Bfs🡪

#include <iostream>

#include <vector>

#include <queue>

#include <omp.h>

using namespace std;

// Graph class representing the adjacency list

class Graph {

int V; // Number of vertices

vector<vector<int>> adj; // Adjacency list

public:

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

// Add an edge to the graph

void addEdge(int v, int w) {

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

}

// Parallel Depth-First Search

void parallelDFS(int startVertex) {

vector<bool> visited(V, false);

parallelDFSUtil(startVertex, visited);

}

// Parallel DFS utility function

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

visited[v] = true;

cout << v << " ";

#pragma omp parallel for

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

int n = adj[v][i];

if (!visited[n])

parallelDFSUtil(n, visited);

}

}

// Parallel Breadth-First Search

void parallelBFS(int startVertex) {

vector<bool> visited(V, false);

queue<int> q;

visited[startVertex] = true;

q.push(startVertex);

while (!q.empty()) {

int v = q.front();

q.pop();

cout << v << " ";

#pragma omp parallel for

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

int n = adj[v][i];

if (!visited[n]) {

visited[n] = true;

q.push(n);

}

}

}

}

};

int main() {

// Create a graph

Graph g(7);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 5);

g.addEdge(2, 6);

/\*

0 -------->1

| / \

| / \

| / \

v v v

2 ----> 3 4

| |

| |

v v

5 6

\*/

cout << "Depth-First Search (DFS): ";

g.parallelDFS(0);

cout << endl;

cout << "Breadth-First Search (BFS): ";

g.parallelBFS(0);

cout << endl;

return 0;

}

Dfs🡪

#include <iostream>

#include <stack>

#include <vector>

#include <omp.h>

using namespace std;

// Define the graph using adjacency list

struct Graph {

int V; // number of vertices

vector<vector<int>> adj; // adjacency list

Graph(int v) {

V = v;

adj.resize(V);

}

void addEdge(int v, int w) {

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

}

};

void parallelDFS(Graph& g, int source) {

stack<int> s;

vector<bool> visited(g.V, false);

// Initialize with the source node

s.push(source);

while (!s.empty()) {

int u = s.top();

s.pop();

if (!visited[u]) {

visited[u] = true;

#pragma omp parallel for

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

int v = g.adj[u][i];

// If v is not visited, add it to the stack

if (!visited[v]) {

s.push(v);

}

}

}

}

// Print the visited array to see the result

cout << "Visited nodes: ";

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

if (visited[i]) {

cout << i << " ";

}

}

cout << endl;

}

int main() {

cout << "\n---Parallel Depth First Search---" << endl;

int n;

cout << "Enter total number of pairs: ";

cin >> n;

Graph g(n);

int u, v;

cout << "Enter edges: \n";

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

cout << " ";

cin >> u >> v;

g.addEdge(u, v);

}

int source = 2;

parallelDFS(g, source);

return 0;

}