```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <omp.h>
using namespace std;

class Graph {
    int V;
    vector<vector<int>> adj;

public:
    Graph(int V) : V(V), adj(V) {}

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    // Parallel BFS using OpenMP
    void parallelBFS(int start) {
        vector<bool> vis(V, false);
        queue<int> q;
        q.push(start);
        vis[start] = true;

        while (!q.empty()) {
            int node;
            #pragma omp critical
            {
                if (!q.empty()) {
```

```
            node = q.front();
            q.pop();
        } else {
            node = -1;
        }
    }

    if (node == -1) continue;

    cout << node << " ";

    #pragma omp parallel for
    for (int i = 0; i < adj[node].size(); i++) {
        int neighbor = adj[node][i];
        bool expected = false;
        #pragma omp critical
        {
            if (!vis[neighbor]) {
                vis[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
  }
}

// Parallel DFS using OpenMP
void parallelDFS(int start) {
  vector<bool> vis(V, false);
  stack<int> s;
  s.push(start);
```

```
    while (!s.empty()) {
        int node;
        #pragma omp critical
        {
            if (!s.empty()) {
                node = s.top();
                s.pop();
            } else {
                node = -1;
            }
        }

        if (node == -1 || vis[node])
            continue;

        #pragma omp critical
        vis[node] = true;

        cout << node << " ";

        #pragma omp parallel for
        for (int i = 0; i < adj[node].size(); i++) {
            int neighbor = adj[node][i];
            #pragma omp critical
            if (!vis[neighbor]) {
                s.push(neighbor);
            }
        }
    }
}
```

```
};

int main() {
    int V, E, u, v, start;
    cout << "Vertices: ";
    cin >> V;
    Graph g(V);

    cout << "Edges: ";
    cin >> E;
    cout << "Enter edges:\n";
    for (int i = 0; i < E; i++) {
        cin >> u >> v;
        g.addEdge(u, v);
    }

    cout << "Start node: ";
    cin >> start;

    cout << "BFS: ";
    g.parallelBFS(start);

    cout << "\nDFS: ";
    g.parallelDFS(start);

    return 0;
}
```