```cpp
/*** Strongly connected Component:
A directed graph is strongly connected if there is a path between all pairs of vertices.
A strongly connected component(SCC) of a directed graph is a maximal strongly connected subgraph.
Tarjan Algorithm is based on following facts:
1. DFS search produces a DFS tree/forest
2. Strongly Connected Components form subtrees of the DFS tree.
3. If we can find head of such subtrees, we can print/store all the nodes in that subtree
   (including head) and that will be one SCC.
4. There is no back edge from one SCC to another (There can be cross edges, but cross edges
will not be used while processing the graph).
To find head of a SCC, we calculate desc and low array (as done for articulation point,
bridge, biconnected component). As discussed in the previous posts, low[u] indicates
earliest visited vertex (the vertex with minimum discovery time) that can be reached from
subtree rooted with u. A node u is head if disc[u] = low[u].
***/
#define mx 100005
vector<int>ed[mx+5],re[mx+5],R[mx+5];
bool vis[mx+5];
stack<int>st;
void dfs1(int u){
    vis[u]=true;
    for(int i=0; i<ed[u].size(); i++){
        int v = ed[u][i];
        if(!vis[v]) dfs1(v);
    }
    st.push(u);
}
void dfs2(int u,int gr){
    vis[u]=true;
    for(int i=0; i<re[u].size(); i++){
        int v = re[u][i];
        if(!vis[v]) dfs2(v,gr);
    }
    R[gr].push_back(u);
}
int main(){
    int tt; scanf("%d",&tt);
    for(int ks=1; ks<=tt; ks++){
        int n,m; scanf("%d%d",&n,&m);
        for(int i=1; i<=m; i++){
            int u,v; scanf("%d%d",&u,&v);
            ed[u].push_back(v); re[v].push_back(u);
        }
        memset(vis,false,sizeof(vis));
        for(int i=1; i<=n; i++){
            if(!vis[i]) dfs1(i);
        }
        memset(vis,false,sizeof(vis));

        int gr=0;
        while(!st.empty()){
            int u = st.top(); st.pop();
            if(vis[u])continue;
            gr++;
            dfs2(u,gr);
            sort(R[gr].begin(), R[gr].end());
        }
        for(int i=1; i<=gr; i++){
            printf("Group %d:",i);
            for(int j=0; j<R[i].size(); j++){
                int u = R[i][j]; printf(" %d",u);
            }
            printf("\n");
        }
        for(int i=1; i<=gr; i++)R[i].clear();
        for(int i=1; i<=n; i++) ed[i].clear();
        for(int i=1; i<=n; i++) re[i].clear();
    }
}
```