

# LAN INFRASTRUCTURE DESIGN

**Submitted by: Ayush Karn**

**Roll: 2K19/CO/454**

**Course: Discrete Structure**

**Submitted to: Mr Ajay Kumar**

## DECLARATION

I hereby declare that project report entitled “LAN INFRASTRUCTURE DESIGN “ submitted by me (**Ayush Karn 2K19/CO/454**) to Delhi Technological University (DTU), Delhi is a record of original work done under the guidance of **Mr. Ajay Kumar** for the course of Discrete Structure. All the codes and implementations are completely written by me.

Name: Ayush Karn

Roll No: 2K19/CO/454

Submitted to: Mr. Ajay Kumar

# CERTIFICATE

This is to certify that Ayush Karn of A6 batch Computer Engineering Department (COE) having Roll No 2K19/CO/454 has successfully completed the project work entitled “LAN INFRASTRUCTURE DESIGN” on Discrete Structures for Third Semester which is to be evaluated as the Mid Term Component.

Signature: Ayush Karn

Date: 22-11-2020

## ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teacher Mr. Ajay Kumar as well as our college (Delhi Technological University, Delhi) which gave me the golden opportunity to do this wonderful project on the topic “LAN INFRASTRUCTURE DESIGN”, which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to them.

Secondly i would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

## ABSTRACT

As we all know how important communication system for human being is. In this project I have developed software which is console based application. It makes it easy to find shortest distance between two routers, expenditure on the cable used, whether or not optical fiber is required, total expenditure etc.

All the expenses are calculated based on the distance that we find using the Dijkstra Algorithm. The use of optical fiber is necessary or not is also determined using the same. The graph traversal technique used is Breadth First Search which is similar to Breath First search technique used in trees but in graph we have to keep track of the visited vertices as we don't want to visit same vertex again and again and be stuck in a loop.

If  $\text{distance} < 10$  the twisted pair cable will be required.

If  $\text{distance} \geq 10$  and  $\leq 50$  Coaxial Cable, Repeater will be required.

If  $\text{distance} > 50$  Optical Fiber will be required. There are also other conditions that are mentioned in the code.

The Compiler used for this project is MINGW and IDE used was Atom. I have also tried to make the output look colorful and given some subtle animations to it.

# TABLE OF CONTENT

Chapter 1: What is Graph Theory?

Chapter 2: Dijkstra Algorithm

Chapter 3: Code and output

Chapter 4: Conclusion

# Chapter1: What is Graph Theory?

In mathematics, graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of vertices (also called nodes or points) which are connected by edges (also called links or lines). A distinction is made between undirected graphs, where edges link two vertices symmetrically, and directed graphs, where edges link two vertices asymmetrically; see Graph (discrete mathematics) for more detailed definitions and for other variations in the types of graph that are commonly considered. Graphs are one of the prime objects of study in discrete mathematics.

In one restricted but very common sense of the term, a graph is an ordered pair  $G = (V, E)$  comprising:

- 1)  $V$ , a set of vertices called nodes
- 2)  $E$ , a set of edges (also called links or lines), which are unordered pairs of vertices (that is, an edge is associated with two distinct vertices).

## Characteristics of graphs:

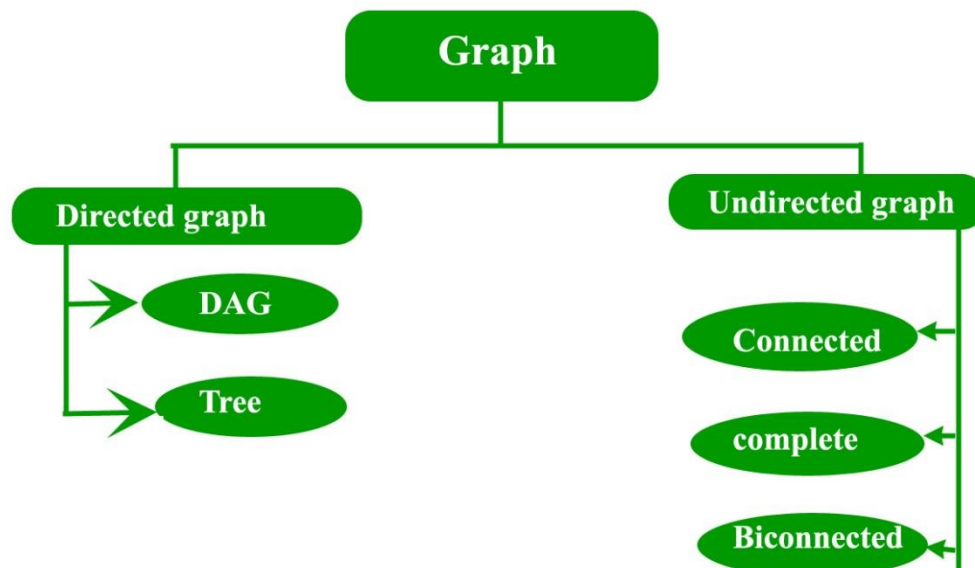
- 1) **Adjacent node:** A node 'v' is said to be adjacent node of node 'u' if and only if there exists an edge between 'u' and 'v'.
- 2) **Degree of a node:** In an undirected graph the number of nodes incident on a node is the degree of the node. In case of directed graph, IN degree of the node is the number of arriving edges to a node. Out degree of the node is the number of departing edges to a node.
- 3) **Path:** A path of length 'n' from node 'u' to node 'v' is defined as sequence of  $n+1$  node.

