**//1. BFS Implementation**

//BFS Implementation

#include<bits/stdc++.h>

using namespace std;

const int N = 10000;

vector<int> g[N];

vector<int> level(N);

void bfs(int src)

{

    queue<int> q;

    vector<bool> vis(N);

    q.push(src); vis[src] = 1;

    while(!q.empty())

    {

        int node = q.front();

        q.pop();

        cout<<node<<"\n";

        // for(int i = 0; i<g[node].size(); i++)

        // {

        //     int child = g[node][i];

        //     if(vis[child]) continue;

        //     q.push(child); vis[child] = 1;

        //     level[child] = level[node]+1;

        // }

        for(auto child : g[node])

        {

            if(vis[child]) continue;

            q.push(child); vis[child] = 1;

            level[child] = level[node]+1;

        }

    }

}

int main()

{

    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);

    int n,e;

    n = 9, e = 10;

    for(int i = 0; i<e; i++)

    {

        int x, y;

        cin>>x>>y;

        g[x].push\_back(y);

        g[y].push\_back(x);

    }

    bfs(1);

    for(int i = 1; i<=n; i++)

    {

        cout<<"Node "<<i<<"  Level : "<<level[i]<<"\n";

    }

}

/\*

Input

1 2

1 3

2 4

2 5

3 6

3 7

3 8

4 5

5 9

6 9

Output

1

2

3

4

5

6

7

8

9

\*/

**//2.BFS Problem**

**//SPOJ Problem**

**//NAKAN Minimum Knight Moves**

//https://www.spoj.com/problems/NAKANJ/

#include<bits/stdc++.h>

using namespace std;

const int INF = 1e9+1;

const int N = 8;

vector<int>g[N];

vector<pair<int,int> > movements = {

    {1,2},{1,-2},{-1,2},{-1,-2}, {2,1},{-2,1},{2,-1},{-2,-1}

};

bool isValid(int x, int y){

    return x>=0 && x<=7 && y>=0 && y<=7;

}

int bfs(pair<int,int> src, pair<int,int> des)

{

    queue<pair<int,int> > q;

    int visited[8][8] = {0};

    int level[8][8];

    for(int i = 0; i<8; i++)

        for(int j = 0; j<8; j++)

            level[i][j] = INF;

    visited[src.first][src.second] = 1;

    level[src.first][src.second] = 0;

    q.push(src);

    while(!q.empty())

    {

        pair<int,int> cur\_v = q.front();

        q.pop();

        int x = cur\_v.first, y = cur\_v.second;

        for(auto move : movements)

        {

            int childx = x + move.first;

            int childy = y + move.second;

            if(isValid(childx, childy))

            {

                if(!visited[childx][childy])

                {

                    q.push({childx, childy});

                    visited[childx][childy] = 1;

                    level[childx][childy] = level[x][y] + 1;

                }

            }

        }

        if(level[des.first][des.second] != INF)

            break;

    }

    return level[des.first][des.second];

}

int main()

{

    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);

    int t; cin>>t;

    string s1, s2;

    while(t--)

    {

        cin>>s1>>s2;

        pair<int,int> src = {s1[0] - 'a', s1[1] - '1' };

        pair<int,int> des = {s2[0] - 'a', s2[1] - '1'};

        cout<<bfs(src,des)<<"\n";

    }

}

/\*

Input:

3

a1 h8

a1 c2

h8 c3

Output:

6

1

4

\*/

**//3.BFS Problem**

**//Bipartite Graph**

//BFS | Bipartite Check | (Disconnected component)

//BPT Graph : No add cycle exits

#include<bits/stdc++.h>

using namespace std;

const int N = 1e7+10;

vector<int>adj[N];

vector<int>side(N,-1);

bool isBP(int src)

{

    queue<int>q;

    q.push(src);

    side[src] = 1;

    while(!q.empty())

    {

        int vertex = q.front();

        q.pop();

        for(auto child : adj[vertex])

