected graph with three nodes and three edges.



Undirectional Graph

Directional graphs have edges with direction. The edges indicate a one-way relationship, in that each edge can only be traversed in a single direction. This figure shows a simple directed graph with three nodes and two edges.

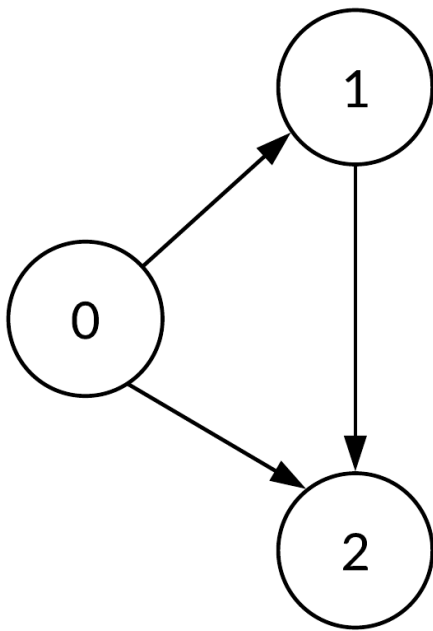


Directional Graph

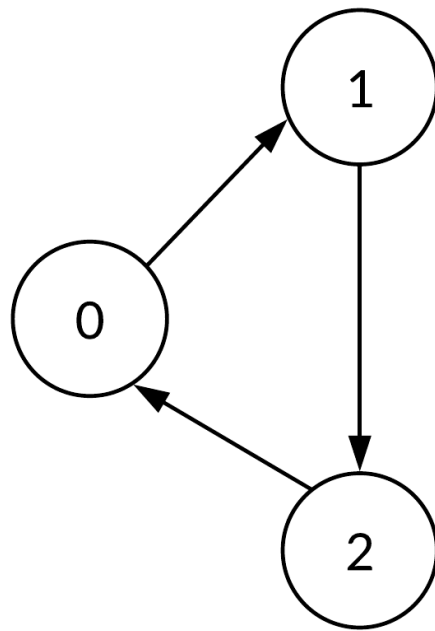
Cyclic/Acyclic Graphs

A cyclic graph is a graph containing at least one graph cycle. A graph that is not cyclic is said to be acyclic. [4]

Acyclic Graph



Cyclic Graph



Cyclic/Acyclic Graph

Calculating total number of paths:

A cyclic graph i.e., it is not a DAG (Directional Acyclic Graph) [5], the number of paths between two nodes can be infinite. [6]

Number of paths between source and node: Infinite (Because of looping)

Complete Code:

The topology we are constructing has 50 nodes therefore we constructed a 50x50 two-dimensional array. The nodes that are directly connected are assigned the values that we used in our original topology whereas the nodes that are not directly connected are assigned the value 999.

Then we implemented a function that was used to calculate the shortest path by iterative method. Further, we implemented another 50x50 array which was used to calculate the total number of paths from source to destination. The number of hops are calculated from source to each node for shortest path including the number of hops from source to destination. [7][8]

CODE:

```
#include<iostream>
#include <ctime>
#include <cstdlib>
#include<windows.h>
#include<conio.h>
using namespace std;

int cost[50][50], n;
const int p = 50;
const int q = 50;

int getMin(int dist[], bool visited[]) {
    int key = 0;
    int min = INT_MAX;
    for (int i = 0; i < n; i++) {
        if (!visited[i] && dist[i] < min) {
            min = dist[i];
            key = i;
        }
    }
    return key;
}

void display(int dist[], int par[]) {
    for (int i = 0; i < n; i++) {
        int temp = par[i];
        cout << i << " <- ";
        while (temp != -1)
        {
            cout << temp+1 << " <- ";
```



```

        temp = par[temp];
    }
    cout << endl;
    cout << "::::Distance = " << dist[i];
    cout << endl;
}
}