$$P(u,v)=(v_0,v_1,v_2,v_3,\dots,v_n)$$

A path is simple if all the nodes are distinct, exception is source and destination is same.

4) **Isolated node:** A node with degree 0 is known as isolated node. Isolated node can be found by Breadth first search (BFS). It finds its application in LAN network in finding whether a system is connected or not.

## Types of graphs:



### Directed graph:

A graph in which the direction of the edge is defined to a particular node is a directed graph.

**Directed Acyclic graph:** It is a directed graph with no cycle. For a vertex 'v' in DAG there is no directed edge starting and ending with vertex 'v'.

**Tree:** A tree is just a restricted form of graph. That is, it is a DAG with a restriction that a child can have only one parent.

### Undirected graph:

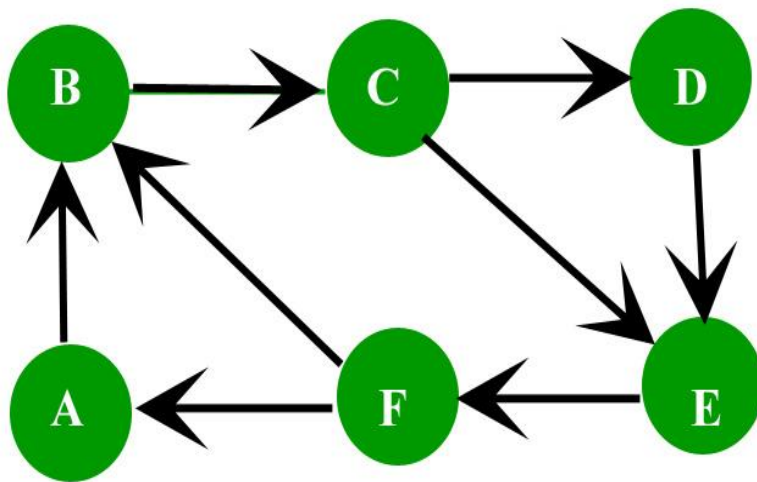
A graph in which the direction of the edge is not defined. So if an edge exists between node 'u' and 'v', then there is a path from node 'u' to 'v' and vice versa.

**Connected graph:** A graph is connected when there is a path between every pair of vertices. In a connected graph there is no unreachable node.

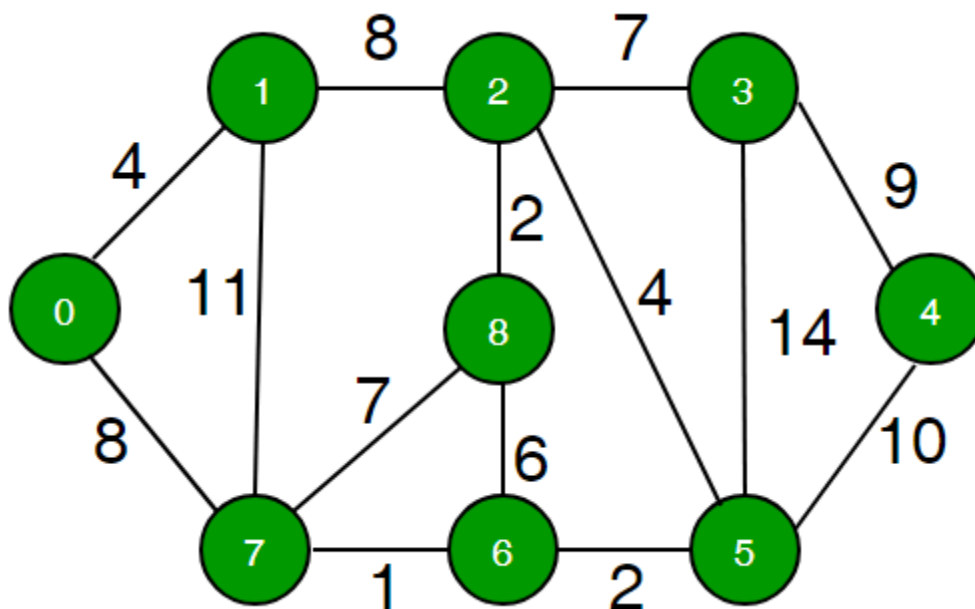


**Complete graph:** A graph in which each pair of graph vertices is connected by an edge. In other words, every node 'u' is adjacent to every other node 'v' in graph 'G'. A complete graph would have  $n(n-1)/2$  edges. See below for proof.

