```
import java.util.ArrayList;
class Striver{
    //****************************************Breadth First
Search************************************************* */
 class Solution{
    public ArrayList<Integer> bfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
    ArrayList<Integer> bfs=new ArrayList<>();
    boolean vis[]=new boolean[V];
    for(int i=0;i<V;i++){
        vis[i]=false;
    }
    Queue<Integer> q=new LinkedList<>();
    q.add(src);
    vis[src]=true;
    while(!q.isEmpty()){
        Integer node=q.poll();
        bfs.add(node);
        for(Integer it: adj.get(node)){
            if(vis[it]==false){
                vis[it]=true;
                q.add(it);
            }
        }
    }
    return bfs;
}
}
    //****************************************Depth First
Search************************************************* */
class Solution{
    public void dfs(int node,boolean vis[],ArrayList<ArrayList<Integer>> adj,
ArrayList<Integer> dfs){
        vis[node]=true;
        dfs.add(node);
        for(Integer it: adj.get(node))
        {
            if(vis[it]==false){
                dfs(it,vis,adj,dfs);
            }
        }
    }
    public ArrayList<Integer> dfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
    ArrayList<Integer> dfs=new ArrayList<>();
    boolean vis[]=new boolean[V];
    for(int i=0;i<V;i++){
        vis[i]=false;
    }
    visited[src]=true;
    dfs(src,vis,adj,dfs);
    return dfs;
    }
    }
    //***************************************Detect Cycle in Undirected Graph
BFS*********************************************** */
class Solution{
        public boolean checkforCycle(int src,int V,boolean vis[],
ArrayList<ArrayList<Integer>> adj,boolean vis[]){
            vis[src]=true;
            Queue<Pair> q=new LinkedList<>();
            q.add(new Pair(src,-1));
            while(!q.isEmpty()){
                int node=q.peek().first;
                int parent=q.peek().second;
                q.remove();
                for(Integer adjacentNode: adj.get(node))
                {
                    if(vis[adjacentNode]==false){
                        vis[adjacentNode]=true;
                        q.add(new Pair(adjacentNode,node));
                    }
                    else if(parent!=adjacentNode){
                        return true;
                    }
                }
            }
            return false;
        }
        public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj){
        boolean vis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        for(int i=0;i<V;i++){
            if(vis[i]==false){
                if(checkCycle(i,V,adj,vis)==true)
                return true;
            }
        }
        return false;
        }
}
    //***************************************Detect Cycle in Undirected Graph
DFS*********************************************** */
class Solution{
        public boolean dfs(int node,int parent,boolean vis[],ArrayList<ArrayList<Integer>>
adj){
            vis[src]=true;
            for(Integer adjacentNode : adj.get(node)){
                if(dfs(adjacentNode,node,vis,adj)==true){
                    return true;
                }
                else if((adjacentNode!=parent)){
                    return true;
                }
            }
            return false;
        }
        public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj){
        boolean vis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        for(int i=0;i<V;i++){
            if(vis[i]==false){
                if(dfs(i,-1,adj,vis)==true)
                return true;
            }
        }
        return false;
        }
}
    //***************************************Detect Cycle in Directed Graph
DFS******************************************** */
class Solution{
        public boolean checkCycle(int node,boolean vis[],ArrayList<ArrayList<Integer>>
adj,boolean vis[],boolean dfsvis[]){
            vis[node]=true;
            dfsvis[node]=true;
            for(Integer it: adj.get(node))
            {
                if(vis[it]==false){
                    if(checkCycle(it,vis,adj,vis,dfsvis)==true){
                         return true;
                }
                else if(dfsvis[it]==true){
                    return true;
                }
            }
            }
        dfsvis[node]=false;
        return false;
        }
        public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj){
        ArrayList<Integer> dfs=new ArrayList<>();
        boolean vis[]=new boolean[V];
        boolean dfsvis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
            dfsvis[i]=false;
        }
        for(int i=0;i<V;i++){
            if(vis[i]==false){
                if(checkCycle(i,adj,vis,dfsvis)==true)
                return true;
            }
        }
        return false;
        }
}
    //**************************************Detect Cycle in Directed Graph
BFS****************************************** */
class Solution{
    public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj){
        int indegree[]=new int[V];
        for(int i=0;i<V;i++){
            for(Integer it:adj.get(i)){
                indegree[it]++;
            }
        }
        Queue<Integer> q=new LinkedList<>();
        for(int i=0;i<V;i++){
            if(indegree[i]==0){
                q.add(i);
            }
        }
        int cnt=0;
        while(!q.isEmpty()){
            int node=q.peek();
            q.remove();
            cnt++;
            for(Integer it: adj.get(node)){
                indegree[it]--;
                if(indegree[it]==0){q.add(it);}
            }
        }
        if(cnt==V){
            return false;
        }
        return true;


    }
}
    //**************************************Topological Sort
BFS****************************************** */
```

```
class Solution{
        public boolean topoSort(int V,ArrayList<ArrayList<Integer>> adj){
            int indegree[]=new int[V];
            for(int i=0;i<V;i++){
                for(Integer it:adj.get(i)){
                    indegree[it]++;
                }
            }
            Queue<Integer> q=new LinkedList<>();
            for(int i=0;i<V;i++){
                if(indegree[i]==0){
                    q.add(i);
                }
            }
            int i=0;
            int topo[]=new int[V];
            while(!q.isEmpty()){
                int node=q.peek();
                q.remove();
                topo[i++]=node;
                for(Integer it: adj.get(node)){
                    indegree[it]--;
                    if(indegree[it]==0){q.add(it);}
                }
            }
            return topo;
        }
    }
    //***************************************Topological Sort
DFS******************************************* */
class Solution{
        public void dfs(int node,boolean vis[],ArrayList<ArrayList<Integer>> adj,
Stack<Integer> st){
            vis[node]=true;
            for(Integer it: adj.get(node))
            {
                if(vis[it]==false){
                    dfs(it,vis,adj,st);
                }
            }
            st.push(node);
        }
        public ArrayList<Integer> dfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
        boolean vis[]=new boolean[V];
        Stack<Integer> st=new Stack<>();
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        for(int i=0;i<V;i++){
            if(vis[i]==false){
                dfs(i,vis,adj,st);
            }
        }
        int ans[]=new int[V];
        int i=0;
        while(!st.isEmpty()){
            ans[i++]=st.peek();
            st.pop();
        }
        return ans;
    }
}
}
```