```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <omp.h>

using namespace std;

// Graph class representing an undirected graph using adjacency list representation
class Graph {
private:
    int numVertices;          // Number of vertices
    vector<vector<int>> adj;  // Adjacency list

public:
    Graph(int vertices) : numVertices(vertices), adj(vertices) {}

    // Add an edge between two vertices
    void addEdge(int src, int dest) {
        adj[src].push_back(dest);
        adj[dest].push_back(src);
    }

    // View the graph
    void viewGraph() {
        cout << "Graph:\n";
        for (int i = 0; i < numVertices; i++) {
            cout << "Vertex " << i << " -> ";
            for (int neighbor : adj[i]) {
                cout << neighbor << " ";
            }
            cout << endl;
        }
    }

    // Perform Breadth First Search (BFS) in parallel
    void bfs(int startVertex) {
        vector<bool> visited(numVertices, false);
        queue<int> q;

        // Mark the start vertex as visited and enqueue it
        visited[startVertex] = true;
        q.push(startVertex);

        while (!q.empty()) {
            int currentVertex = q.front();
            q.pop();
            cout << currentVertex << " ";

            // Enqueue all adjacent unvisited vertices
            #pragma omp parallel for
            for (int neighbor : adj[currentVertex]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    q.push(neighbor);
                }
            }
        }
    }
```

```
    // Perform Depth First Search (DFS) in parallel
    void dfs(int startVertex) {
        vector<bool> visited(numVertices, false);
        stack<int> s;

        // Mark the start vertex as visited and push it onto the stack
        visited[startVertex] = true;
        s.push(startVertex);

        while (!s.empty()) {
            int currentVertex = s.top();
            s.pop();
            cout << currentVertex << " ";

            // Push all adjacent unvisited vertices onto the stack
            #pragma omp parallel for
            for (int neighbor : adj[currentVertex]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    s.push(neighbor);
                }
            }
        }
    }
};

int main() {
    int numVertices;
    cout << "Enter the number of vertices in the graph: ";
    cin >> numVertices;

    // Create a graph with the specified number of vertices
    Graph graph(numVertices);

    int numEdges;
    cout << "Enter the number of edges in the graph: ";
    cin >> numEdges;

    cout << "Enter the edges (source destination):\n";
    for (int i = 0; i < numEdges; i++) {
        int src, dest;
        cin >> src >> dest;
        graph.addEdge(src, dest);
    }

    // View the graph
    graph.viewGraph();

    int startVertex;
    cout << "Enter the starting vertex for BFS and DFS: ";
    cin >> startVertex;

    cout << "Breadth First Search (BFS): ";
    graph.bfs(startVertex);
    cout << endl;

    cout << "Depth First Search (DFS): ";
    graph.dfs(startVertex);
    cout << endl;
```

```
    return 0;
}
```