Experiment no. 9

Aim : Implement graph using adjacency list or matrix and perform DFS or BFS.

Theory:

Algorithms:

Creation of Adjacency list:

1. Declare an array of pointers to a link list having a data field (to store vertex no) and a forward pointer. The no of array of pointer would equal the total no of vertices in the graph.
2. Take the edge set from the user. If for eg. vertex 1 is connected to vertex 2 & 3 in the graph, the 1st location of the array of pointers would point to 2 nodes, one having the data 2 and the other having data 3.
3. In this way construct the entire adjacency list.

DFS (Depth First Search).

1. The start vertex is visited. Next an ~~uni~~ unvisted vertex w adjacent to v is selected & a DFS from w initiated.
2. When a vertex u is reached such that all it adjacant vertices have been visted, we back up to the last vertex visited which has an unvisited vertex w adjacent to it and initiate a DFS search from w.
3. The search terminal when no unvisted vertex can be reached from any of the visted ones.

BFS (~~B~~Breadth First Search).

1. Starting at vertex v and marking it as visited, BFS differs from DFS in that all unvisted vertices adjacent to v are visited next.
2. Then unvisited vertices adjacent to these vertices are visited & so on.
3. A queue is used to store vertices as they are visited so that later search can be initiated from those vertices.

Test Conditions:

Enter the graph with 8 vertices & 10 edges (1, 2), (1, 3), (2, 4), (2, 5), (3, 6), (3, 7), (4, 8), (5, 8), (6, 8), (7, 8).

The order of the vertices visited by DFS is : 1, 2, 4, 8, 5, 6, 3, 7

The order of the vertices visited by BFS is 1, 2, 3, 4, 5, 6, 7, 8

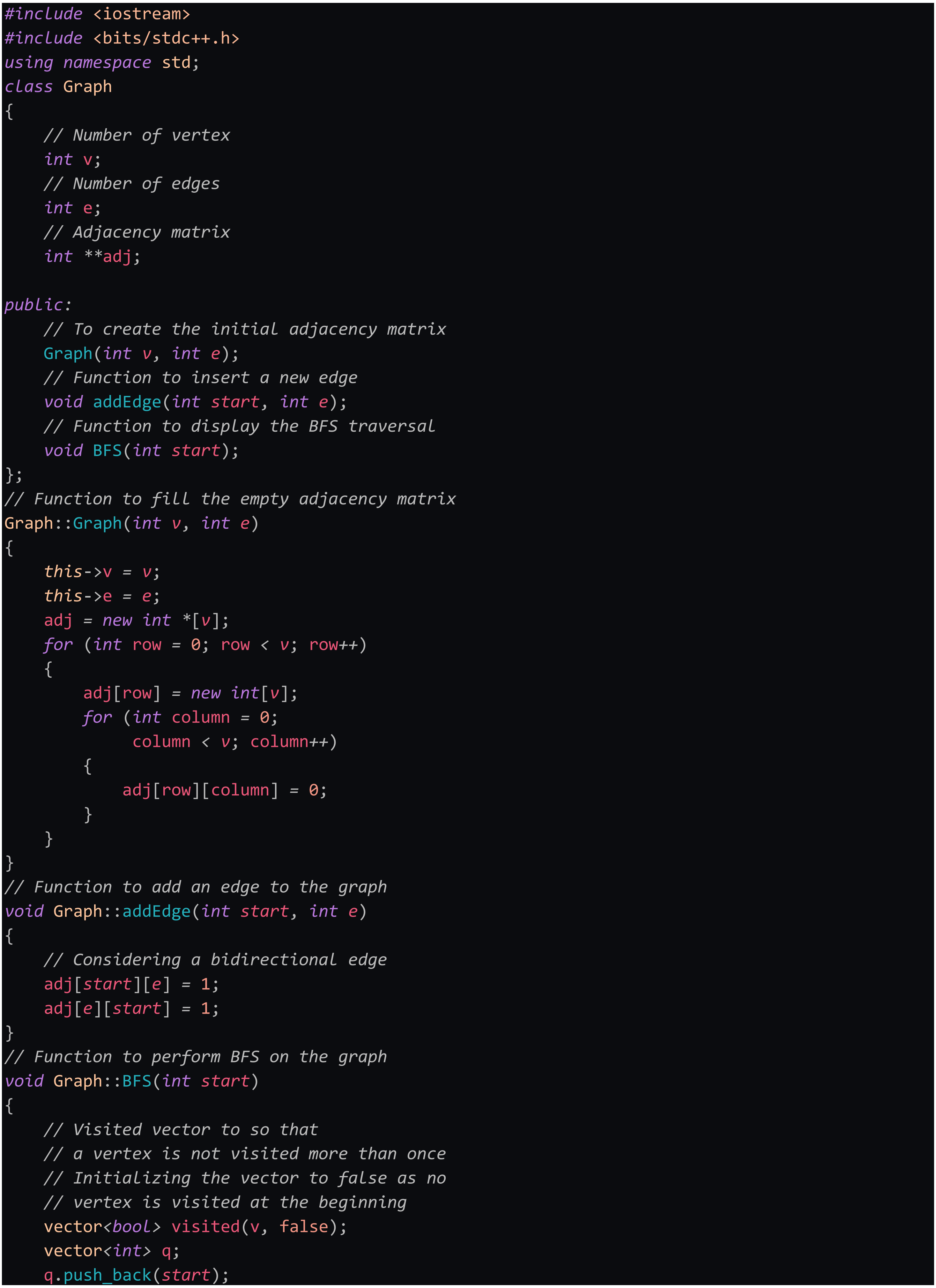
Input :

The no. of vertices & the edge set of the graph.

