Graphs

```
#ifndef GRAPH_H
#define GRAPH_H
#include<iostream>
#include<queue>
#include<stack>
using namespace std;
class Graph
{
public:
	//part1: constructor initializes adjacency matrix
	Graph(int numVertex);
	//part2: returns the number of vertices in the graph
	int GetNumVertices();
	//part3: returns the number of edges in the graph
	int numberOfEdges();
	//part4: inserts edge going from one vertex to another
	void insertEdge(int frmVertex, int toVertex);
	//part5: removes edge going from one vertex to another
	void removeEdge(int frmVertex, int toVertex);
	//part6: returns the degree of the node passed
	int degree(int vertex);

	//part7: outputs the order in which vertices are visited during DFS
	//Starting from node s.
	void depthfirstSearch(int s);

	//part8: outputs the order in which vertices are visited during BFS
		//Starting from node s.
	void breadthfirstSearch(int s);
	void display();
private:
	int** adj_matrix;
	int numVertices;
};
#endif
```

```
#include"Graph.h"
Graph::Graph(int NumVertex)
{
	numVertices = NumVertex;
	adj_matrix = new int* [numVertices];
	for (int i = 0; i < numVertices; i++)
	{
		adj_matrix[i] = new int[numVertices];
		for (int j = 0; j < numVertices; j++)
		{
			adj_matrix[i][j] = 0;
		}
	}
}
```

```
void Graph::insertEdge(int FromVertex, int ToVertex)
{
        if (FromVertex >= numVertices || ToVertex >= numVertices || FromVertex < 0 || ToVertex < 0
)
        {
                cout << "Invalid edge!\n";
        }
        else
        {
                adj_matrix[FromVertex][ToVertex] = 1;
        }
}
```

```
int Graph::degree(int vertex)
{
        int degr = 0;
        for (int i = 0; i < numVertices; ++i)
        {
                if (adj_matrix[vertex][i] == 1)
                {
                        degr++;
                }
        }
        return degr;
}
```

```
int Graph::numberOfEdges()
{
        int edges = 0;
        for (int i = 0; i < numVertices; ++i)
        {
                for (int j = 0; j < numVertices; ++j)
                {
                        if (adj_matrix[i][j] == 1)
                        {
                                edges++;
                        }
                }
        }
        return edges / 2;
}
```

```
void Graph::depthfirstSearch(int s)
  {
          bool* visited = new bool[numVertices];
          for (int i = 0; i < numVertices; i++)
          {
                    visited[i] = false;
          }
          stack<int>* qu = new stack <int>();
          qu->push(s);
          while (!qu->empty())
          {
                    cout << qu->top() << "\t";
                    qu->pop();
                    visited[s] = true;
                    for (int i = 0; i < numVertices; i++)
                    {
                              if (adj_matrix[s][i] != 0 && !visited[i])
                              {
                                        s = i;
