ASSIGNMENT 3

1. **Aim** –There are flight paths between cities.if there is a flight between city a and B then there is an edge between cities.the cost of the edge can be the time that flight take to reach city B or the amount of fuel used.Represent this in a graph.the node can be represented by airport name or city.Use adjacency list or matrix.

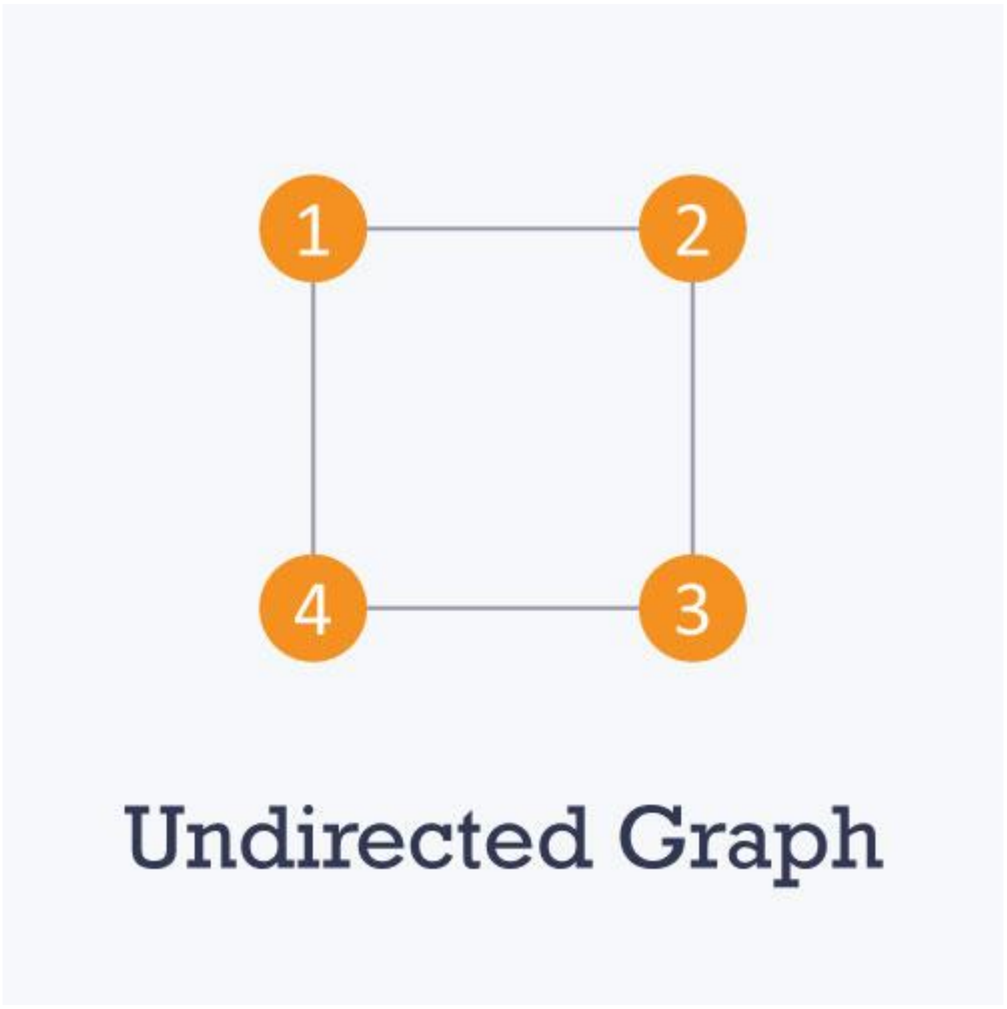
2. **Objective** – To represent the graph using either adjacency list or by matrix.

