#include <iostream>

#include <vector>

#include <queue>

#include <omp.h>

using namespace std;

class Graph {

int V;

vector<vector<int>> adj;

public:

Graph(int V) {

this->V = V;

adj.resize(V);

}

void addEdge(int u, int v) {

adj[u].push\_back(v);

adj[v].push\_back(u); // For undirected graph

}

void parallelBFS(int start) {

vector<bool> visited(V, false);

queue<int> q;

visited[start] = true;

q.push(start);

cout << "\nParallel BFS starting from node " << start << ":\n";

while (!q.empty()) {

int size = q.size();

vector<int> levelNodes;

#pragma omp parallel

{

vector<int> localNodes;

#pragma omp for

for (int i = 0; i < size; i++) {

int node = -1;

bool valid = false;

#pragma omp critical

{

if (!q.empty()) {

node = q.front();

q.pop();

valid = true;

}

}

if (!valid) continue;

localNodes.push\_back(node);

for (int neighbor : adj[node]) {

bool needVisit = false;

#pragma omp critical

{

if (!visited[neighbor]) {

visited[neighbor] = true;

q.push(neighbor);

needVisit = true;

}

}

}

}

#pragma omp critical

levelNodes.insert(levelNodes.end(), localNodes.begin(), localNodes.end());

}

for (int node : levelNodes)

cout << node << " ";

}

cout << endl;

}

void parallelDFSUtil(int node, vector<bool>& visited) {

bool alreadyVisited;

#pragma omp critical

{

alreadyVisited = visited[node];

if (!alreadyVisited) {

visited[node] = true;

cout << node << " ";

}

}

if (alreadyVisited) return;

#pragma omp parallel for

for (int i = 0; i < adj[node].size(); i++) {

int neighbor = adj[node][i];

#pragma omp task

parallelDFSUtil(neighbor, visited);

}

}

void parallelDFS(int start) {

vector<bool> visited(V, false);

cout << "\nParallel DFS starting from node " << start << ":\n";

#pragma omp parallel

{

#pragma omp single

parallelDFSUtil(start, visited);

}

cout << endl;

}

};

int main() {

int V, E;

cout << "Enter number of vertices: ";

cin >> V;

Graph g(V);

cout << "Enter number of edges: ";

cin >> E;

cout << "Enter each edge as two space-separated vertices (u v):\n";

for (int i = 0; i < E; i++) {

int u, v;

cin >> u >> v;

g.addEdge(u, v);

}

int start;

cout << "Enter starting node for traversal: ";

cin >> start;

g.parallelBFS(start);

g.parallelDFS(start);

return 0;

}