```

```

void dijkstra(int src, int cost[50][50]) {
    int par[100], dist[100];
    bool visited[100] = { 0 };
    fill(dist, dist + n, INT_MAX);

    dist[src] = 0;
    par[src] = -1;

    for (int g = 0; g < n - 1; g++) {
        int u = getMin(dist, visited);
        visited[u] = true;
        cout << " min = " << u << endl;
        for (int v = 0; v < n; v++) {
            if (!visited[v] && (dist[u] + cost[u][v]) < dist[v] && cost[u][v] != 9999)
            {
                par[v] = u;
                dist[v] = dist[u] + cost[u][v];
            }
        }
    }

    display(dist, par);
}

```

```

int Count_Path_Using_DFS(int m[p][q], int source, int destination) {
    static int A[50] = { 0 };
    //static array for checking a particular node is visited or not

    //for count a number of path
    static int c = 0;

    A[source] = 1; //marking of visited node

    if (m[source][destination] == 1) {
        c++;
        cout << c << endl;
    }
}

```

[illegible]

[illegible]

[illegible]


```
system("Color F0");  
cout << "Enter n : "; //Number of Nodes  
cin >> n;  
  
srand(time(0));  
int rand_var[102]; //Random value array  
int cost[50][50];  
  
int select;  
cout<<"Random cost matrix? 1(Y) 2(N): ";  
cin>>select;  
  
for (int x = 0; x <100 ; x++)  
  
{  
    rand_var[x] = 1 + (rand() % 15);  
}  
  
/*Cost Matrix with link values as provided in our assignment*/  
int costassignment[50][50]={  
    {0, 1, 999, 999, 999, 999, 999, 999, 999, 999, 999, 4, 8, 999, 999, 999, 999, 999, 999, 999, 999,  
    999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999,  
    999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999},  
    {  
        1,0,5,999,999,999,999,999,999,999,999,999,1,1,999,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999 },  
    {  
        999,5,0,7,999,999,999,999,999,999,999,999,999,8,2,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999 },  
    {  
        999,999,7,0,1,999,999,999,999,999,999,999,999,999,2,2,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999 },  
    {  
        999,999,999,1,0,3,999,999,999,999,999,999,999,999,999,4,4,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999 },  
    {  
        999,999,999,999,3,0,7,999,999,999,999,999,999,999,999,999,1,2,999,999,999,999,999,999,999,  
        ,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
        ,999,999,999,999,999 },  
    {  
        999,999,999,999,999,7,0,5,999,999,999,999,999,999,999,999,999,2,1,999,999,999,999,999,999
```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]


```
999,999,rand_var[48],0,rand_var[52],999,999,999,999,999,999,999,rand_var[53],rand_v  
ar[55],999,999,999,999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,999,999,rand_var[52],0,rand_var[56],999,999,999,999,999,999,999,rand_var[57],ra  
nd_var[58],999,999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,999,999,rand_var[56],0,rand_var[59],999,999,999,999,999,999,999,rand_var[6  
0],rand_var[61],999,999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,999,999,rand_var[59],0,rand_var[62],999,999,999,999,999,999,999,rand_v  
ar[63],rand_var[64],999,999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var  
[40],999,999,999,999,999,999,999,rand_var[62],0,rand_var[65],999,999,999,999,999,99  
9,999,rand_var[66],999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,rand_var[39],999,999,999,999,999,999,999,999,  
rand_var[41],999,999,999,999,999,999,999,rand_var[65],0,999,999,999,999,999,99  
9,999,999,rand_var[67],rand_var[68],999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand  
_var[43],999,999,999,999,999,999,999,999,999,999,999,999,0,rand_var[69],999,999,999,999,999,99  
9,999,rand_var[68],rand_var[70],999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand  
_var[44],rand_var[47],999,999,999,999,999,999,999,rand_var[69],0,rand_var[71],999,99  
9,999,999,999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,rand_var[50],999,999,999,999,999,999,999,rand_var[71],0,rand_var[72],999,999,99  
9,999,999,999,999,rand_var[73],999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,rand_var[51],rand_var[49],999,999,999,999,999,999,999,rand_var[72],0,rand_var[7  
5],999,999,999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,999,rand_var[54],rand_var[53],999,999,999,999,999,999,999,rand_var[75],0,rand_v  
ar[76],999,999,999,999,999,999,999,999,999,999,999,999,999 },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,999,999,rand_var[55],rand_var[57],999,999,999,999,999,999,999,rand_var[76],0,ra  
nd_var[77],999,999,999,999,999,999,999,999,999,999,999,999,999 },
```

```
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[55],rand_var[60],999,999,999,999,999,999,999,rand_var[77],
0,rand_var[78],999,999,999,999,999,999,999,999,999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[61],rand_var[63],999,999,999,999,999,999,999,rand_var
[78],0,rand_var[79],999,999,999,999,999,999,999,999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[64],rand_var[66],999,999,999,999,999,999,999,rand
_var[79],0,rand_var[80],999,999,999,999,999,999,999,999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[67],999,999,999,999,999,999,999,rand_var
[80],0,999,999,999,999,999,999,999,rand_var[81],rand_var[82] },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[68],999,999,999,999,999,999,999,rand_var[83],rand_var[84],rand_var[85],999,999,999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[70],999,rand_var[73],999,999,999,999,999,
999,999,rand_var[83],0,rand_var[86],999,999,999,999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[74],999,rand_var[75],999,999,999,999,999,
rand_var[84],rand_var[86],0,rand_var[87],rand_var[89],999,999,999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[75],999,rand_var[87],0,rand_var[88],rand_var[90],999,999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[76],999,rand_var[88],0,rand_var[91],rand_var[92],999,999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[77],999,rand_var[89],0,rand_var[93],rand_var[94],999,999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[78],999,rand_var[90],0,rand_var[95],rand_var[96],999 },
{
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[79],999,rand_var[91],0,rand_var[97],rand_var[98],999 }
}
```

```
999,999,999,rand_var[94],rand_var[95],0,rand_var[97],rand_var[98] },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[81],  
999,999,999,999,999,999,rand_var[96],rand_var[97],0,rand_var[99] },  
{  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,  
999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,rand_var[82],  
999,999,999,999,999,999,rand_var[98],rand_var[99],0 },  
};  
  
if (select==1) //Selecting between random and hard coded cost matrix  
{  
  
    for (int x=0 ; x < 50; x++)  
{  
for (int y=0 ; y < 50; y++)  
{  
cost[x][y] = costrandom[x][y];  
}  
}  
}  
else if (select==2){  
  
    for (int x=0 ; x < 50; x++)  
{  
for (int y=0 ; y < 50; y++)  
{  
cost[x][y] = costassignment[x][y];  
}  
}  
}  
  
for (int i = 0; i < 50; i++)  
{  
for (int j = 0; j < 50; j++)  
{  
cout << cost[i][j] << " ";  
}  
  
cout << endl;  
}  
  
int src, temp;
```

```

cout << "\nEnter source : "; cin >> src;
dijkstra(src, cost);
cout << "\n\n" << endl;

cout << "Press 1 to get all paths : "; cin >> temp;
if (temp == 1)
    allpath();
}

