**Assignment 1  
  
BFS AND DFS**

#include <iostream>

#include <vector>

#include <queue>

#include <omp.h>

using namespace std;

struct Graph {

int V;

vector<vector<int>> adj;

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);

}

void BFS(int start) {

vector<bool> visited(V, false);

queue<int> q;

visited[start] = true;

q.push(start);

while (!q.empty()) {

int u = q.front();

q.pop();

cout << u << " ";

#pragma omp parallel for

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

int v = adj[u][i];

if (!visited[v]) {

#pragma omp critical

{

visited[v] = true;

q.push(v);

}

}

}

}

cout << endl;

}

void DFS(int start) {

vector<bool> visited(V, false);

#pragma omp parallel

{

#pragma omp single nowait

{

DFSUtil(start, visited);

}

}

cout << endl;

}

void DFSUtil(int u, vector<bool>& visited) {

visited[u] = true;

cout << u << " ";

# pragma omp parallel for

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

int v = adj[u][i];

if (!visited[v]) {

DFSUtil(v, visited);

}

}

}

};

int main() {

int V;

cout << "Enter the number of vertices: ";

cin >> V;

Graph g(V);

int edgeCount;

cout << "Enter the number of edges: ";

cin >> edgeCount;

cout << "Enter the edges (in format 'source destination'): \n";

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

int u, v;

cin >> u >> v;

g.addEdge(u, v);

}

cout << "BFS traversal starting from node 0: ";

g.BFS(0);

cout << "DFS traversal starting from node 0: ";

g.DFS(0);

return 0;

}