3. **Theory –**

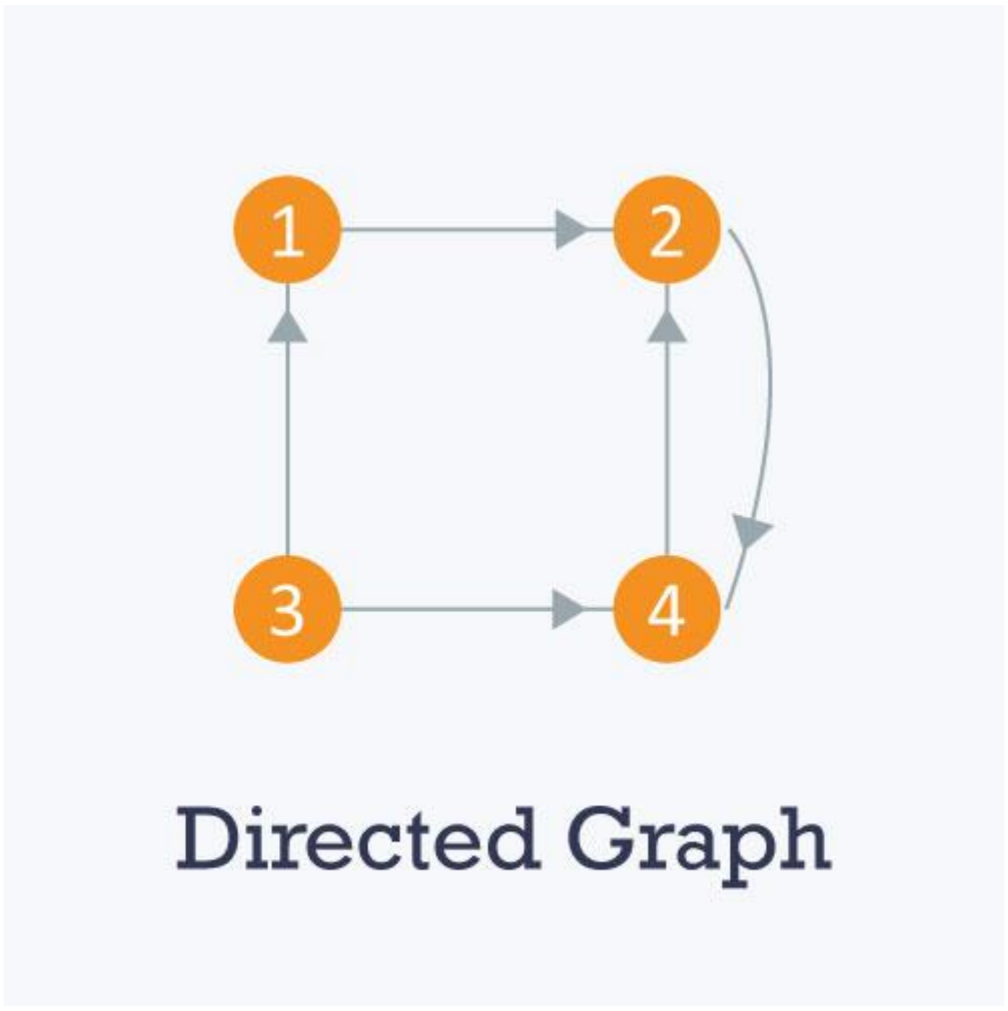
Graph is a data structure that consists of following two components:
1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Types of graphs

- Undirected: An undirected graph is a graph in which all the edges are bi-directional i.e. the edges do not point in any specific direction.

