```java
class TarjansSCC
{
    public static void main (String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();

        while(t-- > 0)
        {
            // arraylist of arraylist to represent graph
            ArrayList<ArrayList<Integer>> adj = new ArrayList<>();

            int V = Integer.parseInt(sc.next());
            int E = Integer.parseInt(sc.next());

            for(int i =0; i < V; i++)
                adj.add(i, new ArrayList<Integer>());

            for(int i = 1; i <= E; i++)
            {    int u = Integer.parseInt(sc.next());
                 int v = Integer.parseInt(sc.next());

                 // adding directed edgese between
                 // vertex 'u' and 'v'
                 adj.get(u).add(v);
            }

            Solution ob = new Solution();
            ArrayList<ArrayList<Integer>> ptr = ob.tarjans(V, adj);

            for(int i=0; i<ptr.size(); i++)
            {
                for(int j=0; j<ptr.get(i).size(); j++)
                {
                    if(j==ptr.get(i).size()-1)
                        System.out.print(ptr.get(i).get(j));
                    else
                        System.out.print(ptr.get(i).get(j) + " ");
                }
                System.out.print(",");
            }
            System.out.println();
        }
    }
}// } Driver Code Ends


//User function Template for Java

class Solution
{
    static int id =1; // for assigning ids

    public ArrayList<ArrayList<Integer>> tarjans(int V, ArrayList<ArrayList<Integer>> adj)
{

        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        boolean stack[] = new boolean[V]; //O(1) to check wheater its on stack or not
traveresing original stack
        int [] ids = new int[V]; //store the ids
        int[] lowValues = new int[V];
        Stack<Integer> stackQue = new Stack<Integer>();//to add vertice while we recurse

        for(int i=0;i<V;i++){
            if(ids[i]==0){ // if not visited visit
                dfs(i,adj,stack,ids,lowValues,stackQue,result);
            }
        }
        //just sorting the result in order, but not neccessary
       Collections.sort(result,new Comparator<ArrayList<Integer>>(){
           public int compare(ArrayList<Integer> list1,ArrayList<Integer> list2){
               return list1.get(0)-list2.get(0);
           }
       });
        return result;

    }

    public static void dfs(int vertex,ArrayList<ArrayList<Integer>> adj, boolean stack[],int[]ids,int[] lowValues,Stack<Integer> stackQue,ArrayList<ArrayList<Integer>> result){

        ids[vertex] = id;
        lowValues[vertex]=id++;
        stack[vertex]=true;
        stackQue.push(vertex);

        ArrayList<Integer> cl = adj.get(vertex);
        int len = cl.size();
        for(int i=0;i<len;i++){
            int nextNode = cl.get(i);
            if(ids[nextNode]==0){ // if nextNode not vistied, then visit
                dfs(nextNode,adj,stack,ids,lowValues,stackQue,result);
            }
            if(stack[nextNode]){ //if we encountered a node which is already visted,then if its on stack only
                lowValues[vertex]=Math.min(lowValues[vertex],lowValues[nextNode]);
            }//this implementations avoid cross edges, i.e leaking of lowValue. A corss edge is edge which does not point to its ancestor.

        }
          //when lowvalue and id are same, it means we had looped back to original positon and its a SCC
        if(lowValues[vertex]==ids[vertex]){
            ArrayList<Integer> list = new ArrayList<Integer>();
            while(true){
                int node = stackQue.pop();
                list.add(node);
                stack[node]=false;
                if(node==vertex){
                    break;}
            }
            //add these components to a list.
            Collections.sort(list);
            result.add(list);
        }
}




}
```