**Bi-connected graph:** A connected graph which cannot be broken down into any further pieces by deletion of any vertex. It is a graph with no articulation point.



**Directed Graph**



**Undirected Graph**

## Chapter 2: Dijkstra Algorithm

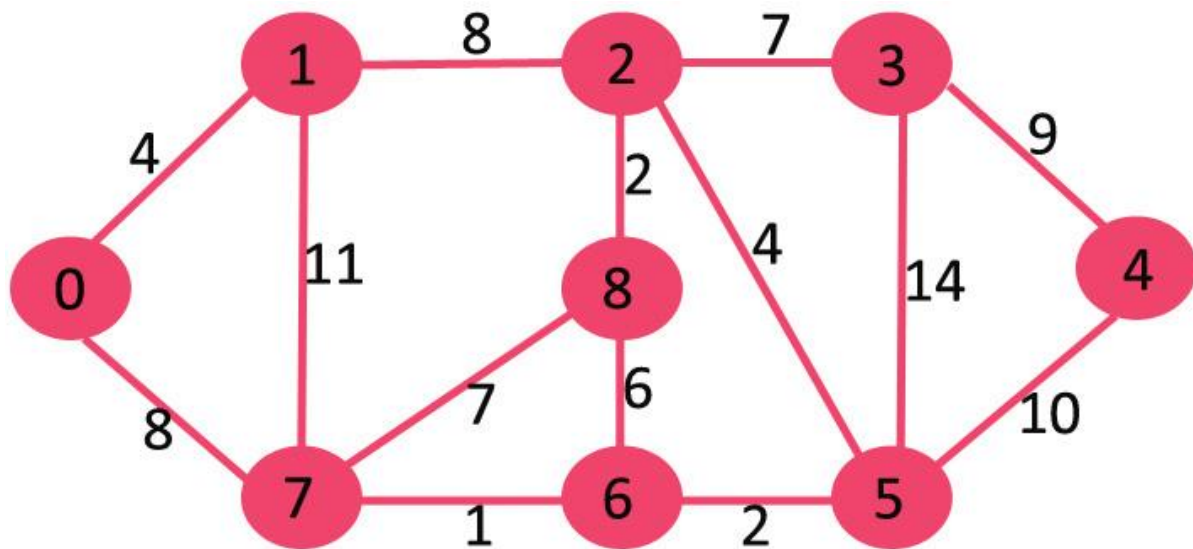
**Dijkstra algorithm** is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Below are the detailed steps used in Dijkstra algorithm to find the shortest path from a single source vertex to all other vertices in the given graph. Worst case time complexity of this Algorithm is  $n^2$

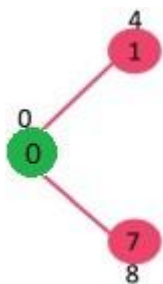
### ALGORITHM

- 1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
- 2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
- 3) While sptSet doesn't include all vertices
  - a) Pick a vertex u which is not there in sptSet and has minimum distance value.
  - b) Include u to sptSet.
  - c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

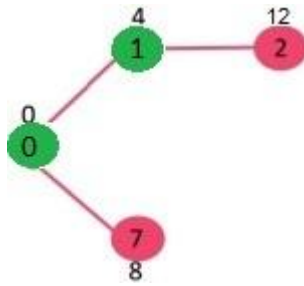
Let us understand with the following example:



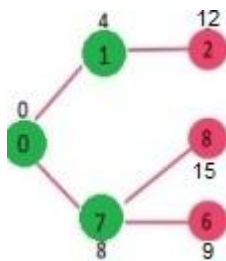
The set `sptSet` is initially empty and distances assigned to vertices are  $\{0, \text{INF}, \text{INF}, \text{INF}, \text{INF}, \text{INF}, \text{INF}, \text{INF}\}$  where `INF` indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in `sptSet`. So `sptSet` becomes  $\{0\}$ . After including 0 to `sptSet`, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following sub graph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green colour.



