Friendship and Gang

Problem statement :
There are **N** no of students.
You will be given the friendship between some students.You have to return no of gangs.
Note: Group of students who are friend to atleast one of the student in that group is called as **Gang**.

Examlpe N=6

```
Given

1,2 are friends

3,4 are friends

5,6 are friends

1,5 are friends

Output:2 gangs
```

Explanation: 1,2 are friends and 1,5 are also friends and 5,6 are friends they will from a gang.similarly 3,4 will form a gang. one gang contain 1,2,5,6
another gang contain 3,4

Input Format

- First line of input contain no of testcases **T**
- For each test case you are given **N** no of student and **M** no of friendships.
- Next **M** lines contain friendships between two students.

Constraints

- **0 <= T <=1000**
- **0 <= N <=10**$_6$
- **0 <= M <=N**

Output Format

- For each testcase output the number of gangs will be formed Solution:

Solution in C:

```
#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


void dfs(int k,int visited[],int matrix[1000][1000],int n){

    visited[k]=1;

    int i;

    for(i=1;i<=n;i++){

        if(matrix[k][i]==1 && visited[i]==0){

            dfs(i,visited,matrix,n);

        }

    }

}

int calculate(int matrix[1000][1000],int n,int visited[]){

    int c=0,z;

    for(z=1;z<=n;z++){

        if(visited[z]==0){

            c++;

            dfs(z,visited,matrix,n);

        }

    }

    return c;
}


int main()

{

    // Write your code here

    int t;

    scanf("%d",&t);

    while(t--){

        int n,m,i;

        scanf("%d %d",&n,&m);// n students and m edges as frienships

        int matrix[1000][1000]={0};

        while(m--){

            int a,b;

            scanf("%d %d",&a,&b);

            matrix[a][b]=1;

            matrix[b][a]=1;

            //adding edge between two people

        }

        int visited[n+1];

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

            visited[i]=0;

        }

        int count=calculate(matrix,n,visited);

        printf("%d\n",count);


    }


    return 0;

}
```

Solution in C++:

```
#include <bits/stdc++.h>

using namespace std;

int findparent(int n,unordered_map<int,int> &mp){

    while(mp[n]!=0)

        n=mp[n];

    return n;

}

int main(){

    int t;

    cin>>t;

    while(t--){

        int m,n,x,y,count=0;

        cin>>m>>n;

        unordered_map<int,int> mp,np;

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

            cin>>x>>y;

            int sp=findparent(x,mp);

            int dp=findparent(y,mp);

            if(sp!=dp)

                mp[dp]=sp;

        }

        for(auto i:mp)

            if(i.second!=0 && np[findparent(i.second,mp)]==0){

                np[i.first]=1;

                count++;

            }

        cout<<m-count<<endl;

    }

}
```

Solution in Python:

```
import math

import os

import random

import re

import sys


t=int(input())

while(t!=0):

n,m=input().split()

n,m=int(n),int(m)

input_list=[]

while(m!=0):

  l=input().split()

  input_list.append(l)

  m=m-1


new_list=range(m)

gang_list=[]

k=0

for i in input_list:

  for j in i:

    length=len(input_list)

    for z in range(k+1,length):

      if j in input_list[z]:

        a=input_list[z]

        del input_list[z]

        input_list[k].extend(a)

        break

  k+=1
```

```
no_of_gangs=len(input_list)


for i in range(1,no_of_gangs):
    input_list[0].extend(input_list[i])


set1=set(input_list[0])


set1=list(set1)
no_of_people_involved=len(set1)
total=n+no_of_gangs-no_of_people_involved
print(total)



t=t-1
```

Solution in java Using OOPS:

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.function.*;
import java.util.regex.*;
import java.util.stream.*;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;
```

```
class Graph{
    int nodes;
    public int[][] matrix;
    int[] visited;
    Graph(int n){
        this.matrix= new int[n+1][n+1];
        this.nodes=n;
        visited= new int[n+1];
    }
    public void addEdge(int d1,int d2) {
        this.matrix[d1][d2] = 1;
        this.matrix[d2][d1] = 1;

    }
    public void dfs(int k){
        this.visited[k]=1;
        for(int i=1;i<=this.nodes;i++){
            if(matrix[k][i]==1 && this.visited[i]==0){
                dfs(i);
            }
        }
    }
    public int  gangs(){
        int c=0;
        for(int z=1;z<=this.nodes;z++){
            if(visited[z]==0){
                c++;
                dfs(z);
            }
        }
        return c;
    }

}

class Solution {
public static void main(String[] args) {
    Scanner scan= new Scanner(System.in);
    int t=scan.nextInt();
    while(t-- >0){
        int n=scan.nextInt();
        int q=scan.nextInt();
        Graph g= new Graph(n);

        while(q-- >0){
            int a=scan.nextInt();
            int b=scan.nextInt();
            g.addEdge(a,b);

        }
        System.out.println(g.gangs());

    }
}

}
```

Solution in java without OOPS:

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.function.*;
import java.util.regex.*;
import java.util.stream.*;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;



public class Solution {
  public static void main(String[] args) {
      Scanner sc=new Scanner(System.in);
    int t=sc.nextInt();
    while(t-->0){
      int n=sc.nextInt();
      int m=sc.nextInt();
      int arr[][]=new int[n][n];
      for(int i=0;i<m;i++){
        int p=sc.nextInt();
        int q=sc.nextInt();
```

```
                arr[p-1][q-1]=1;
                arr[q-1][p-1]=1;
            }
            System.out.println(start(arr));
        }

    }

  public static int start(int[][] M){
    if (M == null || M.length == 0 || M[0].length == 0) return 0;
    int n = M.length;
    int numCircles = 0;
    boolean[] visited = new boolean[n];
    for (int i = 0; i < n; i++) {
      if (!visited[i]) {
        dfs(M, i, visited, n);
        numCircles++;
      }
    }
    return numCircles;
  }

  private static void dfs(int[][] M, int i, boolean[] visited, int n){
    for (int j = 0; j < n; j++) {
      if (M[i][j] == 1 && !visited[j]) {
        visited[j] = true;
        dfs(M, j, visited, n);
      }
  }
      }
}
```