Gopal's Startup

Problem : Gopal started a new social networking startup called Gwitter. In Gwitter people can follow other people. One more thing about Gwitter is if A follows B and B follows C then A follows C indirectly.

Gopal wants to write an algorithm to divide people among groups. A group is said to be a bunch of people who follow every other person either directly or indirectly.

Now he wants to know the number of people in the largest group. Can you find out this for him?

Difficulty : Medium

Pre-Requisites : Graph, DFS

Explaination : The problem is to find the number of people in the largest strongly connected component. A strongly connected component is a subset s of the vertices of G such that every node u in s is reachable from every other node v in s.

We can use Kosaraju algorithm to find the strongly connected components it is a 2-pass dfs algorithm.
Algorithm :

1. DFS is applied on the given graph and nodes are put in to the stack according to their finishing time.
2. Graph is reversed
3. Nodes are popped out of the stack and DFS is run again and number of times the DFS is applied gives the number of components

More Explaination :- http://www.geeksforgeeks.org/strongly-connected-components/

Complexity : O(|V| + |E|) if the graph is represented as adjacency list
O(|V|^2) if the graph is represented as adjacency matrix

Code (C++) :

```
#include<bits/stdc++.h>
using namespace std;

#define SIZE 1002
stack<int> s;
int compSize = 0;
```

```
int mx = 0;

int dfs(int adj[][SIZE], int *vis, int u, int n){
        vis[u]=1;
        for(int i=0; i<n; i++)
                if(adj[u][i] && !vis[i])
                        dfs(adj, vis, i, n);
        s.push(u);
        return 0;
}

void dfs2(int adj[][SIZE], int *vis, int u, int n){
        vis[u]=1;
        compSize++;
        for(int j=0; j<n; j++)
                if(!vis[j] && adj[u][j])
                        dfs2(adj, vis, j, n);
}

int kosaraju(int adj[][SIZE], int *vis, int n){
        int components=0;

        //First pass
        for(int i=0; i<n; i++)
                if(!vis[i])
                        dfs(adj, vis, i, n);

        //Making visited again to zero
        for(int i=0; i<n; i++)
                vis[i]=0;

       //Getting transpose of the graph
       for(int i=0; i<n; i++){
          for(int j=i+1; j<n; j++){
             int t = adj[i][j];
             adj[i][j] = adj[j][i];
             adj[j][i] = t;
          }
       }

        //Second pass
        while(!s.empty()){
```

```
            int u=s.top();
            s.pop();
            if(!vis[u]){
                compSize = 0;
                dfs2(adj, vis, u, n);
                components++;
                mx = max(mx, compSize);
            }
        }
        return mx;
}

void solve(){
        int i, j, x, y, n, m, ans;
        int adj[SIZE][SIZE]={0};
        int vis[SIZE]={0};
        compSize = 0;
        mx = 0;
        while(!s.empty())
            s.pop();
        scanf("%d%d", &n, &m);
        for(i=0; i<m; i++){
            scanf("%d%d", &x, &y);
            adj[x-1][y-1] = 1;
        }
        ans = kosaraju(adj, vis, n);
        printf("%d\n", ans);
}

int main(){
        int t;
        scanf("%d", &t);
        while(t--)
            solve();
        return 0;
}
```