Name: ANYA AGARWAL

Section: B (13)

Roll no: 2021122

I. Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)

Input Format:

Input will be the graph in the form of adjacency matrix or adjacency list.

Source vertex number and destination vertex number is also provided as an input.

Output Format:

Output will be 'Yes Path Exists' if path exists, otherwise print 'No Such Path Exists'

#include <iostream>

#include <fstream>

#include <vector>

#include <list>

#include <unordered\_map>

using namespace std;

class graph

{

public:

unordered\_map<int, list<int>> adj;

void addEdge(int u, int v, bool direction)

{

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

if (!direction)

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

}

bool dfs(int src, int des, vector<bool> &vis)

{

vis[src] = true;

for (auto nbr : adj[src])

{

if (!vis[nbr])

{

if (nbr == des)

return true;

if (dfs(nbr, des, vis))

return true;

}

}

return false;

}

};

int main(void)

{

ifstream inputFile("inputs.txt");

ofstream outputFile("outputs.txt");

if (!inputFile)

{

cerr << "Error opening input file." << endl;

return 1;

}

int T;

inputFile >> T;

while (T--)

{

int n = 0;

inputFile >> n;

int m = 0;

inputFile >> m;

graph g;

vector<bool> vis(n + 1, false);

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

{

int u, v;

inputFile >> u >> v;

g.addEdge(u, v, 0);

}

int src, des;

inputFile >> src >> des;

if (!outputFile)

{

cerr << "Error opening output file." << endl;

return 1;

}

if (g.dfs(src, des, vis))

{

outputFile << "Yes path exist\n";

}

else

{

outputFile << "No path exist\n";

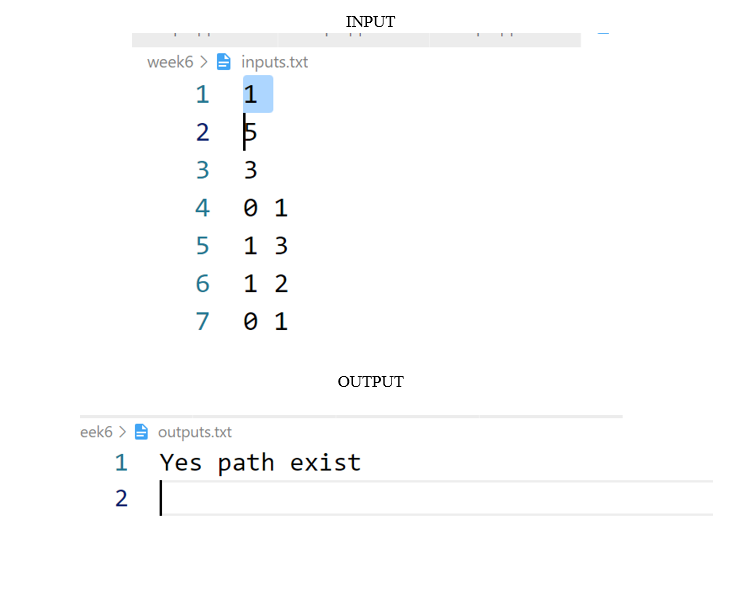
}

}

outputFile.close();

return 0;

}



II. Given a graph, design an algorithm and implement it using a program to find if a graph is

bipartite or not. (Hint: use BFS)

Input Format:

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Output will be 'Yes Bipartite' if graph is bipartite, otherwise print 'Not Bipartite'

#include <iostream>

#include <fstream>

#include <vector>

#include <list>

#include<queue>

#include <unordered\_map>

using namespace std;

class graph {

public:

unordered\_map<int, list<int>> adj;

void addEdge(int u, int v, bool direction) {

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

if (!direction)

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

}

bool bipartite(int V) {

vector<int> colour(V,-1);

colour[0]=0;

queue<int> q;

q.push(0);

while(!q.empty())

{

int node=q.front();

q.pop();

for(auto it:adj[node])

{

if(colour[it]==-1)

{

colour[it]=!colour[node];

q.push(it);

}

else if(colour[it]==colour[node])

{

return false;

}

}

}

return true;