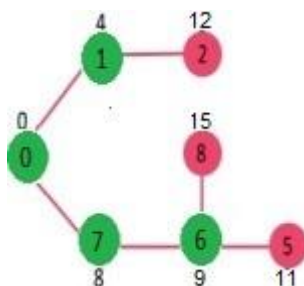
Pick the vertex with minimum distance value and not already included in SPT (not in `sptSet`). The vertex 1 is picked and added to `sptSet`. So `sptSet` now becomes  $\{0, 1\}$ . Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



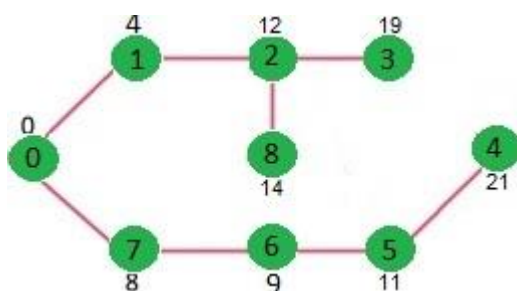
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



We repeat the above steps until sptSet does include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).



## **Advantages:-**

- 1) It is used in Google Maps
- 2) It is used in finding Shortest Path.
- 3) It is used in geographical Maps
- 4) To find locations of Map this refers to vertices of graph.
- 5) Distance between the locations refers to edges.
- 6) It is used in IP routing to find Open shortest Path First.
- 7) It is used in the telephone network.

## **Disadvantages:-**

- 1) It does blind search so it wastes lot of time while processing.
- 2) It cannot handle negative edges.
- 3) This leads to acyclic graphs and most often cannot obtain the right shortest path.

**Worst Time Complexity= $n^2$**

## Chapter 3: Code and Output

### Code:

```
#include<iostream>

#include<windows.h>

#include<stdlib.h>

using namespace std;

#define MAX 10

#define INF 99999


void graph();

void aboutproject();

void help();

void trademark();

void opening();

void graphic();

void menu()

{


system("cls");

system("COLOR DF");

int c;

//char ch='y';
```

```
do
{
cout<<"\n\t LAN INFRASTRUCTURE DESIGN";
cout<<"\n\t 1. ABOUT PROJECT";
cout<<"\n\t 2. GO To SOFTWARE ";
cout<<"\n\t 3. HELP";
cout<<"\n\t 4. EXIT";
cout<<"\n\t Enter your choice =>";
cin>>c;
switch(c)
{
case 1:aboutproject();
break;
case 2:graph();
break;
case 3:help();
break;
case 4:trademark();
cout<<"\n";
cout<<"\n";
exit(system("pause"));
break;
default: cout<<"\n\t Wrong choice";
```

```

cout<<"\n\t Please Enter again";

}

}while(1);

}

void aboutproject(){
    system("cls");
    system("COLOR 4F");
    cout<<"\n\n\t\t\t\t\t===== WELCOME TO LAN
INFRASTRUCTURE DESIGN =====";

    Sleep(100);
    cout<<"\n\n\t\t\t\t\t*****SOFTWARE DETAILS*****";
    Sleep(100);
    cout<<"\n\n\t\t\t\t\tDevloper          : Ayush Karn";
    Sleep(100);
    cout<<"\n\n\t\t\t\t\tProgramming Language   : C++";
    Sleep(10);
    cout<<"\n\n\t\t\t\t\tCompiler Version / IDE      : MINGW
/ ATOM";
    Sleep(100);
    cout<<"\n\n\t\t\t\t\tThank You for trying this program";
    Sleep(100);