        {

            if(side[child] == -1)

            {

                q.push(child);

                side[child] = side[vertex]^1;

            }

            else if(side[child] == side[vertex])

            {

                return false;

            }

        }

    }

    return true;

}

bool isBipartite(int V){

    for(int i = 1; i<=V; i++){ // Vertices are from 1 to V

        if(side[i] == -1){

            if(isBP(i) == false){

                return false;

            }

        }

    }

    return true;

}

int main(){

    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);

    int V, E;

    cin >> V >> E;

    for(int i = 0; i < E; i++){

        int u, v;

        cin >> u >> v;

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    cout<<isBipartite(V);

    return 0;

}

/\*

input

4 5

1 2

1 4

2 3

3 4

1 3

output

0

**//4.BFS Problem GFG**

**//Steps by Knight**

//BFS

//GFG : Steps by Knight6

//https://practice.geeksforgeeks.org/problems/steps-by-knight5927/1?page=1&category[]=BFS&sortBy=submissions

#include<bits/stdc++.h>

using namespace std;

vector<pair<int,int>> moves = {

    {-2,-1},{-2,1},{-1,-2},{-1,2},{1,-2},{1,2},{2,-1},{2,1}

};

bool isValid(int x,int y, int N){

    return x >= 1 && y >= 1 && x <= N && y<= N;

}

int minStepToReachTarget(vector<int>&KnightPos, vector<int>&TargetPos, int N)

{

    int level[N+1][N+1];

    for(int i = 1; i<=N; i++){

        for(int j = 1; j<=N; j++){

            level[i][j] = -1;

        }

    }

    queue<pair<int,int>>q;

    q.push({KnightPos[0],KnightPos[1]});

    level[KnightPos[0]][KnightPos[1]] = 0;

    while(!q.empty())

    {

        pair<int,int> node = q.front();

        q.pop();

        for(auto move : moves)

        {

            int childx = move.first + node.first;

            int childy = move.second + node.second;

            if(isValid(childx,childy,N)){

                if(level[childx][childy] == -1){

                    q.push({childx,childy});

                    level[childx][childy] = level[node.first][node.second]+1;

                }

            }

            if(level[TargetPos[0]][TargetPos[1]] != -1)

            {

                return level[TargetPos[0]][TargetPos[1]];

            }

        }

    }

}

int main(){

    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);

    int t; cin>>t;

    while(t--)

    {

        vector<int> KnightPos(2);

        vector<int> TargetPos(2);

        int N; cin>>N;

        cin>>KnightPos[0]>>KnightPos[1];

        cin>>TargetPos[0]>>TargetPos[1];

        cout<< minStepToReachTarget(KnightPos,TargetPos,N) <<"\n";

    }

    return 0;

}

**//5.BFS**

**Connected Component**

//BFS Connected Component

#include<bits/stdc++.h>

using namespace std;

const int N = 10000;

vector<int> g[N];

vector<bool> vis(N);

set<int> allNodes;

void bfs(int src)

{

    queue<int> q;

    q.push(src); vis[src] = 1;

    while(!q.empty())

    {

        int node = q.front();

        q.pop();

        for(auto child : g[node])

        {

            if(vis[child]) continue;

            q.push(child); vis[child] = 1;

        }

    }

}

int main()

{

    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);

    int n,e; cin>>e;

    for(int i = 0; i<e; i++)

    {

        int x, y; cin>>x>>y;

        allNodes.insert(x); allNodes.insert(y);

        g[x].push\_back(y); g[y].push\_back(x);

    }

    int connected\_component = 0;

    for(auto it : allNodes)

    {

        if(vis[it]) continue;

        bfs(it);

        connected\_component++;

    }

    cout<<connected\_component<<"\n";

}

/\*

Input

7

2 4

2 5

3 6

3 7

3 8

4 5

5 9

Output

2

\*/

**//6.BFS**

**Distance and Shortest Path from 4 to 6**

//BFS | Distance and Shortedst Path from 4 to 6.