```

Outputs Obtained:

OUTPUTS:

The shortest path shown in this output is same as that of assignment:

"D:\Downloads\Source (1).exe"	"D:\Downloads\Source (1).exe"
0 <-	25 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 0	Distance = 14
1 <- 1 <-	26 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 1	Distance = 19
2 <- 14 <- 13 <- 2 <- 1 <-	27 <- 27 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 5	Distance = 20
3 <- 14 <- 13 <- 2 <- 1 <-	28 <- 20 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
Distance = 5	Distance = 20
4 <- 4 <- 14 <- 13 <- 2 <- 1 <-	29 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
Distance = 6	Distance = 21
5 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	30 <- 32 <- 22 <- 11 <- 1 <-
Distance = 9	Distance = 11
6 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	31 <- 22 <- 11 <- 1 <-
Distance = 13	Distance = 10
7 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	32 <- 23 <- 22 <- 11 <- 1 <-
Distance = 18	Distance = 10
8 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 17	Distance = 9
9 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	34 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 16	Distance = 12
10 <- 1 <-	35 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 4	Distance = 16
11 <- 2 <- 1 <-	36 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 2	Distance = 15
12 <- 2 <- 1 <-	37 <- 37 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 2	Distance = 16
13 <- 13 <- 2 <- 1 <-	38 <- 38 <- 37 <- 26 <- 25 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-
Distance = 3	Distance = 21
14 <- 4 <- 14 <- 13 <- 2 <- 1 <-	39 <- 30 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-
Distance = 7	Distance = 23
15 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	40 <- 31 <- 32 <- 22 <- 11 <- 1 <-
Distance = 10	Distance = 16
16 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	41 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 11	Distance = 11
17 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	42 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 14	Distance = 17
18 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	43 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 15	Distance = 18
19 <- 10 <- 19 <- 18 <- 7 <- 17 <- 6 <- 5 <- 4 <- 14 <- 13 <- 2 <- 1 <-	44 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 18	Distance = 21
20 <- 11 <- 1 <-	45 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 9	Distance = 19
21 <- 11 <- 1 <-	46 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 6	Distance = 20
22 <- 22 <- 11 <- 1 <-	47 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 7	Distance = 22
23 <- 34 <- 23 <- 22 <- 11 <- 1 <-	48 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 10	Distance = 25
24 <- 24 <- 34 <- 23 <- 22 <- 11 <- 1 <-	49 <- 48 <- 47 <- 46 <- 44 <- 43 <- 42 <- 33 <- 23 <- 22 <- 11 <- 1 <-
Distance = 13	Distance = 24

