**Problem No: 01**

**Topic: Graph**

**Problem Title:**

Represent a undirected weighted graph using adjacency list representation.

**Objectives:**

To represent adjacency list of a weighted graph.

**Source Code:**

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

typedef pair<int, int> P;

int main()

{

int vertex, edge;

ifstream fin;

fin.open("weighted graph.txt");

fin >> vertex >> edge;

vector <P> v[vertex];

cout << vertex << " " << edge << endl;

for(int i = 0; i < edge; i++){

int source;

P p;

fin >> source >> p.first >> p.second;

v[source].push\_back(p);

int s2;

s2 = p.first;

p.first = source;

v[s2].push\_back(p);

}

for(int i = 0; i < vertex; i++){

cout << i << " - ";

for(int j = 0; j < v[i].size(); j++){

cout << v[i][j].first << " (" << v[i][j].second << ") - ";

}

cout << endl;

}

return 0;

}

**Input (file I/O):**

5 5

0 1 4

0 3 6

0 4 2

1 2 3

2 4 5

**Output:**



**Problem No: 02**

**Topic: Graph**

**Problem Title:**

Represent a undirected weighted graph using adjacency matrix representation.

**Objectives:**

To construct an adjacency matrix of a undirected weighted graph.

**Source Code:**

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

void addEdge(vector<int> graph[], int u, int v);

int main()

{

ifstream fin;

fin.open("weighted graph.txt");

int n, e;

fin >> n;

fin >> e;

int m[n][n] = {};

vector <int> graph[n];

int u, v, w;

for(int i = 0; i < e; i++){

fin >> u;

fin >> v;

fin >> w;

addEdge(graph, u, v);

m[u][v] = w;

m[v][u] = w;

}

cout << "Adjacency Matrix for Undirected weighted Graph: \n";

for(int i = 0; i < n; i++){

for(int j = 0; j < n; j++)

cout << m[i][j] << " ";

cout << endl;

}

return 0;

}

void addEdge(vector<int> graph[], int u, int v)

{

graph[u].push\_back(v);

graph[v].push\_back(u);

