/\*\*

\* Group 13

\* Christopher Jantzen

\* Christopher Messmer

\* Nicholas Hemminger

\*/

#include <iostream>

#include <queue>

class Graph {

private:

int numV; // Number of vertices in the graph

int \*\*adjacencyMatrix; // Adjacency matrix of the graph

int\*\* buildEdges(int pairs[]); // Array with number of pairs

int maximum(int depths[]); // Maximum value of an array

public:

Graph(int numV); // numV is number of vertices

void buildAdjacencyMatrix(int pairs[]);

int\* BFS(int v); // v is starting point

int diameter();

int\*\* Component();

// Test methods

void printAdjMatrix() {

std::cout << "Adjacency Matrix\n";

for (int i=0; i<this->numV; i++) {

for (int j=0; j<this->numV; j++) {

std::cout << this->adjacencyMatrix[i][j] << " ";

}

std::cout << "\n";

}

std::cout << "\n";

}

void showQ(std::queue<int> gq) {

std::queue<int> g = gq;

while (!g.empty())

{

std::cout << " " << g.front();

g.pop();

}

std::cout << '\n';

}

};

Graph::Graph(int numV) {

this->numV = numV;

}

int\*\* Graph::buildEdges(int pairs[]) {

int\*\* edges = 0;

edges = new int\*[pairs[0]];

int index = 1;

int edgesIndex = 0;

while(true) {

if (pairs[index] == -1) {

break;

} else {

edges[edgesIndex] = new int[2];

edges[edgesIndex][0] = pairs[index];

edges[edgesIndex][1] = pairs[index+1];

edgesIndex++;

}

index += 2;

}

std::cout << "Edges:\n{";

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

std::cout << "{";

for (int j=0; j<2; j++) {

if (j == 0) {

std::cout << edges[i][j] << ", ";

} else {

std::cout << edges[i][j];

}

}

if (i == edgesIndex-1) {

std::cout << "}";

} else {

std::cout << "}, ";

}

}

std::cout << "}\n\n";

return edges;

}

void Graph::buildAdjacencyMatrix(int pairs[]) {

int \*\*edges = buildEdges(pairs);

int\*\* aMatrix = 0;

aMatrix = new int\*[this->numV];

for (int i=0; i<this->numV; i++) {

aMatrix[i] = new int[this->numV];

for (int j=0; j<this->numV; j++) {

aMatrix[i][j] = 0;

}

}

for (int i=0; i<pairs[0]; i++) {

aMatrix[edges[i][0]][edges[i][1]] = 1;

aMatrix[edges[i][1]][edges[i][0]] = 1;

}

this->adjacencyMatrix = aMatrix;

printAdjMatrix();

}

int\* Graph::BFS(int v) {

// List of "visited" booleans for each vertex

bool \*visited = new bool[this->numV];

for (int i=0; i<this->numV; i++) {

visited[i] = false;

}

std::queue<int> bfsQ;

int distance[this->numV];

distance[v] = 0;

visited[v] = true;

bfsQ.push(v);

while(!bfsQ.empty()) {

v = bfsQ.front();

// showQ(bfsQ);

bfsQ.pop();

for (int i=0; i<this->numV; i++) {

if((adjacencyMatrix[v][i] == 1) && !visited[i]) {

visited[i] = true;

bfsQ.push(i);

distance[i] = distance[v]+1;

}

}

}

int depth = 0;

for (int i=0; i<this->numV; i++) {

if (visited[i] == false) {

depth = -1;

}

}

if (depth != -1) {

depth = maximum(distance);

}

int\* set;

int numVisited = 0;

for (int i=0; i<this->numV; i++) {

if(visited[i] == true) numVisited++;

}

set = new int[numVisited + 2];

int index = 1;

int numberCheckingIfVisited = 0;

while(index < numVisited+1) {

if(visited[numberCheckingIfVisited] == true) {

set[index++] = numberCheckingIfVisited;

}

numberCheckingIfVisited++;

}

set[0] = numVisited;

set[numVisited+1] = depth;

return set;

}

int Graph::maximum(int depths[]) {

int greatest = depths[1];

for (int i=0; i<this->numV; i++) {

if (greatest < depths[i]) {

greatest = depths[i];

}

}

return greatest;

}

int Graph::diameter() {

int arr[this->numV];

int \*temp;

int index = 0;

for (int i=0; i<this->numV; i++) {

temp = BFS(i);

arr[i] = temp[temp[0]+1];

if (arr[i] < 0) {

return -1;

} else if (arr[i] > this->numV) {

return -1;

}

}

return maximum(arr);

}

int\*\* Graph::Component() {

int\*\* sets;

sets = new int\*[this->numV]; // max space needed if only individual nodes

int sizes[this->numV];

for (int i=0; i<this->numV; i++) {

sizes[i] = 0;

}

sets[0] = BFS(0);

sizes[0] = sets[0][0];

if (sets[0][0] == this->numV) {

for (int i=1; i<this->numV+1; i++) {

std::cout << sets[0][i] << " ";

}

std::cout << std::endl;

return sets;

}

bool \*visited = new bool[this->numV];

for (int i=0; i<this->numV; i++) {

visited[i] = false;

}

for (int i=0; i<this->numV; i++) {

visited[sets[0][i]] = true;

}

int setsIndex = 1;

while (true) {

for (int i=1; i<sets[0][0]+1; i++) {

if (!visited[i]) {

sets[setsIndex] = BFS(i);

sizes[setsIndex] = sets[setsIndex][0];

setsIndex++;

continue;

}

}

break;

}

std::cout << "Component sets\n";

for (int i=0; i<this->numV; i++) {

for (int j=1; j<sizes[i]+1; j++) {

std::cout << sets[i][j] << " ";

}

std::cout << std::endl;

}

return sets;

}

int main() {

int n = 6;

int arr[] = {5, 0, 1, 0, 2, 1, 5, 2, 3, 3, 5, -1}; // Test case 1

// int arr[] = {5, 0, 1, 0, 2, 1, 5, 2, 3, 3, 4, -1}; // Test case 2

Graph \*g = new Graph(n);

g->buildAdjacencyMatrix(arr);

int \*set = g->BFS(3);

int maxDepth = g->diameter();

std::cout << "Diameter: " << maxDepth << "\n\n";

int \*\*componentSet = g->Component();

return 0;

}