For random cost matrix:

"D:\Downloads\Source (1).exe"	"D:\Downloads\Source (1).exe"
0 <-	25 <- 36 <- 35 <- 24 <- 23 <- 22 <- 11 <- 1 <-
Distance = 0	Distance = 56
1 <- 1 <-	26 <- 38 <- 39 <- 29 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 14	Distance = 62
2 <- 2 <- 1 <-	27 <- 38 <- 39 <- 29 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 26	Distance = 58
3 <- 3 <- 2 <- 1 <-	28 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 29	Distance = 50
4 <- 4 <- 3 <- 2 <- 1 <-	29 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 32	Distance = 48
5 <- 5 <- 4 <- 3 <- 2 <- 1 <-	30 <- 21 <- 11 <- 1 <-
Distance = 42	Distance = 28
6 <- 6 <- 5 <- 4 <- 3 <- 2 <- 1 <-	31 <- 21 <- 11 <- 1 <-
Distance = 57	Distance = 34
7 <- 7 <- 6 <- 5 <- 4 <- 3 <- 2 <- 1 <-	32 <- 32 <- 21 <- 11 <- 1 <-
Distance = 67	Distance = 46
8 <- 20 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-	33 <- 33 <- 32 <- 21 <- 11 <- 1 <-
Distance = 61	Distance = 49
9 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-	34 <- 24 <- 23 <- 22 <- 11 <- 1 <-
Distance = 63	Distance = 46
10 <- 1 <-	35 <- 35 <- 24 <- 23 <- 22 <- 11 <- 1 <-
Distance = 15	Distance = 51
11 <- 1 <-	36 <- 27 <- 38 <- 39 <- 29 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 15	Distance = 63
12 <- 12 <- 1 <-	37 <- 39 <- 29 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 18	Distance = 57
13 <- 3 <- 2 <- 1 <-	38 <- 29 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 27	Distance = 54
14 <- 4 <- 3 <- 2 <- 1 <-	39 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 35	Distance = 56
15 <- 15 <- 4 <- 3 <- 2 <- 1 <-	40 <- 31 <- 21 <- 11 <- 1 <-
Distance = 37	Distance = 38
16 <- 16 <- 15 <- 4 <- 3 <- 2 <- 1 <-	41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 51	Distance = 40
17 <- 17 <- 16 <- 15 <- 4 <- 3 <- 2 <- 1 <-	42 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 60	Distance = 44
18 <- 9 <- 20 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-	43 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 66	Distance = 49
19 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-	44 <- 44 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 55	Distance = 50
20 <- 11 <- 1 <-	45 <- 44 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 21	Distance = 52
21 <- 11 <- 1 <-	46 <- 46 <- 44 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 26	Distance = 54
22 <- 22 <- 11 <- 1 <-	47 <- 47 <- 46 <- 44 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 39	Distance = 66
23 <- 23 <- 22 <- 11 <- 1 <-	48 <- 47 <- 46 <- 44 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 44	Distance = 56
24 <- 24 <- 23 <- 22 <- 11 <- 1 <-	49 <- 40 <- 30 <- 41 <- 31 <- 21 <- 11 <- 1 <-
Distance = 51	Distance = 59