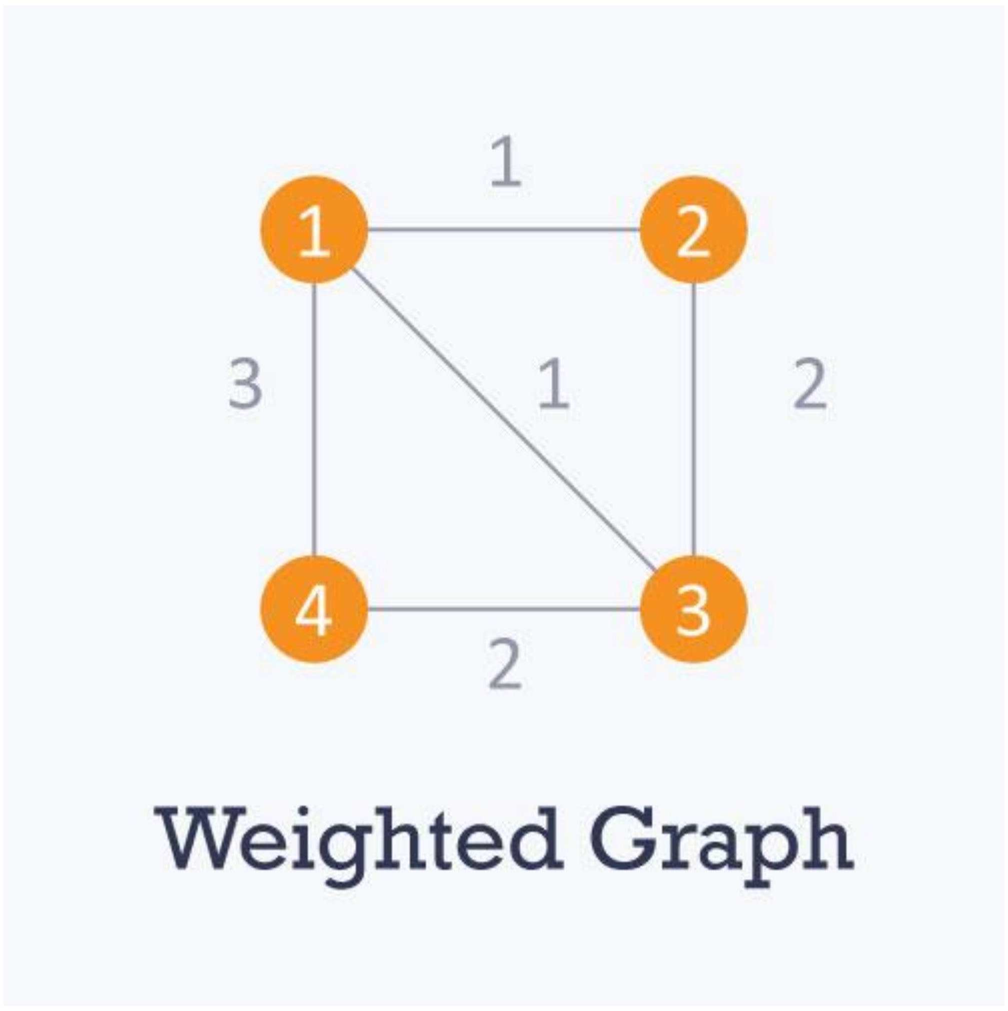


- Directed: A directed graph is a graph in which all the edges are uni-directional i.e. the edges point in a single direction.



- Weighted: In a weighted graph, each edge is assigned a weight or cost. Consider a graph of 4 nodes as in the diagram below. As you can see each edge has a weight/cost assigned to it. If you want to go from vertex 1 to vertex 3, you can take one of the following 3 paths:
  - 1 -> 2 -> 3
  - 1 -> 3
  - 1 -> 4 -> 3

  Therefore the total cost of each path will be as follows: - The total cost of 1 -> 2 -> 3 will be (1 + 2) i.e. 3 units - The total cost of 1 -> 3 will be 1 unit - The total cost of 1 -> 4 -> 3 will be (3 + 2) i.e. 5 units