}

**Input (file I/O):**

5 5

0 1 4

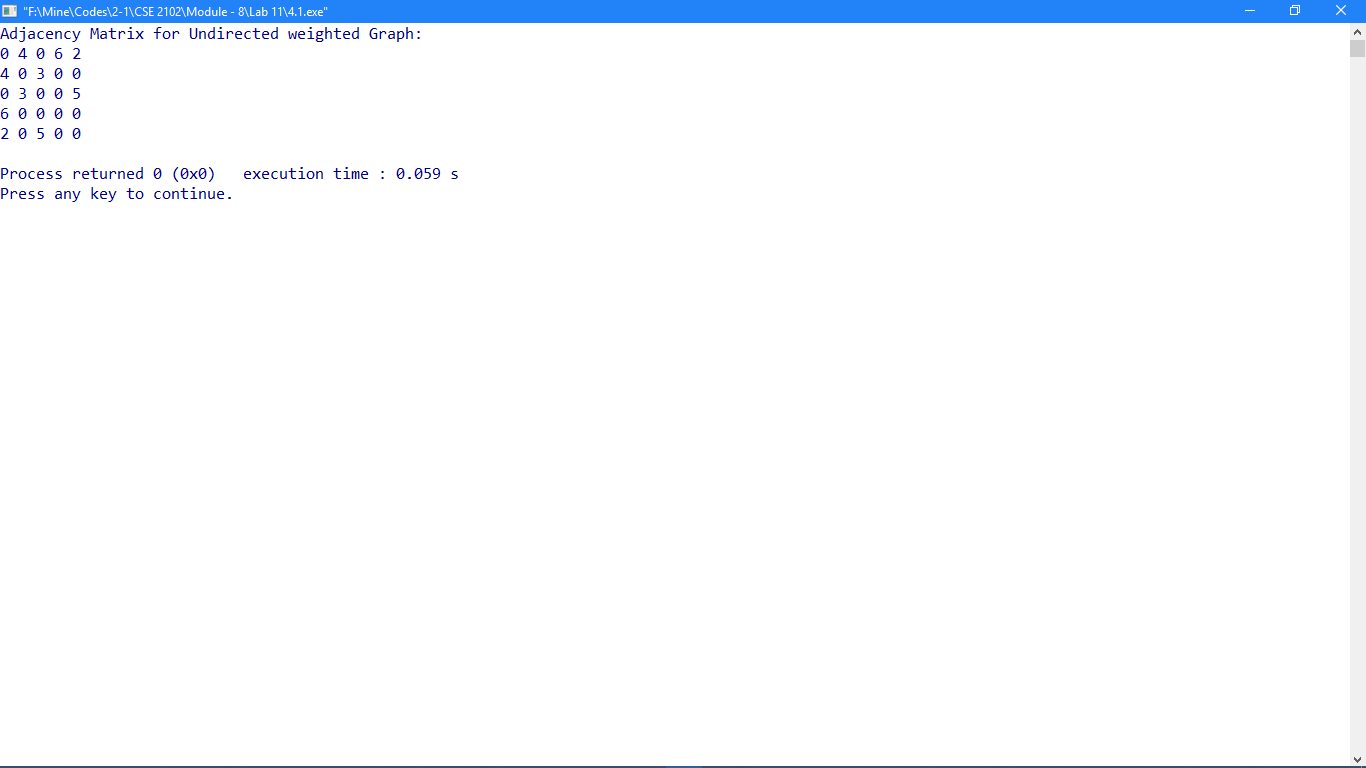
0 3 6

0 4 2

1 2 3

2 4 5

**Output:**



**Problem No: 03**

**Topic: Graph**

**Problem Title:**

Traverse a graph using BFS traversal.

**Objectives:**

To implement BFS traversal.

**Source Code:**

#include <iostream>

#include <list>

#include <fstream>

using namespace std;

class Graph{

int V;

list<int> \*adj;

public:

Graph(int V);

void addEdge(int v, int w);

void BFS(int s);

};

Graph::Graph(int V){

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w){

adj[v].push\_back(w);

adj[w].push\_back(v);

}

void Graph::BFS(int s){

bool \*visited = new bool[V];

for(int i = 0; i < V; i++)

visited[i] = false;

list<int> queue;

visited[s] = true;

queue.push\_back(s);

list<int>::iterator i;

while(!queue.empty()){

s = queue.front();

cout << s << " ";

queue.pop\_front();

for (i = adj[s].begin(); i != adj[s].end(); ++i){

if (!visited[\*i]){

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

int main()

{

int vertex, edge;

ifstream fin;

fin.open("graph.txt");

fin >> vertex >> edge;

Graph g(vertex);

for(int i = 0; i < edge; i++){

int startNode, endNode;

fin >> startNode >> endNode;

g.addEdge(startNode, endNode);

}

cout << "Breadth First Traversal "

<< "(starting from vertex 0) \n";

g.BFS(0);

return 0;

