Topological Sort(Kahn's Algorithm):

```
class Solution
{
	public:
	//Function to return list containing vertices in Topological order.
	vector<int> topoSort(int V, vector<int> adj[])
	{
	    vector<int>ans;
	    vector<int>indegree(V,0);
        for(int i=0;i<V;i++){
            for(auto it:adj[i]){
                indegree[it]++;
            }
        }
        queue<int>q;
        for(int i=0;i<V;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }
        while(!q.empty()){
            int node=q.front();
            q.pop();

            ans.push_back(node);
            for(auto it:adj[node]){
                indegree[it]--;
                if(indegree[it]==0){
                    q.push(it);
                }
            }
        }
        return ans;
	}
};
```

Detect a cycle in a directed graph:

```
class Solution {
  public:
    // Function to detect cycle in a directed graph.
    bool isCyclic(int V, vector<int> adj[]) {
         vector<int>ans;
	    vector<int>indegree(V,0);
        for(int i=0;i<V;i++){
            for(auto it:adj[i]){
                indegree[it]++;
            }
        }
        queue<int>q;
        for(int i=0;i<V;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }
        int cnt=0;
        while(!q.empty()){
            int node=q.front();
            q.pop();
            cnt++;

            ans.push_back(node);
            for(auto it:adj[node]){
                indegree[it]--;
                if(indegree[it]==0){
                    q.push(it);
                }
            }
        }
        // for(int i=0;i<V;i++){
        //     if(indegree[i]>0){
        //         return true;
        //     }
        // }
        // return false;

        //or


         if(cnt==V){
           return false;
       }
       else{
           return true;
       }
    }
};
```

Course Schedule-I

```
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& nums) {
        int cnt=0;
        vector<int>adjList[numCourses];
        vector<int>indegree(numCourses,0);
        for(int i=0;i<nums.size();i++){
            int v=nums[i][0];//1
            int u=nums[i][1];//0
            indegree[u]++;
            adjList[v].push_back(u);
        }

        queue<int>q;
        for(int i=0;i<indegree.size();i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }
        while(!q.empty()){
            int x=q.front();
            q.pop();

            for(int it:adjList[x]){
                indegree[it]--;
                if(indegree[it]==0){
                    q.push(it);
                }
            }
        }
        for(int i=0;i<indegree.size();i++){
            if(indegree[i]>0){
                return false;
            }
        }
        return true;
    }
};
```

Course Schedule-II:

```
vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {
    vector<int>adj[numCourses];
    // vector<int>vis(numCourses,0);
    vector<int>indegree(numCourses,0);
    // vector<int>pathVis(numCourses,0);
    vector<int>v;
    for(auto it:prerequisites){
      int u=it[0];
    int v=it[1];
    adj[u].push_back(v);
    indegree[v]++;
}
int cnt=0;
queue<int>q;
for(int i=0;i<numCourses;i++){
    if(indegree[i]==0){
        q.push(i);
        v.push_back(i);
        cnt++;
    }
}
while(!q.empty()){
    int ele=q.front();
    q.pop();


    for(auto it:adj[ele]){
        indegree[it]--;
        if(indegree[it]==0){
            cnt++;
            q.push(it);
            v.push_back(it);
        }
    }
}
if(cnt!=numCourses){
    vector<int>ds;
    return ds;
}
else{
    reverse(v.begin(),v.end());
    return v;
}
```