Bfs🡪

#include <iostream>

#include <queue>

#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 parallelBFS(Graph g, int source) {

queue<int> q;

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

// Initialize with the source node

q.push(source);

visited[source] = true;

while (!q.empty()) {

int u = q.front();

q.pop();

#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, mark it visited and enqueue it

if (!visited[v]) {

#pragma omp critical

{

visited[v] = true;

}

#pragma omp critical

{

q.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 << "---Parallel Breadth First Search---\n";

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;

parallelBFS(g, source);

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;

}