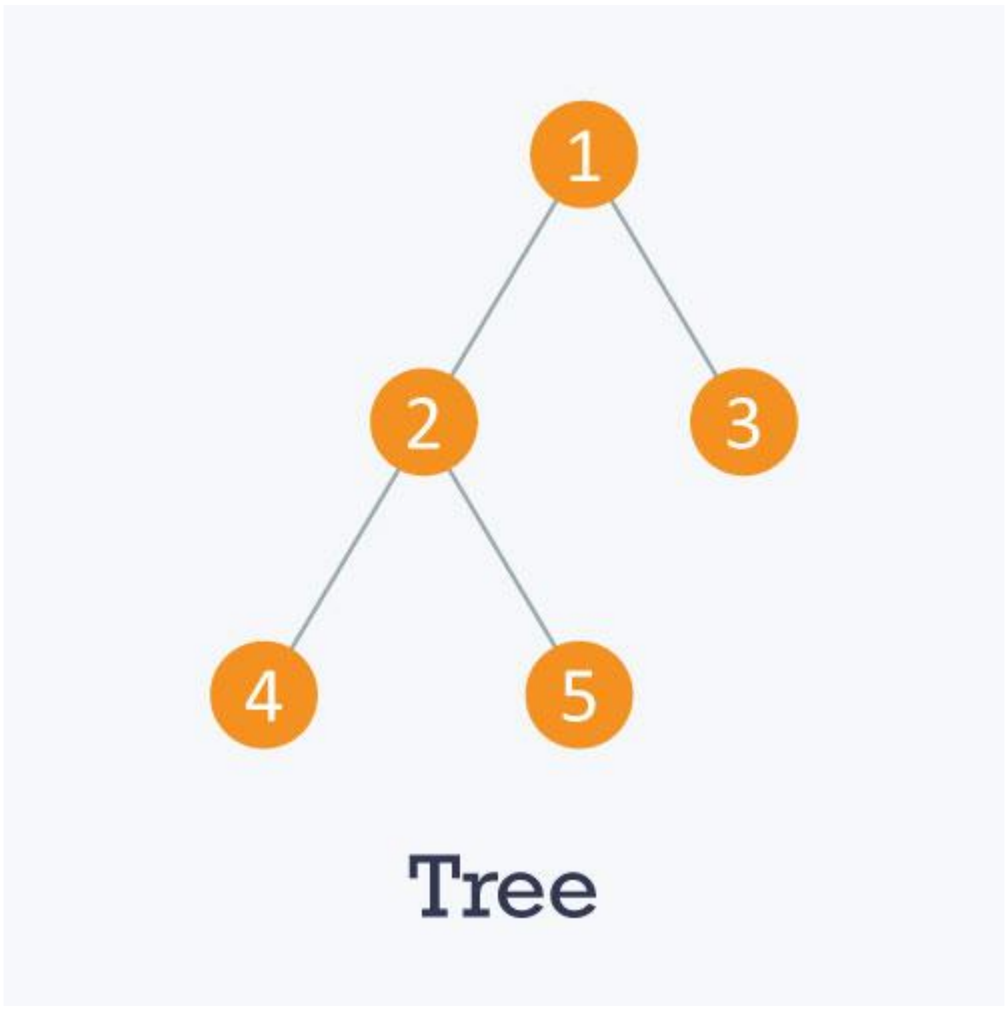


- Cyclic: A graph is cyclic if the graph comprises a path that starts from a vertex and ends at the same vertex. That path is called a cycle. An acyclic graph is a graph that has no cycle.

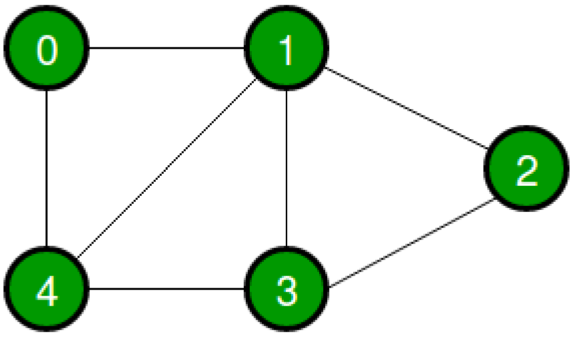
  A **tree** is an undirected graph in which any two vertices are connected by only one path. A tree is an acyclic graph and has N - 1 edges where N is the number of vertices. Each node in a graph may have one or multiple parent nodes. However, in a tree, each node (except the root node) comprises exactly one parent node.

  Note: A root node has no parent.

  A tree cannot contain any cycles or self loops, however, the same does not apply to graphs.



Following is an example of an undirected graph with 5 vertices.