Output:

The order of vertices visited in both DFS & BFS.

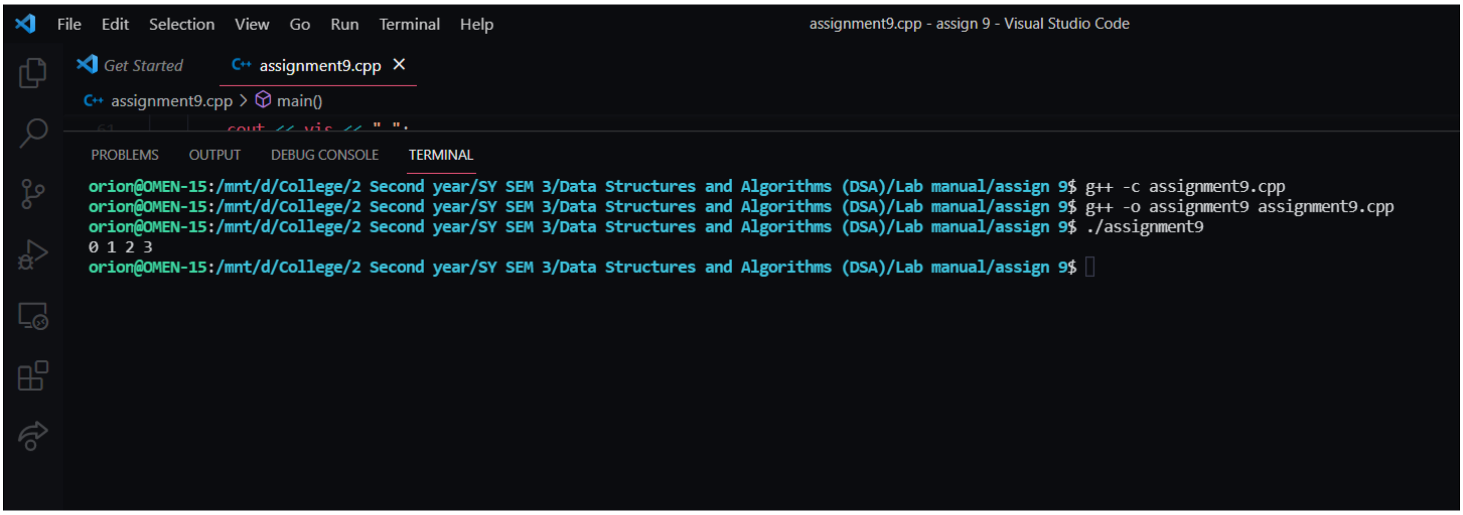
Program –



```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
class Graph
{
    // Number of vertex
    int v;
    // Number of edges
    int e;
    // Adjacency matrix
    int **adj;

public:
    // To create the initial adjacency matrix
    Graph(int v, int e);
    // Function to insert a new edge
    void addEdge(int start, int e);
    // Function to display the BFS traversal
    void BFS(int start);
};
// Function to fill the empty adjacency matrix
Graph::Graph(int v, int e)
{
    this->v = v;
    this->e = e;
    adj = new int *[v];
    for (int row = 0; row < v; row++)
    {
        adj[row] = new int[v];
        for (int column = 0;
             column < v; column++)
        {
            adj[row][column] = 0;
        }
    }
}
// Function to add an edge to the graph
void Graph::addEdge(int start, int e)
{
    // Considering a bidirectional edge
    adj[start][e] = 1;
    adj[e][start] = 1;
}
// Function to perform BFS on the graph
void Graph::BFS(int start)
{
    // Visited vector to so that
    // a vertex is not visited more than once
    // Initializing the vector to false as no
    // vertex is visited at the beginning
    vector<bool> visited(v, false);
    vector<int> q;
    q.push_back(start);
    // Set source as visited
    visited[start] = true;
    int vis;
    while (!q.empty())
    {
        vis = q[0];
        // Print the current node
        cout << vis << " ";
        q.erase(q.begin());
        // For every adjacent vertex to the current vertex
        for (int i = 0; i < v; i++)
        {
            if (adj[vis][i] == 1 && (!visited[i]))
            {
                // Push the adjacent node to the queue
                q.push_back(i);
                // Set
                visited[i] = true;
            }
        }
    }
}
// Driver code
int main()
{
    int v = 5, e = 4;
    // Create the graph
    Graph G(v, e);
    G.addEdge(0, 1);
    G.addEdge(0, 2);
    G.addEdge(1, 3);
    G.BFS(0);
}
```

Output-



```
orion@OMEN-15:/mnt/d/College/2 Second year/SY SEM 3/Data Structures and Algorithms (DSA)/Lab manual/assign 9$ g++ -c assignment9.cpp
orion@OMEN-15:/mnt/d/College/2 Second year/SY SEM 3/Data Structures and Algorithms (DSA)/Lab manual/assign 9$ g++ -o assignment9 assignment9.cpp
orion@OMEN-15:/mnt/d/College/2 Second year/SY SEM 3/Data Structures and Algorithms (DSA)/Lab manual/assign 9$ ./assignment9
0 1 2 3
orion@OMEN-15:/mnt/d/College/2 Second year/SY SEM 3/Data Structures and Algorithms (DSA)/Lab manual/assign 9$
```