                                        qu->push(i);
                                        break;
                              }
                    }
          }
          cout << "\n";
  }


  void Graph::breadthfirstSearch(int s)
  {
          bool* visited = new bool[numVertices];
          for (int i = 0; i < numVertices; i++)
          {
                    visited[i] = false;
          }
          queue<int>* qu = new queue<int>();
          qu->push(s);
          while (!qu->empty())
          {
                    s = qu->front();
                    cout << qu->front() << "\t";
                    qu->pop();
                    visited[s] = true;
                    for (int i = 0; i < numVertices; i++)
                    {
                              if (adj_matrix[s][i] != 0 && !visited[i])
                              {
                                        qu->push(i);
                                        visited[i] = true;
                              }
                    }
          }
          cout << "\n";
  }


  void Graph::display()
  {
          int i, j;
          for (i = 0; i < numVertices; i++)
          {
```

```
                for (j = 0; j < numVertices; j++)
                {
                        cout << adj_matrix[i][j] << "  ";
                }
                cout << endl;
        }
}
```

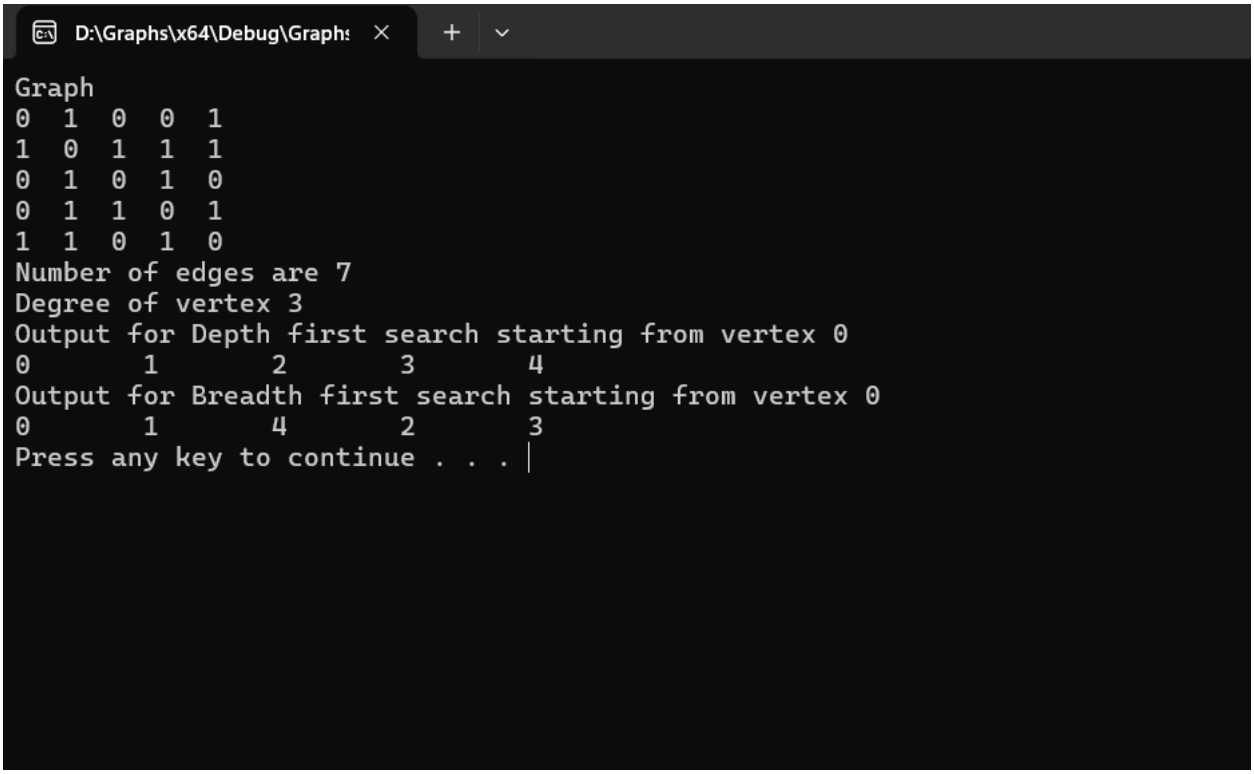
Driver

```
#include<iostream>
#include"Graph.h"
int main()
{
        Graph* g; //creating an object of graph with 5 vertices
        g = new Graph(5);

        //inserting edges in the graph
        g->insertEdge(0, 1);
        g->insertEdge(0, 4);
        g->insertEdge(1, 0);
        g->insertEdge(1, 2);
        g->insertEdge(1, 3);
        g->insertEdge(1, 4);
        g->insertEdge(2, 1);
        g->insertEdge(2, 3);
        g->insertEdge(3, 1);
        g->insertEdge(3, 2);
        g->insertEdge(3, 4);
        g->insertEdge(4, 0);
        g->insertEdge(4, 1);
        g->insertEdge(4, 3);
        cout << "Graph\n";
        g->display();
        //display total number of edges
        cout << "Number of edges are " << g->numberOfEdges() << endl;

        //display degree of vertex number 4
        cout << "Degree of vertex " << g->degree(4) << endl;

        cout << "Output for Depth first search starting from vertex 0 " << endl;
        g->depthfirstSearch(0);
        cout << "Output for Breadth first search starting from vertex 0 " << endl;
        g->breadthfirstSearch(0);
        system("pause");
        return 0;
}
```


D:\Graphs\x64\Debug\Graphs
Graph
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 1 0 1
1 1 0 1 0
Number of edges are 7
Degree of vertex 3
Output for Depth first search starting from vertex 0
0 1 2 3 4
Output for Breadth first search starting from vertex 0
0 1 4 2 3
Press any key to continue . . .