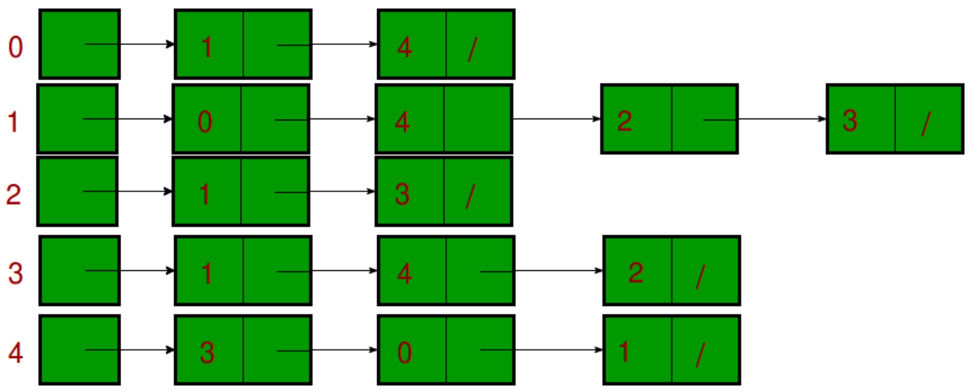
Following two are the most commonly used representations of a graph.
1. Adjacency Matrix
2. Adjacency List
There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0
\end{array}
$$

Adjacency List:
An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is adjacency list representation of the above graph.



4. **Algorithm –**

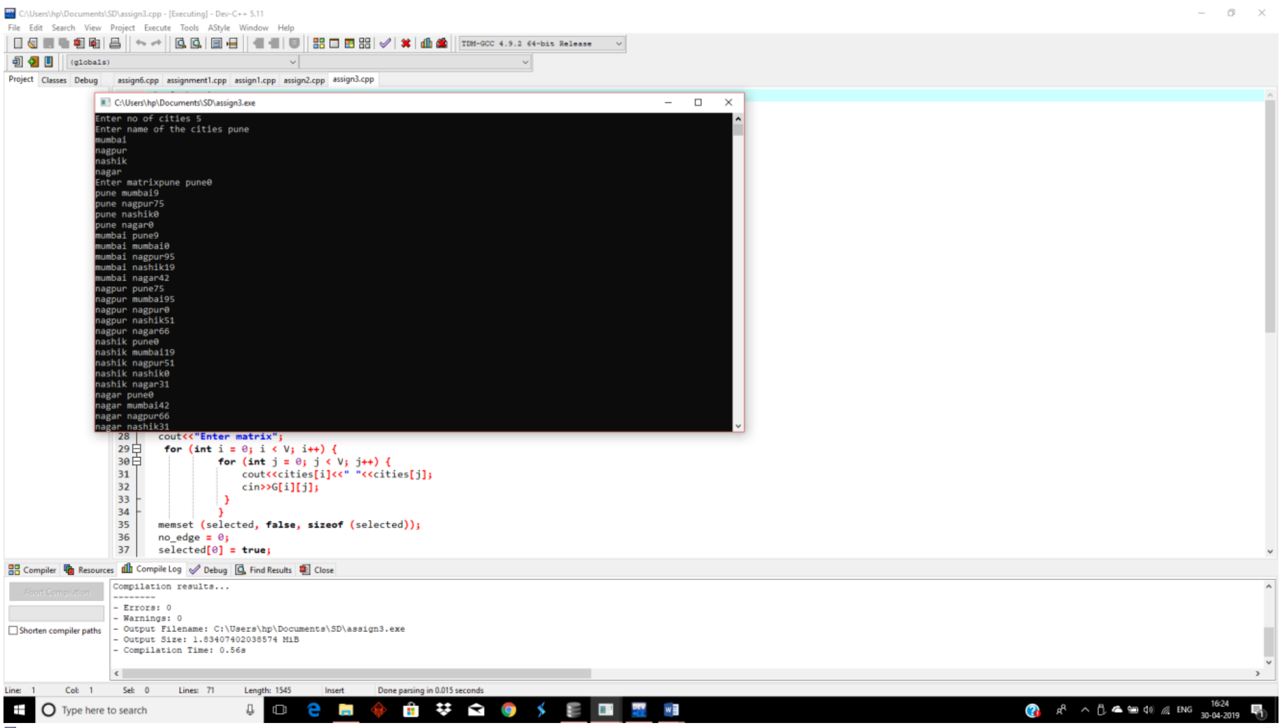
```
1. void addEdge(vector<int> adj[], int u, int v)
2. {
3. adj[u].push_back(v);
4. adj[v].push_back(u);
5. }
6. void printGraph(vector<int> adj[], int V)
7. {
8. for (int v = 0; v < V; ++v)
9. {
        i. cout << "\n Adjacency list of vertex "
            1. << v << "\n head ";
       ii. for (auto x : adj[v])
      iii. cout << "-> " << x;
       iv. printf("\n");
10.    }
11.    }
```