```



```
        cout<<"\n\n";

        system("pause");

        menu();
    }

    void help(){

        system("cls");

        system("COLOR 8F");

        cout<<"\n\t\t\t\t\t==== HELP =====";
```

```
cout<<"\n\t\t\t\t\tWELCOME TO LAN INFRASTRUCTURE  
DESIGN ";
```

```
Sleep(10);
```

```
cout<<"\n\t\t\t\t\tIn this Project you can find shortest distance b/w  
two building" ;
```

```
Sleep(10);
```

```
cout<<"\n\t\t\t\t\t-In first option you can see the information about us"  
;
```

```
Sleep(10);
```

```
cout<<"\n\t\t\t\t\t-In second option you can operate the software." ;
```

```
Sleep(10);
```

```
cout<<"\n\t\t-In third option you can view Help for the project";
```

```
Sleep(10);
```

```
cout<<"\n\t\t- In Fourth option you can EXIT." ;
```

```
Sleep(10);
```

```
cout<<" Press any key to continue ";
```

```
for (int i=25; i>=1; i--)
```

```
{
```

```
Sleep(20);
```

```
}
```

```
    cout<<"\n";
```

```
    cout<<"\n";
```

```
    system("pause");
```

```
    menu();
```

```
}
```

```
void gotoxy(short x, short y)          //definition of gotoxy  
function//
```

```
{
```

```
COORD pos = {x,y};
```

```
SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
```

```
}
```

```
void trademark()
```

```
{
```

```
    system("cls");
```

```
    system("COLOR 8F");
```

```
    int j;
```

```
cout<<"\n\n\n\n\n\t LAN INFRASTRUCTURE DESIGN " ;
```

```
    cout<<"\n-----" ;
```

```
cout<<"\n\t\t\t AN ART'S PRODUCTION" ;
```

```
cout<<"\n\t\t\t -----" ;
```

```
cout<<"\n\n\t\t\t COPYRIGHT @ 2020" ;
```

```
cout<<"\n\t\t\t -----" ;
```

```
cout<<"\n\n\n\t\t\t T H A N K S F O R U S I N G" ;
```

```
cout<<"\n\n\n\t\t\t MY PROGRAM" ;
```

```
cout<<"\n\t\t\t =====" ;
```

```
cout<<"\n\t\t\t BY AYUSH KARN";
```

```
for(j=6;j<=65;j++)
```

```

{
    gotoxy(j,7);
    putchar(205);
    gotoxy(j,23);
    putchar(205);
}

gotoxy(6,7);    putchar(201);    //LOOPS FOR BORDERS//
gotoxy(65,7);
putchar(187);
gotoxy(6,23);
putchar(200);
gotoxy(65,23);
putchar(188);
for(j=8;j<=22;j++)
{
    gotoxy(6,j);
    putchar(186);
    gotoxy(65,j);
    putchar(186);
}

Sleep(200);    //GIVING DELAY//
}

void dijk(int G[MAX][MAX],int n,int start){

```

```
system("cls");
system("COLOR 9F");
int cost[MAX][MAX];
int dist[MAX];
int visited[MAX];
int pred[MAX];
int i,j;
int count;
int mindist;
int nextnode;
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        if(G[i][j]==0){
            cost[i][j]=INF;
        }
        else
        {
            cost[i][j]=G[i][j];
        }
    }
}
for(i=0;i<n;i++){
    dist[i]=cost[start][i];
```

```

    pred[i]=start;
    visited[i]=0;
}
dist[start]=0;
visited[start]=1;
count=1;
while(count<(n-1))
{ mindist=INF;
  for(i=0;i<n;i++){
    if((dist[i]<mindist)&&(!visited[i])){
      mindist=dist[i];
      nextnode=i;
    }
  }
  visited[nextnode]=1;
  for(i=0;i<n;i++){
    if(!visited[i]){
      if((mindist+cost[nextnode][i])<dist[i]){

dist[i]=mindist+cost[nextnode][i]; //formula
      pred[i]=nextnode;

```

```

        }
    }
}
count++;
}
for(i=0;i<n;i++){
    if(i!=start){
        cout<<"\n\t The distance of Building from
"<<start<<" to "<<i<<" is: "<<dist[i];
        cout<<"\n\t The path is : "<<i;
        j=i;
        do{
            j=pred[j];
            cout<<"\n\t <-"<<j;
        }while(j!=start);
        if(dist[i]<10){
            long int price;
            long int price2;
            price=1000*dist[i];
            price2=price+2000;
            cout<<"\n\t Twisted Pair Cable are required";
            cout<<"\n\t Price of Twisted Pair Cable
Rs"<<price;

```

```

        cout<<"\n\t Local Area Network are required";
        cout<<"\n\t Price of Local Area Network Rs2000";
        cout<<"\n\t Total Price : "<<price2;
    }
    else if(dist[i]>=10 && dist[i]<=50){
        long int price1;
        long int price3;
        price1=2000*dist[i];
        price3=price1+3000+1000;
        cout<<"\n\t Coaxial Cable are required";
        cout<<"\n\t Price of Coaxial Cable Rs"<<price1;
        cout<<"\n\t Metropolitan Area Network are
required";

        cout<<"\n\t Price of Metropolitan Area Network
Rs3000";

        cout<<"\n\t Repeater are required";
        cout<<"\n\t Price of Repeater Rs1000";
        cout<<"\n\t Total Price : "<<price3;
    }
    else if(dist[i]>50){
        long int price4;
        long int price5;
        price4=3000*dist[i];

