Code:

```
#include <iostream>
#include <queue>
using namespace std;
class Matrix
{
private:
    int **adjMatrix;
    int vertices;
    bool *visitedNodes;

public:
    Matrix(int v)
    {
        vertices = v;
        adjMatrix = new int *[vertices];
        for (int i = 0; i < v; i++)
        {
            adjMatrix[i] = new int[vertices];
            for (int j = 0; j < v; j++)
            {
                adjMatrix[i][j] = 0;
            }
        }
    }

    void displayMatrix()
    {
        int i, j;
        cout << "\nAdjacency Matrix: " << endl;

        cout << "   ";
        for (i = 0; i < vertices; i++)
        {
            cout << i << " ";
        }
        cout << endl;

        for (i = 0; i < vertices; i++)
        {

            for (j = 0; j < vertices; j++)
            {
                if (j == 0)
                {
                    cout << i << ": ";
                }
                cout << adjMatrix[i][j] << " ";
            }
            cout << endl;
        }
    }
```

```
    void insertEdge(int u, int v)
    {
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
    }
    void BFS(queue<int> queue, bool *visitedNodes)
    {

        if (queue.empty())
            return;

        int visited = queue.front();
        queue.pop();

        cout << visited << " ";

        for (int i = 0; i < vertices; i++)
        {
            if (adjMatrix[visited][i] == 1 && (!visitedNodes[i]))
            {
                visitedNodes[i] = true;
                queue.push(i);
            }
        }
        BFS(queue, visitedNodes);
    }
    void DFS(bool *visitedNodes, int visited)
    {
        visitedNodes[visited] = true;
        cout << visited << " ";
        for (int i = 0; i < vertices; i++)
        {
            if (adjMatrix[visited][i] == 1 && (!visitedNodes[i]))
            {
                DFS(visitedNodes, i);
            }
        }
    }
};

int main()
{
    bool flag = true;
    int option, nodes, edges;

    cout << "Enter number of nodes: ";
    cin >> nodes;
    Matrix m(nodes);

jump:
    cout << "Enter number of edges: ";
    cin >> edges;
```

```
    if (edges > (nodes * (nodes - 1)) / 2)
    {
        cout << "Invalid Input!!! (Number of edges cannot be
greater than n(n-1)/2)" << endl;
        goto jump;
    }
    else if (edges < 0)
    {
        cout << "Number of edges cannot be negative!!!" << endl;
        goto jump;
    }
    else if (edges == 0)
    {
        cout << "Enter a non empty graph!!!" << endl;
        goto jump;
    }

    for (int i = 1; i <= edges; i++)
    {
        int a, b;
        cout << "Enter end nodes of edge " << i - 1 << ": " <<
endl;
        cin >> a >> b;
        m.insertEdge(a, b);
    }

    while (flag)
    {
        cout << "\n-------- Menu -------- \n1. Display Adjacency
Matrix \n2. Breadth First Search \n3. Depth First Search \n4. Exit
\nChoose Option: ";
        cin >> option;
        cout << endl;

        switch (option)
        {
        case 1:
            m.displayMatrix();
            break;
        case 2:
            int n_bfs;
        jump_bfs:
            cout << "Enter starting node: ";
            cin >> n_bfs;
            if (n_bfs >= 0 || n_bfs < nodes)
            {
                queue<int> queue;
                bool *visitedNodes = new bool[nodes];
                for (int i = 0; i < nodes; i++)
                {
                    visitedNodes[i] = false;
```

```
                }

                cout << "\nBFS on starting vertex " << n_bfs << ":
";
                queue.push(n_bfs);
                visitedNodes[n_bfs] = true;
                m.BFS(queue, visitedNodes);
            }
            else
            {
                cout << "Enter a node between 0 and " << nodes - 1
<< endl;
                goto jump_bfs;
            }
            break;
        case 3:
            int n_dfs;
        jump_dfs:
            cout << "Enter starting node: ";
            cin >> n_dfs;
            if (n_dfs >= 0 || n_dfs < nodes)
            {
                bool *visitedNodes = new bool[nodes];

                for (int i = 0; i < nodes; i++)
                {
                    visitedNodes[i] = false;
                }
                visitedNodes[n_dfs] = true;
                cout << "\nDFS on starting vertex " << n_dfs << ":
";
                m.DFS(visitedNodes, n_dfs);
            }
            else
            {
                cout << "Enter a node between 0 and " << nodes - 1
<< endl;
                goto jump_dfs;
            }
            break;
        case 4:
            flag = false;
            cout << "Successfully Terminated!!!" << endl;
            break;
        default:
            cout << "Invalid Input!!!" << endl;
            break;
        }
    }

    return 0;
}
```

```
Enter number of nodes: 5
Enter number of edges: 5
Enter end nodes of edge 0:
0
1
Enter end nodes of edge 1:
0
2
Enter end nodes of edge 2:
0
3
Enter end nodes of edge 3:
1
2
Enter end nodes of edge 4:
2
4

-------- Menu --------
1. Display Adjacency Matrix
2. Breadth First Search
3. Depth First Search
4. Exit
Choose Option: 2

Enter starting node: 2

BFS on starting vertex 2:  2 0 1 4 3
-------- Menu --------
1. Display Adjacency Matrix
2. Breadth First Search
3. Depth First Search
4. Exit
Choose Option: 3

Enter starting node: 2

DFS on starting vertex 2:  2 0 1 3 4
-------- Menu --------
1. Display Adjacency Matrix
2. Breadth First Search
3. Depth First Search
4. Exit
Choose Option:
```