WEEK 6:

/\*

NAME : NANDINI MAURYA

UNIVERSITY ROLL NUMBER : 2021320

SECTION : B

PROBLEM 1 : 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>

using namespace std;

bool dfsutils(vector<int> G[], vector<bool>& visited, int src, int dest) {

    if (src == dest)

        return true;

    visited[src] = true;

    for (int i = 0; i < G[src].size(); i++) {

        if (!visited[G[src][i]]) {

            if (dfsutils(G, visited, G[src][i], dest))

                return true;

        }

    }

    return false;

}

bool DFS(vector<int> G[], int v, int src, int dest) {

    vector<bool> visited(v, false);

    return dfsutils(G, visited, src, dest);

}

int main() {

    ifstream inputFile("C:\\Users\\user\\Desktop\\Lab questions\\codes\\C++\\input3.txt");

    ofstream outputFile("C:\\Users\\user\\Desktop\\Lab questions\\codes\\C++\\output3.txt");

    if (!inputFile.is\_open()) {

        cerr << "Error opening the file!" << endl;

        return 1;

    }

    int test;

    inputFile >> test;

    while (test--) {

        int v, e;

        inputFile >> v >> e;

        int maxi; *// Assuming this is the maximum number of vertices*

        inputFile >> maxi;

        vector<int> adj[maxi + 1];

        for (int i = 0; i < e; i++) {

            int u, v;

            inputFile >> u >> v;

            adj[u].push\_back(v);

            adj[v].push\_back(u);

        }

        int src, dest;

        inputFile >> src >> dest;

        if (DFS(adj, maxi + 1, src, dest))

           outputFile  << "Yes, Path Exists" << endl;

        else

           outputFile  <<"No Such Path Exists" << endl;

    }

    inputFile.close();

    outputFile.close();

    return 0;

}

\*\*\*\*\*\*\*\*\*INPUT \*\*\*\*\*\*\*\*\*

4

6 5

6

1 2

1 3

2 4

3 5

4 6

1 6

5 4

5

1 2

2 3

3 4

4 5

1 5

5 3

5

1 2

1 3

2 4

1 5

5 4

6

1 2

1 3

2 4

3 4

1 5

\*\*\*\*\*\*\*\*\*OUTPUT \*\*\*\*\*\*\*\*\*

Yes, Path Exists

Yes, Path Exists

No Such Path Exists

No Such Path Exists

/\*

NAME : NANDINI MAURYA

UNIVERSITY ROLL NUMBER : 2021320

SECTION : B

PROBLEM 2 : 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 <bits/stdc++.h>

#include <vector>

using namespace std;

enum Color { UNCOLORED, RED, BLUE };

bool isBipartite(vector<int> G[], int v, int s) {

    vector<Color> color(v + 1, UNCOLORED);

    color[s] = RED;

    queue<int> q;

    q.push(s);

    while (!q.empty()) {

        int src = q.front();

        q.pop();

        for (int i = 0; i < G[src].size(); i++) {

            int neighbor = G[src][i];

            if (color[neighbor] == UNCOLORED) {

*// Assign opposite color to neighbor*

                color[neighbor] = (color[src] == RED) ? BLUE : RED;

                q.push(neighbor);

            } else if (color[neighbor] == color[src]) {

*// If adjacent vertices have same color, graph is not bipartite*

                return false;

            }

        }

    }

    return true;

}

int main() {

    ifstream inputFile("C:\\Users\\user\\Desktop\\Lab questions\\codes\\C++\\input4.txt");

    ofstream outputFile("C:\\Users\\user\\Desktop\\Lab questions\\codes\\C++\\output4.txt");

    if (!inputFile.is\_open()) {

        cerr << "Error opening the file!" << endl;

        return 1;

    }

    int test;

    inputFile >> test;

    while (test--) {

        int v, e;

        inputFile >> v >> e;

        int maxi; *// Assuming this is the maximum number of vertices*

        inputFile >> maxi;

        vector<int> adj[maxi + 1];

        for (int i = 0; i < e; i++) {

            int u, v;

            inputFile >> u >> v;

            adj[u].push\_back(v);

            adj[v].push\_back(u);

        }

    bool bipartite = isBipartite(adj, v, 1);

    if (bipartite) {

        outputFile << "Yes Bipartite" << endl;

    } else {

        outputFile << "Not Bipartite" << endl;

    }

    }

    return 0;

}

\*\*\*\*\*\*\*\*\*\*\*INPUT\*\*\*\*\*\*\*\*\*\*\*\*

4

5 4

5

1 2

1 3

2 4

3 5

4 4

4

1 2

2 3

3 4

4 1

6 5

6

1 2

1 3

2 4

3 5

4 6

3 3

3

1 2

2 3

3 1

\*\*\*\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\*\*\*\*\*

Yes Bipartite

Yes Bipartite

Yes Bipartite

Not Bipartite

/\*

NAME : NANDINI MAURYA

UNIVERSITY ROLL NUMBER : 2021320

SECTION : B

PROBLEM 3 : 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 <bits/stdc++.h>

#include <vector>

using namespace std;

bool dfsutils(vector<int> G[], vector<bool>& visited, int src, int parent) {

    visited[src] = true;

    for (int i = 0; i < G[src].size(); i++) {

        int neighbor = G[src][i];

        if (!visited[neighbor]) {

            if (dfsutils(G, visited, neighbor, src))

                return true;

        } else if (neighbor != parent) {

*// If the neighbor is visited and not the parent, then it's a cycle*

            return true;

        }

    }

    return false;

}

bool DFS(vector<int> G[], int v) {

    vector<bool> visited(v + 1, false);

    for (int i = 1; i <= v; i++) {

        if (!visited[i] && dfsutils(G, visited, i, -1))

            return true;

    }

    return false;

}

int main() {

    ifstream inputFile("C:\\Users\\user\\Desktop\\Lab questions\\codes\\C++\\input5.txt");

    ofstream outputFile("C:\\Users\\user\\Desktop\\Lab questions\\codes\\C++\\output5.txt");

    if (!inputFile.is\_open()) {

        cerr << "Error opening the file!" << endl;

        return 1;

    }

    int test;

    inputFile >> test;

    while (test--) {

        int v, e;

        inputFile >> v >> e;

        int maxi; *// Assuming this is the maximum number of vertices*

        inputFile >> maxi;

        vector<int> adj[maxi + 1];

        for (int i = 0; i < e; i++) {

            int u, v;

            inputFile >> u >> v;

            adj[u].push\_back(v);

            adj[v].push\_back(u);

        }

    if (DFS(adj, v))

        outputFile << "Yes, Cycle Exists" << endl;

    else

        outputFile << "No Cycle Exists" << endl;

    }

    return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*INPUT \*\*\*\*\*\*\*\*\*\*\*\*

4

3 3

3

1 2

2 3

3 1

4 3

4

1 2

2 3

2 4

4 4

4

1 2

2 3

3 4

4 1

5 4

5

1 2

2 3

3 4

4 5

\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT \*\*\*\*\*\*\*\*\*\*\*\*

Yes, Cycle Exists

No Cycle Exists

Yes, Cycle Exists

No Cycle Exists