}

**Input (file I/O):**

5 5

0 1

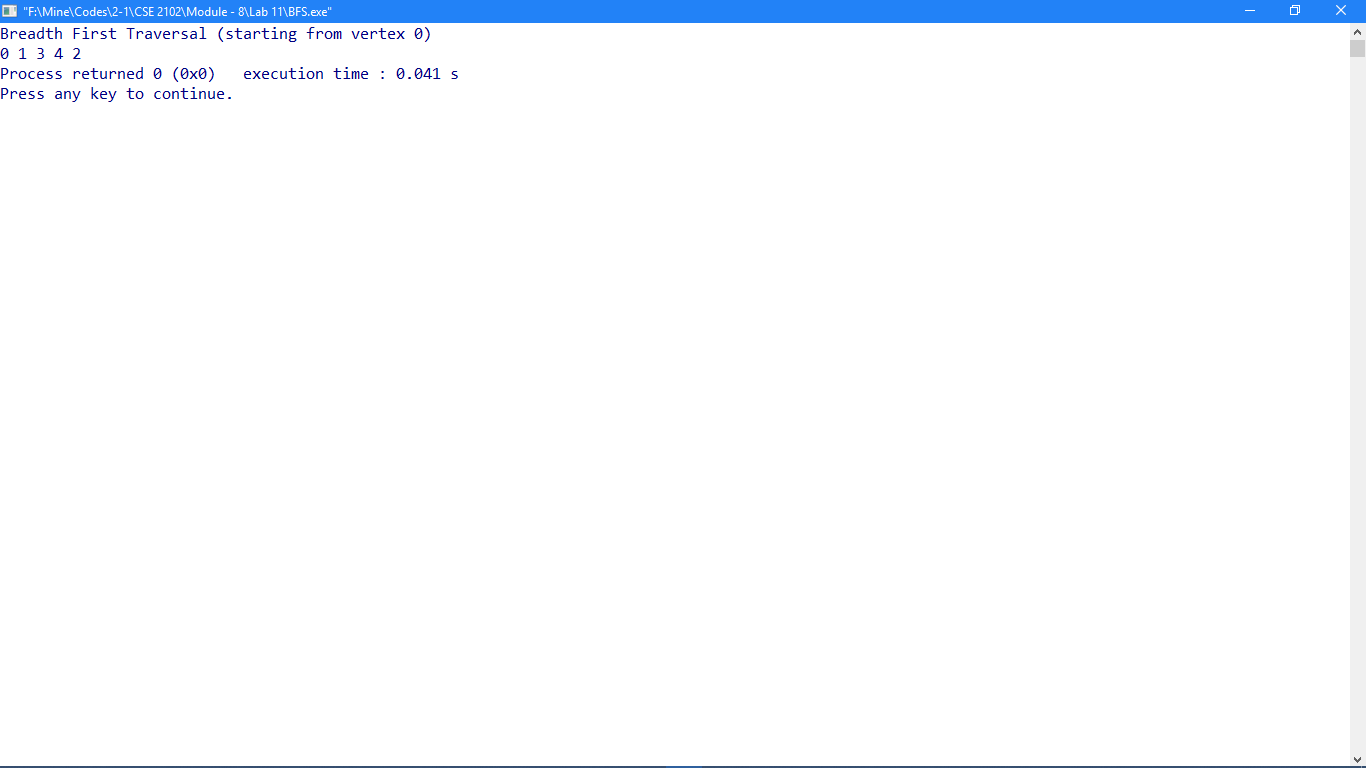
0 3

0 4

1 2

2 4

**Output:**



**Problem No: 04**

**Topic: Graph**

**Problem Title:**

Traverse a graph using DFS traversal.

**Objectives:**

To implement DFS traversal.

**Source Code:**

#include<iostream>

#include<list>

#include <fstream>

using namespace std;

class Graph{

int V;

list<int> \*adj;

void DFSUtil(int v, bool visited[]);

public:

Graph(int V);

void addEdge(int v, int w);

void DFS(int v);

};

Graph::Graph(int V){

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w){

adj[v].push\_back(w);

adj[w].push\_back(v);

}

void Graph::DFSUtil(int v, bool visited[]){

visited[v] = true;

cout << v << " ";

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFSUtil(\*i, visited);

}

void Graph::DFS(int v){

bool \*visited = new bool[V];

for (int i = 0; i < V; i++)

visited[i] = false;

DFSUtil(v, visited);

}

int main()

{

int vertex, edge;

ifstream fin;

fin.open("graph.txt");

fin >> vertex >> edge;

Graph g(vertex);

for(int i = 0; i < edge; i++){

int startNode, endNode;

fin >> startNode >> endNode;

g.addEdge(startNode, endNode);

}

cout << "Depth First Traversal "

<< "(starting from vertex 0) \n";

g.DFS(0);

return 0;

}

**Input (file I/O):**

5 5

0 1

0 3

0 4

1 2

2 4

**Output:**

