```java
class BridgesAndArticulationPoints {
   static int id=0;
    public static class Edge{
        int src;
        int dst;
        Edge(int src,int dst){
            this.src = src;
            this.dst = dst;
        }
    }
    public static void main(String args[] ) throws Exception {

        //Scanner
        Scanner sc = new Scanner(System.in);
        int vertices  = sc.nextInt();
        int edges = sc.nextInt();
        ArrayList<ArrayList<Integer>> graph = new ArrayList<ArrayList<Integer>>();
        for(int i=0;i<vertices;i++){
            graph.add(new ArrayList<Integer>());
        }
        for(int i=0;i<edges;i++){
            int src = sc.nextInt();
            int dst = sc.nextInt();
            graph.get(src).add(dst);
        }
          //passing a graph to find brides and articulations
         findBridesAndAriculations(graph,vertices);

    }

    public static void findBridesAndAriculations(ArrayList<ArrayList<Integer>> graph,int V)
{
        int ids[] = new int[V];
        int lowValues[] = new int[V];
        boolean []visited = new boolean[V];
        boolean[] arti = new boolean[V]; // to stor articulations
        ArrayList<Edge> edges = new ArrayList<Edge>(); // to store edges
        for(int i=0;i<V;i++){
            if(!visited[i]){
        // children only need for root to verify wheather it is a articulation pointyou
can't just get size of list and determine the size,it abou
                int children = dfs(i,graph,-1,visited,ids,lowValues,arti,edges);
                // you can't just get size of list and determine the size,it about the
traversing.
                arti[i]=(children>1);
            }
        }
        ArrayList<Integer> articulates = new ArrayList<Integer>();
        for(int i=0;i<V;i++){
            if(arti[i]){
                articulates.add(i);
            }
        }
        Collections.sort(articulates);
        Collections.sort(edges,new Comparator<Edge>(){
            public int compare(Edge e1,Edge e2){
                if(e1.src!=e2.src){
                    return e1.src-e2.src;
                }
                return e1.dst-e2.dst;
            }
```

```
59          });
60          System.out.println(articulates.size());
61          for(int node : articulates){
62              System.out.print(node+" ");
63          }
64          System.out.println();
65          System.out.println(edges.size());
66          for(Edge edge : edges){
67              System.out.println(edge.src + " " + edge.dst);
68          }
69      }
70
71      public static int dfs(int vertex,ArrayList<ArrayList<Integer>> graph,int
parent,boolean[] visited,int[] ids,int[] lowValues,boolean[] arti,ArrayList<Edge> edges){
72       visited[vertex]=true;
73       ids[vertex]=id;
74       lowValues[vertex]=id++;
75       int children=0;
76       ArrayList<Integer> cl = graph.get(vertex);
77       int len = cl.size();
78       for(int i=0;i<len;i++){
79           int nextNode = cl.get(i);
80           if(!visited[nextNode]){
81               children++; // we need these children only for root, though its waste of
tracking for other nodse
82               dfs(nextNode,graph,vertex,visited,ids,lowValues,arti,edges);
83               //when recusion is returned, you wil check wheater it had got lowerValue
84               lowValues[vertex]=Math.min(lowValues[vertex],lowValues[nextNode]);
85           }
86           //if encountered already visited node then min with its lowvalue
87           lowValues[vertex]=Math.min(lowValues[vertex],ids[nextNode]);
88
89           if(lowValues[nextNode]>=ids[vertex]){
90               //its articulaton through cycle "=" & brige ">" its sufficient check for
intermediate
91               //for root this condition would not be sufficient,but are again cheking for
root at end
92               arti[vertex]=true;
93           }
94            if(lowValues[nextNode]>ids[vertex]){
95               //its a bride check because it couldn't reach to the ancestor on its own,it
needs this bridge
96               if(nextNode>vertex){
97               edges.add(new Edge(vertex,nextNode));
98               }
99               else{
100               edges.add(new Edge(nextNode,vertex));
101              }
102          }
103      }
104      return children;
105     }
106 }
```