#include<bits/stdc++.h>

using namespace std;

const int N = 10000;

vector<int> g[N];

vector<bool> vis(N);

vector<int> level(N);

vector<int> parent(N);

void input(int e)

{

    for(int i = 0; i<e; i++)

    {

        int x, y;

        cin>>x>>y;

        g[x].push\_back(y);

        g[y].push\_back(x);

    }

}

void bfs(int src, int des)

{

    queue<int> q;

    q.push(src); vis[src] = 1;

    parent[src] = src;

    while(!q.empty())

    {

        int node = q.front();

        q.pop();

        for(int i = 0; i<g[node].size(); i++)

        {

            int child = g[node][i];

            if(vis[child]) continue;

            q.push(child); vis[child] = 1;

            parent[child] = node; // Save parent of child to node.

            level[child] = level[node]+1;

            if(child == des) return; // We reached our destination

        }

    }

}

void Print\_Path(int src,int des)

{

    cout<<"Path: ";

    stack<int> st;

    while(des != src)

    {

        st.push(des);

        des = parent[des];

    }

    st.push(src);

    while(st.size()>1)

    {

        cout<<st.top()<<"-> ";

        st.pop();

    }

    cout<<st.top()<<"\n";

}

int main()

{

    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);

    int n = 9, e = 10;

    int src = 4, des = 6;

    input(e);

    bfs(src,des);

    cout<<"Distance = "<<level[des]<<"\n";

    Print\_Path(src,des);

}

/\*

Input

1 2

1 3

2 4

2 5

3 6

3 7

3 8

4 5

5 9

6 9

Output

Distance = 3

Path: 4-> 5-> 9-> 6

\*/

**//7.BFS Problem**

**//UVA 10653**

**//Bombs! NO they Are Mines**

//https://vjudge.net/problem/UVA-10653

//UVA 10653 Bombs! NO they are Mines!!

//Using BFS | Converting 2D array to 1D array.

#include<bits/stdc++.h>

using namespace std;

const int N = 1000;

int R,C;            // Number of ROW and Number of Column

vector<int> g[N\*N]; //List of size N\*N

vector<int> level(N\*N);

bool isValid(int idx)

{

    int x = idx/N;  // Getting the Row number

    int y = idx%N;  //Getting the column number

    return x>=0 && y>=0 && x<R && y<C && level[idx] == -1;

}

int bfs(int src, int des)

{

    queue<int> q;

    q.push(src); level[src] = 0;

    while(!q.empty())

    {

        int node = q.front();

        q.pop();

        vector<int> itr = {1,-1,N,-N};

        for(int i : itr)

            if(isValid(node+i)) g[node].push\_back(node+i);

        for(auto child : g[node])

        {

            q.push(child);

            level[child] = level[node]+1;

            if(child == des) return level[child];

        }

    }

}

int main()

{

    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);

    while(true)

    {

        cin>>R>>C;

        if(!R & !C) return 0;

        fill(level.begin(), level.end(), -1);

        int ROW; cin>>ROW;

        for(int i = 0; i<ROW; i++)

        {

            int r\_no; cin>>r\_no;

            int bomb; cin>>bomb;

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

            {

                int c\_no; cin>>c\_no;

                level[r\_no\*N + c\_no] = -2;

            }

        }

        int sx, sy, dx, dy;

        cin>>sx>>sy>>dx>>dy;

        int src = sx\*N + sy;

        int des = dx\*N + dy;

        cout<<bfs(src,des)<<"\n";

    }

}

/\*

Input

10 10

9

0 1 2

1 1 2

2 2 2 9

3 2 1 7

5 3 3 6 9

6 4 0 1 2 7

7 3 0 3 8

8 2 7 9

9 3 2 3 4

0 0

9 9

5 4

3

0 1 3

1 1 2

2 2 1 2

0 0

1 3

0 0

Output

18

8

\*/