5. **Program code –**

```
#include <iostream>

#include <cstring>

using namespace std;


#define INF 9999999


//#define V 5


//int G[V][V] = {

//  {0, 9, 75, 0, 0},
// {9, 0, 95, 19, 42},
// {75, 95, 0, 51, 66},
// {0, 19, 51, 0, 31},
// {0, 42, 66, 31, 0}
//};

int main () {
  int no_edge,sum=0,V;
  cout<<"Enter no of cities";
  cin>>V;
  string cities[V];
  cout<<"Enter name of the cities";
  for (int i = 0; i < V; i++) {
  cin>>cities[i];
  }
  int G[V][V];
  int selected[V];
  cout<<"Enter matrix";
   for (int i = 0; i < V; i++) {
          for (int j = 0; j < V; j++) {
                  cout<<cities[i]<<" "<<cities[j];
                  cin>>G[i][j];
           }
                          }
  memset (selected, false, sizeof (selected));
  no_edge = 0;
  selected[0] = true;


  int x;
  int y;
  cout << "Edge" << " : " << "Weight";
  cout << endl;
  while (no_edge < V - 1) {


    int min = INF;
    x = 0;
    y = 0;


    for (int i = 0; i < V; i++) {
     if (selected[i]) {
        for (int j = 0; j < V; j++) {
         if (!selected[j] && G[i][j]) {
            if (min > G[i][j]) {
               min = G[i][j];
               x = i;
               y = j;
            }


         }
       }
     }
    }
```

```
      cout << x <<  " - " << y << " :  " << G[x][y];
      sum=sum+G[x][y];
      cout << endl;
      selected[y] = true;
      no_edge++;
    }
 cout<<"cost = "<<sum;
  return 0;
}
```

