**Practical 1 : DFS ( Sequential and Parallel Algorithms )**

#include <iostream>

#include <vector>

#include <stack>

#include <omp.h>

#include <cstdlib>

using namespace std;

const int MAX = 100000;

vector<int> graph[MAX];

bool visited[MAX];

void dfs(int node) {

stack<int> s;

s.push(node);

while (!s.empty()) {

int curr\_node = s.top();

s.pop();

if (!visited[curr\_node]) {

visited[curr\_node] = true;

cout << curr\_node << " ";

#pragma omp parallel for

for (int i = 0; i < graph[curr\_node].size(); i++) {

int adj\_node = graph[curr\_node][i];

if (!visited[adj\_node]) {

s.push(adj\_node);

}

}

}

}

}

int main() {

cout << "This is Atharva Pingale's code";

cout << "\nPractical 1 : DFS ( Sequential and Parallel algorithms )";

int n, m, start\_node;

double start\_time, end\_time;

cout << "\n\nEnter number of nodes : ";

cin >> n;

cout << "Enter number of edges : ";

cin >> m;

cout << "Enter the starting node of the graph : ";

cin >> start\_node;

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

int random\_u = rand() % 999999;

int random\_v = rand() % 999999;

graph[random\_u].push\_back(random\_v);

graph[random\_v].push\_back(random\_u);

}

// Sequential Algorithm

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

visited[i] = false;

}

start\_time = omp\_get\_wtime();

dfs(start\_node);

end\_time = omp\_get\_wtime();

double seq\_time = end\_time - start\_time;

// Parallel Algorithm

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

visited[i] = false;

}

start\_time = omp\_get\_wtime();

dfs(start\_node);

end\_time = omp\_get\_wtime();

double parallel\_time = end\_time - start\_time;

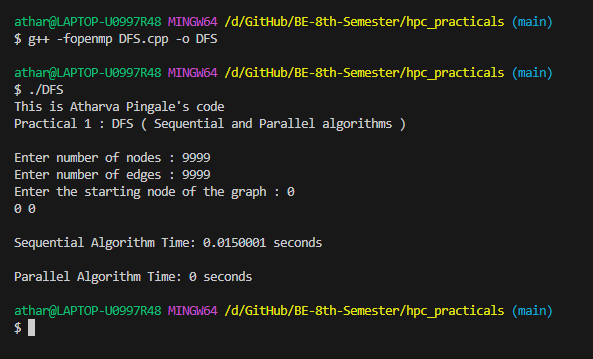
cout << "\n\nSequential Algorithm Time: " << seq\_time << " seconds\n";

cout << "\nParallel Algorithm Time: " << parallel\_time << " seconds\n";

return 0;

}

**Output :**

****