```



```

        price5=price4+4000+1000;
        cout<<"\n\t Optical fiber are required ";
        cout<<"\n\t Price of optical fiber Rs"<<price4;
        cout<<"\n\t Wide Area Network are required";
        cout<<"\n\t Price of Wide Area Network Rs4000";
        cout<<"\n\t Reapeater are required";
        cout<<"\n\t Price of Reapeater Rs1000";
        cout<<"\n\t Total Price :"<<price5;

    }

}

cout<<"\n";

}

}

void graph(){
    system("cls");
    system("COLOR 9F");
    int G[MAX][MAX];
    int n;
    int i,j;
    int start;
    cout<<"\n\t Enter Number of Building :";
    cin>>n;

```

```

        cout<<"\n\t Enter Building Connection:";

        for(i=0;i<n;i++){
            for(j=0;j<n;j++){
                cin>>G[i][j];
            }
        }

        cout<<"\n\t Enter Whose Building Which Contain Router  :";

        cin>>start;

        dijk(G,n,start); //function call

        cout<<"\n\n";

        system("pause"); //just like getch()

        menu();

        //
    }

    void opening()

    {
        system("cls");

        system("COLOR 1F");

        int i=1,j=1;

        while((i<32)&&(j<12))

        {
            system("cls");

```

```

gotoxy(i,j);
cout<<" \n\t\t\t\t\t=====PROJECT REPORT===== ";
Sleep(20);
gotoxy((56+i),(j+2));
cout<<"\n\n\n\t\t\t\t\t*****LAN INFRASTRUCTURE
DESIGN*****";
Sleep(50);
cout<<"\n\nCOLLEGE : DELHI TECHNOLOGICAL
UNIVERSITY";
cout<<"\nSUBMITTED BY: AYUSH KARN";
cout<<"\nROLL NO : 2K19/CO/454";
cout<<"\nUnder the guidance of:- Mr. Ajay Kumar Sir \n";
i=i+3;
j=j+1;
}

cout<<"\n";
cout<<"\n";
system("pause");
}
void graphic()
{
system("cls");

