//PROBLEM 1(WEEK 6)

#include <bits/stdc++.h>

using namespace std;

void dfs(vector<int> arr[], int source, int V, bool \*visited)

{

visited[source] = true;

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

{

if (arr[source][i] != 0 && !visited[i])

{

dfs(arr, i, V, visited);

}

}

}

bool checkPath(vector<int> arr[], int V, int source, int destination)

{

bool visited[V];

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

visited[i] = false;

dfs(arr, source, V, visited);

return visited[destination];

}

int main()

{

int n;

cin >> n;

vector<int> arr[n];

int temp;

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

{

for (int j = 0; j < n; j++)

{

cin >> temp;

arr[i].push\_back(temp);

}

}

int source, destination;

cin >> source >> destination;

if (checkPath(arr, n, source - 1, destination - 1))

{

cout << "Yes Path Exists.\n";

}

else

{

cout << "No Such Path Exists.\n";

}

return 0;

}

***OUTPUT***

3

1 1 0

1 0 1

0 1 1

1 3

Yes Path Exists.

//PROBLEM 2(WEEK 6)

#include <bits/stdc++.h>

using namespace std;

bool isBipartiteUtil(vector<int> G[], int src, int colorArr[], int V)

{

colorArr[src] = 1;

queue<int> q;

q.push(src);

while (!q.empty())

{

int u = q.front();

q.pop();

if (G[u][u] == 1)

return false;

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

{

if (G[u][v] != 0 && colorArr[v] == -1)

{

colorArr[v] = 1 - colorArr[u];

q.push(v);

}

else if (G[u][v] != 0 && colorArr[v] == colorArr[u])

return false;

}

}

return true;

}

bool isBipartite(vector<int> G[], int V)

{

int colorArr[V];

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

colorArr[i] = -1;

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

if (colorArr[i] == -1)

if (isBipartiteUtil(G, i, colorArr, V) == false)

return false;

return true;

}

int main()

{

int n;

cin >> n;

vector<int> G[n];

int temp;

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

{

for (int j = 0; j < n; j++)

{

cin >> temp;

G[i].push\_back(temp);

}

}

if (isBipartite(G, n))

{

cout << "Yes Bipartite\n";

}

else

{

cout << "Not Bipartite\n";

}

return 0;

}

***OUTPUT***

3

1 1 0

1 1 0

1 0 1

Not Bipartite

//PROBLEM 3(WEEK 6)

#include <bits/stdc++.h>

using namespace std;

bool CheckCycle(int node, vector<int> adj[], int vis[], int dfsvis[])

{

vis[node] = 1;

dfsvis[node] = 1;

for (auto it : adj[node])

{

if (!vis[it])

{

if (CheckCycle(it, adj, vis, dfsvis))

return true;

}

else if (dfsvis[it])

return true;

}

dfsvis[node] = 0;

return false;

}

bool isCycle(vector<int> adj[], int N)

{

int vis[N + 1], dfsVis[N + 1];

memset(vis, 0, sizeof(vis));

memset(dfsVis, 0, sizeof(dfsVis));

for (int i = 1; i <= N; i++)

{

if (!vis[i])

{

if (CheckCycle(i, adj, vis, dfsVis))

return true;

}

}

return false;

}

int main()

{

int n, m;

cin >> n >> m;

vector<int> adj[n + 1];

for (int i = 1; i <= m; i++)

{

int u, v;

cin >> u >> v;

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

}

if (isCycle(adj, n))

cout << "Cycle Exists" << endl;

else

cout << "No Cycle Exists" << endl;

return 0;

}

***OUTPUT***

5

1 0 1 0 0

No Cycle Exists