Cycle in graph

Undirected : DFS, BFS, DSU(Disjoint Set Union)
Directed : DFS, BFS, Topological sort(Kahn's Algo)

Topological Sort

DFS Approach

Idea

Node that comes at last must be present at last. Hence, the idea is to store last visited at bottom. Thus, *Stack* comes into picture.

Algorithm

- Make visited *Array* to tackle both disconnected & visited Nodes.
- Call DFS on each unvisited node
    - Call DFS on unvisited Neighbours
    - After making all calls to Neighbours Store current Node in a Stack.
- Finally, pop all the elements of stack into resultant *Vector*

Source Code

```
static void topoUtil(ArrayList<ArrayList<Integer>> adj,boolean []visited, Stack<Integer> s,int curr){
    visited[curr] = true;
    for(int neighbour : adj.get(curr)){
        if(!visited[neighbour])
            topoUtil(adj,visited,s,neighbour);
    }
    s.add(curr);
}
static int[] topoSort(int V, ArrayList<ArrayList<Integer>> adj){
    boolean visited[] = new boolean[V];
    Stack<Integer> s = new Stack<>();
    for(int i=0;i<V;i++){
        if(!visited[i])
            topoUtil(adj,visited,s,i);
    }
    int ans[] = new int[s.size()];
    for(int i = 0;!s.isEmpty();i++)
        ans[i] = s.pop();
    return ans;
}
```

BFS Approach || Kahn's Algo

Idea

The idea is that all nodes which will be at starting will have indegree 0.

Algorithm

- Store Indegree of all nodes in *Array*.
- Push nodes in *Queue* whose indegree == 0.
- Now, For each node in *Queue*.
    - Pop the current node & Store into resultant *Array*
    - Remove indegree count of all neighbours of current node.
    - If neighbours indegree becomes 0 the push into queue.
- Finally, return resultant *Array.*

Source Code

```
static int[] topoSort(int V, ArrayList<ArrayList<Integer>> adj){
    int []indeg = new int[V];
    for(int i = 0 ; i<V ; i++){
        for(int it : adj.get(i)){
            indeg[it]++;
        }
    }

    Queue<Integer> q = new LinkedList<>();
    for(int i = 0 ;i<V ; i++){
        if(indeg[i] == 0)
            q.add(i);
    }
    int topo[] = new int[V];
    int i = 0;
    while(!q.isEmpty()){
        int node = q.poll();
        topo[i++] = node;
        for(int it : adj.get(node)){
            indeg[it]--;

            if(indeg[it] == 0)
            q.add(it);
        }
    }
    return topo;
}
```