```

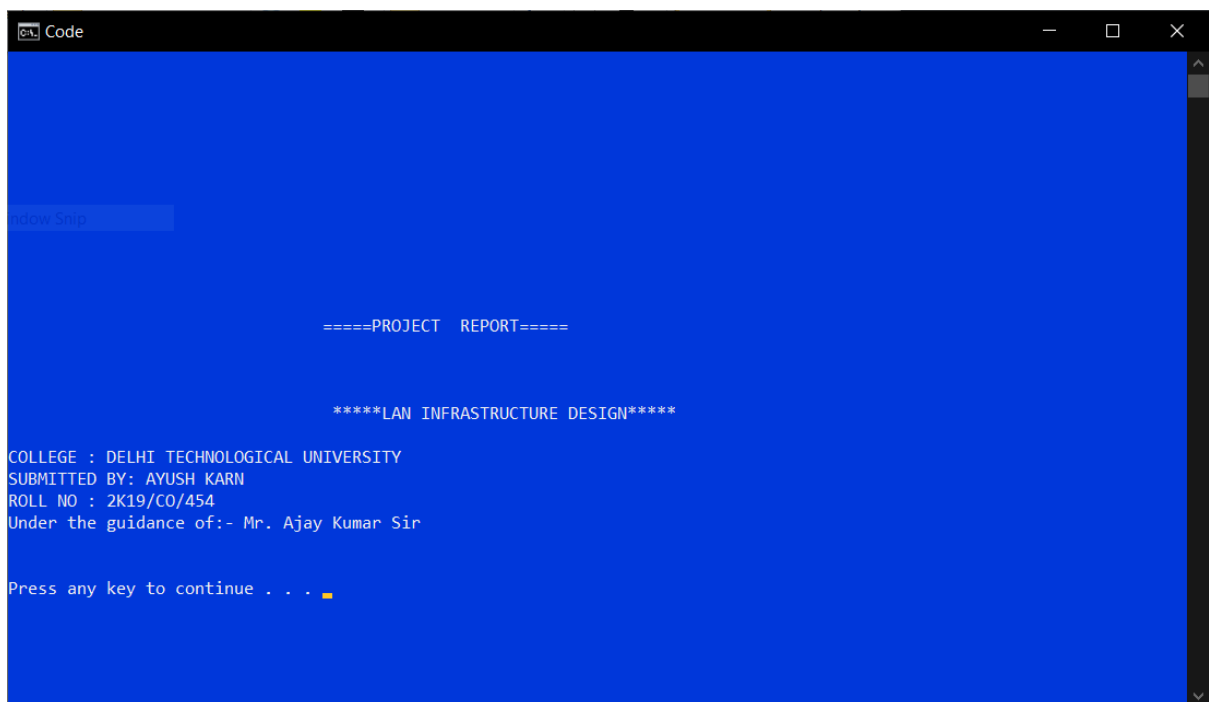
```
system("COLOR 3F");
gotoxy(20,14);
cout<<"\n\t\t ^^^^LAN INFRASTRUCTURE DESIGN^^^^ ";
gotoxy(40,20);
cout<<" ";
Sleep(1000);

gotoxy(22,20);
cout<<"\n\t\t\t Welcome to program developed by ";
gotoxy(28,25);
cout<<"\n\t\t\t Ayush Karn ";
gotoxy(28,27);
cout<<"\t\t\t LOADING.... ";
for(int p=0;p<=9;p++)
{
Sleep(200);
gotoxy(36+p,27);
cout<<".";
}
Sleep(1000);
system("cls");

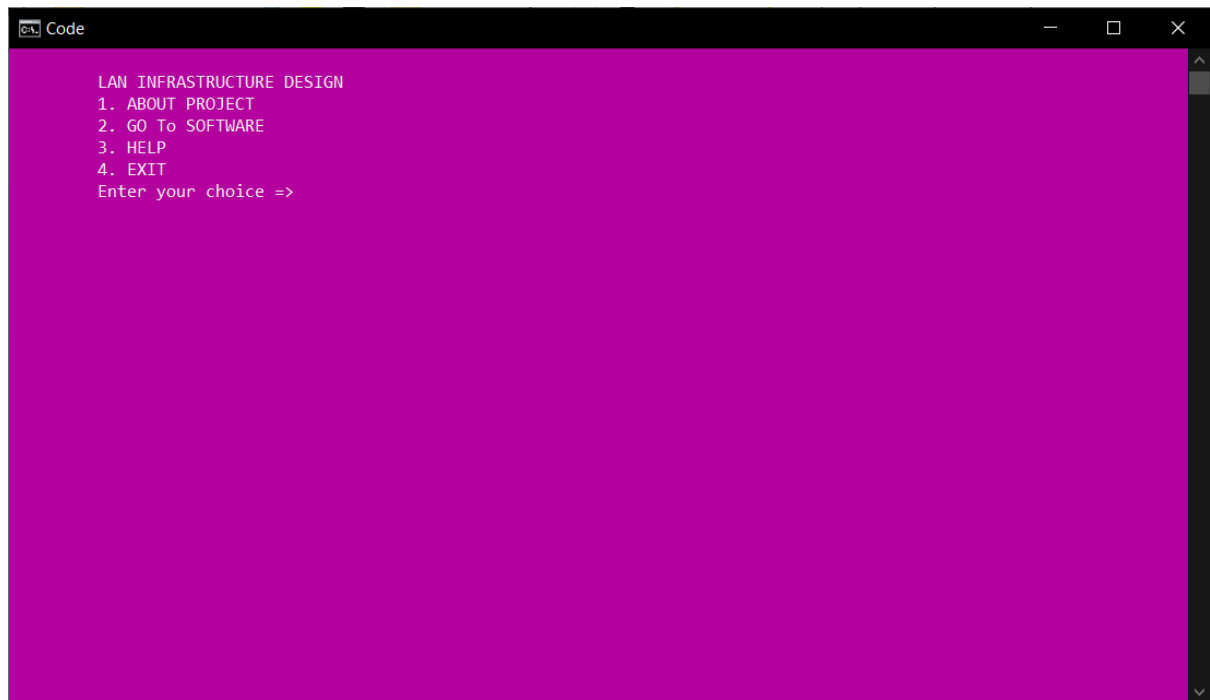
}
```

```
int main(){  
  
    system("cls");  
    opening();  
    system("cls");  
    graphic();  
    system("cls");  
    menu();  
    return 0;  
}
```