}

};

int main(void) {

ifstream inputFile("input.txt");

ofstream outputFile("outputs.txt");

if (!inputFile) {

cerr << "Error opening input file." << endl;

return 1;

}

int T;

inputFile >> T;

while (T--) {

int n = 0;

inputFile >> n;

int m = 0;

inputFile >> m;

graph g;

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

int u, v;

inputFile >> u >> v;

g.addEdge(u, v, 0);

}

// int src, des;

// inputFile >> src >> des;

if (!outputFile) {

cerr << "Error opening output file." << endl;

return 1;

}

if (g.bipartite(n)) {

outputFile << "Graph is bipartite\n";

} else {

outputFile << "Graph is not bipartite\n";

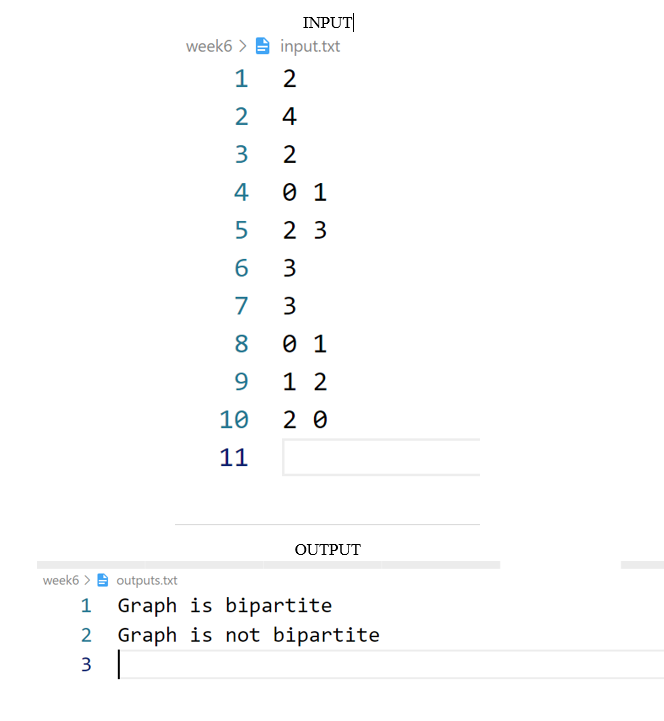
}

}

outputFile.close();

return 0;

}



III. Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

Input Format:

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Output will be 'Yes Cycle Exists' if cycle exists otherwise print 'No Cycle Exists'.

#include <iostream>

#include <fstream>

#include <vector>

#include <list>

#include <queue>

#include <unordered\_map>

using namespace std;

class graph

{

public:

unordered\_map<int, list<int>> adj;

void addEdge(int u, int v, bool direction)

{

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

if (!direction)

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

}

bool dfsCheck(int node, int vis[], int pathVis[])

{

vis[node] = 1;

pathVis[node] = 1;

for (auto it : adj[node])

{

if (!vis[it])

{

if (dfsCheck(it, vis, pathVis) == true)

return true;

}

else if (pathVis[it])

{

return true;

}

}

pathVis[node] = 0;

return false;

}

bool isCycle(int V)

{

int vis[V] = {0};

int pathVis[V] = {0};

for (int i = 0; i < V; i++)

{

if (!vis[i])

{

if (dfsCheck(i, vis, pathVis) == true)

return true;

}

}

return false;

}

};

int main(void)

{

ifstream inputFile("input1.txt");

ofstream outputFile("outputs.txt");

if (!inputFile)

{

cerr << "Error opening input file." << endl;

return 1;

}

int T;

inputFile >> T;

while (T--)

{

int n = 0;

inputFile >> n;

int m = 0;

inputFile >> m;

graph g;

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

{

int u, v;

inputFile >> u >> v;

g.addEdge(u, v, 1);

}

if (!outputFile)

{

cerr << "Error opening output file." << endl;

return 1;

}

if (g.isCycle(n))

{

outputFile << "Yes Cycle Exists\n";

}

else

{

outputFile << "No Cycle Exists\n";

}

}

outputFile.close();

return 0;

}