Infinite loop when trying to find total number of paths:

Select "D:\Downloads\Source (1).exe"

200991
200992
200993
200994
200995
200996
200997
200998
200999
201000
201001
201002
201003
201004
201005
201006
201007
201008
201009
201010
201011
201012
201013
201014
201015
201016
201017
201018
201019
201020
201021
201022
201023
201024
201025
201026
201027
201028
201029
201030
201031
201032
201033
201034
201035
201036
201037
201038
201039

Challenges:

50 Nodes Complexity:

The number of nodes were 50 which turned out be a major constraint as we needed to implement 50x50 array which resulted in 2500 elements array. We filled this matrix manually and then we repeated this process 10 times to meet our project requirement.

Cyclic Topology

Our topology was **non-directional** and **cyclic** due to which it became impossible to calculate all the paths as our output from the code began running infinitely. However, if our topology was directional and contained a smaller number of nodes, we would have been able to calculate this also.

Prospects:

Prospects of our Project include:

- Decreasing Time and Space Complexity of our Code
- Implementation of Image Processing based IIOT device that takes in Topology Diagram to give desired outputs
- Implementation of Image Processing based IIOT device that takes in Topology Diagram to give desired outputs
- ML based predictions for efficient routing [9]

Conclusion:

We were able to successfully calculate the shortest path, distance, and the number of hops from source to all other destinations. However, we were unable to calculate the total number of paths and the number of hops in each path due to the reasons mentioned above. Finally, we were able to implement our problem on software to get the required solutions and overall, it was a good learning experience for us.

References

- [1] Graphonline.ru. 2022. *Create Graph online and find shortest path or use other algorithm*. [online] Available at: <<https://graphonline.ru/en/#>> [Accessed 24 May 2022].
- [2] <https://www.programiz.com/dsa/graph-adjacency-matrix#:~:text=An%20adjacency%20matrix%20is%20a,direct%20path%20between%20two%20vertices>.
- [3] Programiz.com. 2022. *Dijkstra's Algorithm*. [online] Available at: <<https://www.programiz.com/dsa/dijkstra-algorithm>> [Accessed 24 May 2022].
- [4] Mathworks. 2022. [online] Available at: <<https://www.mathworks.com/help/matlab/math/directed-and-undirectedgraphs.html#:~:text=Undirected%20graphs%20have%20edges%20that,graphs%20have%20edges%20with%20direction.>>> [Accessed 24 May 2022].
- [5] En.wikipedia.org. 2022. *Directed acyclic graph - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Directed_acyclic_graph> [Accessed 24 May 2022].
- [6] Hassan, O., 2022. *Find all paths between two graph nodes*. [online] Stack Overflow. Available at: <<https://stackoverflow.com/questions/9535819/find-all-paths-between-two-graph-nodes>> [Accessed 24 May 2022].
- [7] Github. 2022. *Dijkstra Algorithm for finding shortest path to all vertices from a single source..* [online] Available at: <https://gist.github.com/nateshmbhat/1a79daa00a148e44ad79c3e338977440#file-dijkstra_algorithm-cpp> [Accessed 24 May 2022].
- [8] 2022. [online] Available at: <[https://www.simplilearn.com/tutorials/cpp-tutorial/random-number-generator-incpp#:~:text=How%20to%20Generate%20Random%20Numbers%20in%20C%2B%2B%20With%20a%20Range,\(rand\(\)%20%25%20100\).>](https://www.simplilearn.com/tutorials/cpp-tutorial/random-number-generator-incpp#:~:text=How%20to%20Generate%20Random%20Numbers%20in%20C%2B%2B%20With%20a%20Range,(rand()%20%25%20100).>)> [Accessed 24 May 2022].
- [9] Sharma, Deepak & Dhurandher, Sanjay & Woungang, Isaac & Srivastava, Rohit & Mohananey, Anhad & Rodrigues, Joel. (2016). A Machine Learning-Based Protocol for Efficient Routing in Opportunistic Networks. IEEE Systems Journal. PP. 10.1109/JSYST.2016.2630923.