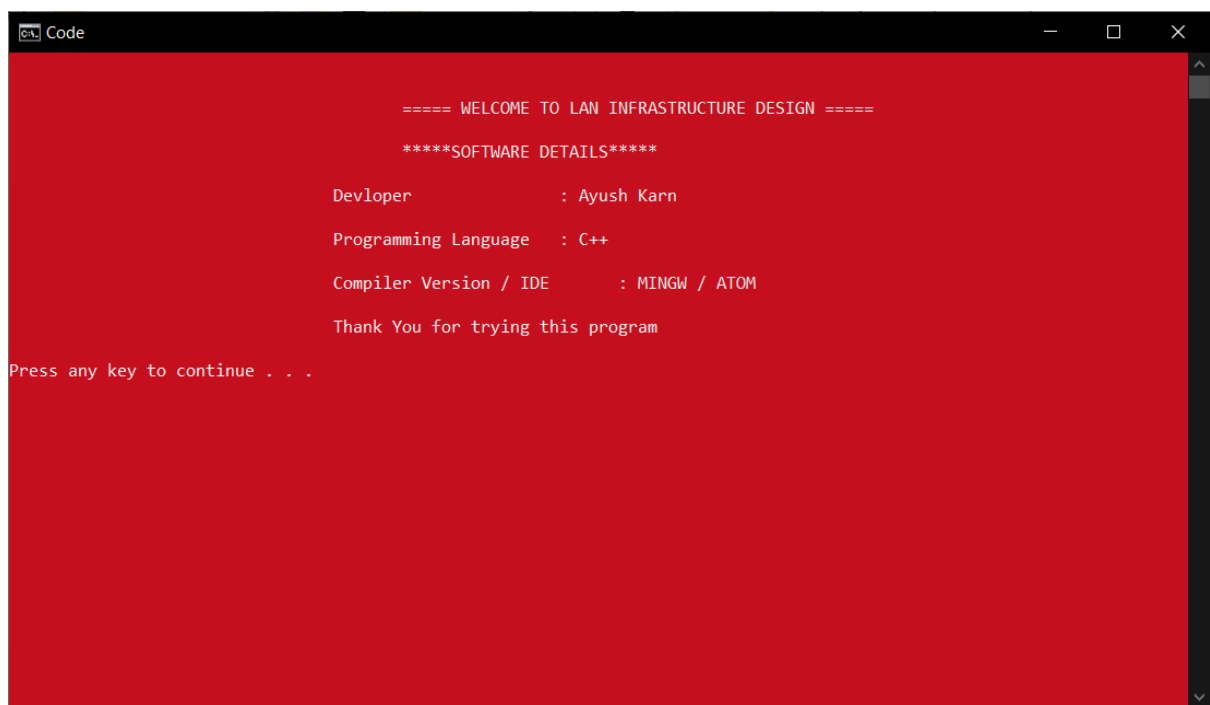
## Output:



```
Code  
Window Snip  
  
====PROJECT REPORT====  
  
*****LAN INFRASTRUCTURE DESIGN*****  
  
COLLEGE : DELHI TECHNOLOGICAL UNIVERSITY  
SUBMITTED BY: AYUSH KARN  
ROLL NO : 2K19/CO/454  
Under the guidance of:- Mr. Ajay Kumar Sir  
  
Press any key to continue . . .
```



```
LAN INFRASTRUCTURE DESIGN
1. ABOUT PROJECT
2. GO To SOFTWARE
3. HELP
4. EXIT
Enter your choice =>
```



```
===== WELCOME TO LAN INFRASTRUCTURE DESIGN =====

*****SOFTWARE DETAILS*****

Devloper           : Ayush Karn
Programming Language : C++
Compiler Version / IDE : MINGW / ATOM

Thank You for trying this program

Press any key to continue . . .
```

```
Code

The distance of Building from 0 to 1 is: 10
The path is :1
<-0
Coaxial Cable are required
Price of Coaxial Cable Rs20000
Metropolitan Area Network are required
Price of Metropolitan Area Network Rs3000
Repeater are required
Price of Repeater Rs1000
Total Price :24000

The distance of Building from 0 to 2 is: 20
The path is :2
<-0
Coaxial Cable are required
Price of Coaxial Cable Rs40000
Metropolitan Area Network are required
Price of Metropolitan Area Network Rs3000
Repeater are required
Price of Repeater Rs1000
Total Price :44000

The distance of Building from 0 to 3 is: 30
The path is :3
<-2
<-0
Coaxial Cable are required
Price of Coaxial Cable Rs60000
Metropolitan Area Network are required
Price of Metropolitan Area Network Rs3000
Repeater are required
Price of Repeater Rs1000
Total Price :64000

Press any key to continue . . .
```

```
Code

===== HELP =====
WELCOME TO LAN INFRASTRUCTURE DESIGN
In this Project you can find shortest distance b/w two building
-In first option you can see the information about us
-In second option you can operate the software.
-In third option you can view Help for the project
- In Fourth option you can EXIT. Press any key to continue

Press any key to continue . . .
```

```
LAN INFRASTRUCTURE DESIGN
-----

      COPYRIGHT @ 2020
      -----

      T H A N K S   F O R   U S I N G

      M Y   P R O G R A M
      =====
      B Y   A Y U S H   K A R N

Press any key to continue . . .
```



## Chapter 4 Conclusion and References

From this project I would like to conclude that I have actually learnt a great deal about Discrete Structure and especially Graph Traversal, Shortest Path finding techniques, Dijkstra Algorithm and many other related topics.

Credits:

- 1) <https://www.geeksforgeeks.org> (Geeks for Geeks)
- 2) <https://www.wikipedia.org> (Wikipedia)
- Discrete Mathematics by Rosen.