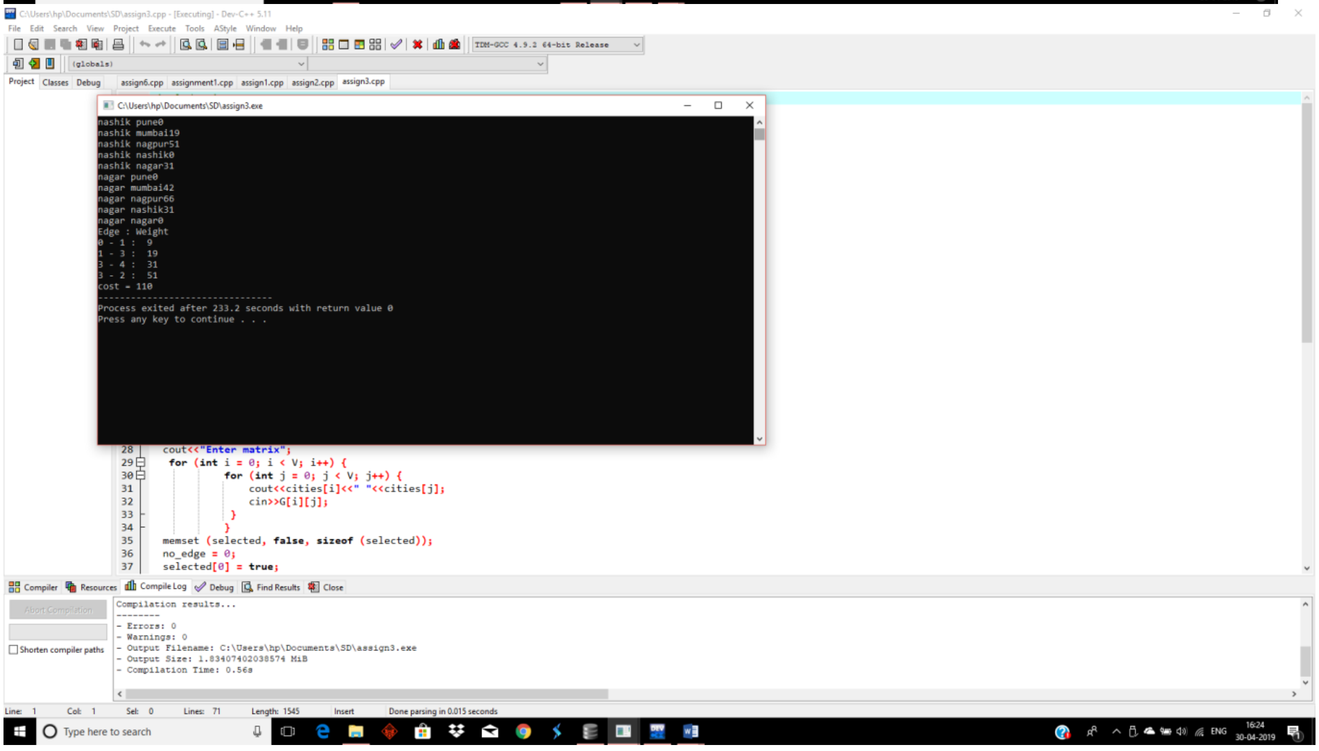
6. **Output screen shots/Copy Past From Terminal –**



```
Enter no of cities 5
Enter name of the cities pune
mumbai
nagpur
nashik
nagar
Enter matrixpune pune0
pune mumbai9
pune nagpur75
pune nashik0
pune nagar0
mumbai pune9
mumbai mumbai0
mumbai nagpur95
mumbai nashik19
mumbai nagar42
nagpur pune75
nagpur mumbai95
nagpur nagpur0
nagpur nashik51
nagpur nagar66
nashik pune0
nashik mumbai19
nashik nagpur51
nashik nashik0
nashik nagar31
nagar pune0
nagar mumbai42
nagar nagpur66
nagar nashik31
```

```
 cout<<"Enter matrix";
  for (int i = 0; i < V; i++) {
          for (int j = 0; j < V; j++) {
              cout<<cities[i]<<" "<<cities[j];
              cin>>G[i][j];
          }
         }
 memset (selected, false, sizeof (selected));
 no_edge = 0;
 selected[0] = true;
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\hp\Documents\SD\assign3.exe
- Output Size: 1.83407402038574 MiB
- Compilation Time: 0.56s
```



```
nashik pune0
nashik mumbai19
nashik nagpur51
nashik nashik0
nashik nagar31
nagar pune0
nagar mumbai42
nagar nagpur66
nagar nashik31
nagar nagar0
Edge : Weight
0 - 1 :  9
1 - 3 :  19
3 - 4 :  31
3 - 2 :  51
cost = 110
--------------------------------
Process exited after 233.2 seconds with return value 0
Press any key to continue . . .
```

```
 cout<<"Enter matrix";
  for (int i = 0; i < V; i++) {
          for (int j = 0; j < V; j++) {
              cout<<cities[i]<<" "<<cities[j];
              cin>>G[i][j];
          }
         }
 memset (selected, false, sizeof (selected));
 no_edge = 0;
 selected[0] = true;
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\hp\Documents\SD\assign3.exe
- Output Size: 1.83407402038574 MiB
- Compilation Time: 0.56s
```

7. **Conclusion** –

Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.