#include <iostream>

#include <vector>

#include <unordered\_set>

#include <unordered\_map>

#include <stack>

using namespace std;

class Graph {

unordered\_map<int, vector<int>> adjList;

public:

void addEdge(int u, int v) {

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

}

void DepthFirstSearch(int v, unordered\_set<int>& visited, unordered\_set<int>& group) {

visited.insert(v);

group.insert(v);

for (int child : adjList[v]) {

if (visited.find(child) == visited.end()) {

DepthFirstSearch(child, visited, group);

}

}

}

unordered\_set<int> findLargestGroup() {

unordered\_set<int> visited;

unordered\_set<int> largestGroup;

for (auto& pair : adjList) {

int node = pair.first;

if (visited.find(node) == visited.end()) {

unordered\_set<int> group;

DepthFirstSearch(node, visited, group);

if (group.size() > largestGroup.size()) {

largestGroup = group;

}

}

}

return largestGroup;

}

void displaylargest(unordered\_set<int>& largestGroup) {

cout << "Largest Connected Group: ";

for (int node : largestGroup) {

cout << node << " ";

}

cout << endl;

}

};

int main() {

Graph graph;

graph.addEdge(1, 2);

graph.addEdge(2, 3);

graph.addEdge(3, 1);

graph.addEdge(4, 5);

graph.addEdge(5, 6);

graph.addEdge(6, 2);

graph.addEdge(9, 3);

graph.addEdge(7, 1);

unordered\_set<int> largestGroup = graph.findLargestGroup();

graph.displaylargest(largestGroup);

return 0;

}