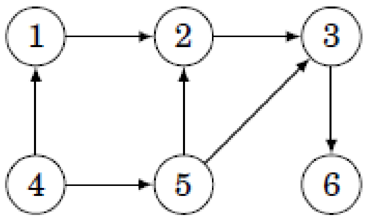
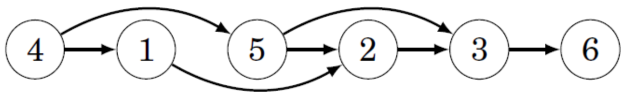
Topological Sorting

A **topological sort** is an ordering of the nodes of a directed graph such that if there is a path from node *a* to node *b* , then node *a* appears before node *b* in the ordering.

For example, for the graph,



One Topological Sort is [4–>1->5->2->3->6].



An acyclic graph always has a topological sort. **However, if the graph contains a cycle, it is not possible to form a topological sort, because no node of the cycle can appear before the other nodes of the cycle in the ordering.** It turns out that depth-first search can be used to both check if a directed graph contains a cycle and, if it does not contain a cycle, to construct a topological sort.

DFS

Algorithm

The idea is to go through the nodes of the graph and always begin a depth-first search at the current node if it hasn't been processed yet. When the node don't have any child or don't have any further paths it is called as processed node.

> **NOTE :** A topological sort isn't unique, and there can be several topological sorts for a graph.

The complete implementation is (*Note : The code treats the input graph as acyclic*) :

```
#include<bits/stdc++.h>
using namespace std;

void dfsHelper(vector<vector<int>>adj,vector<bool>&visited,int i,stack<int> &st)
{
    if(visited[i])return;
    visited[i]=1;
    cout<<i<<" ";
    for(auto u:adj[i]){
        dfsHelper(adj,visited,u,st);
    }
    st.push(i);
}

void dfs(vector<vector<int>>adj,int n){
    vector<bool>visited(n,0);
    stack<int> st;
    for(int i=1;i<n;i++){
        dfsHelper(adj,visited,i,st);
    }
    cout<<endl;
    while(!st.empty()){
        cout<<st.top()<<" ";
        st.pop();
    }
}

int main(){
    int n;
    cin>>n;
    vector<vector<int>>adj(n+1);
    int edge;
    cin>>edge;
    for(int i=0;i<edge;i++){
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
    }
    dfs(adj,n+1);
}
```

BFS or Kahn's Algorithm

The approach is based on the below fact:
A DAG G has at least one vertex with in-degree 0 and one vertex with out-degree 0.

Algorithm

- Count the indegree of each nodes.
- Pick all the node with indegree as 0 and add them to queue and answer vector.
- Iterate till queue becomes empty .
  - Remove a node from the queue and relax the adjacent nodes.
  - If the indegree of relaxed node becomes 0, push them in queue and answer vector.

```
 while(!q.empty()){
        int front=q.front();
        q.pop();
        for(auto u:adj[front]){
            --indegree[u];
            if(indegree[u]==0){
                ans.push_back(u);
                q.push(u);
            }
        }
}
```

The complete implementation is :

```
#include<bits/stdc++.h>
using namespace std;

void bfs(vector<vector<int>>adj,int V){
    vector<int>indegree(V,0);
    vector<int>ans;
    queue<int>q;
    //Calculating Indegree of each Node
    for(int i=1;i<V;i++){
        for(auto u:adj[i]){
            indegree[u]++;
        }
    }


    //Nodes with indegree 0 comes at start
    for(int i=1;i<V;i++){
        if(indegree[i]==0){
            ans.push_back(i);
            q.push(i);
        }
    }

    //We go on relaxing the further nodes and push the nodes
    //with indegree 0 in ans .
    while(!q.empty()){
        int front=q.front();
        q.pop();
        for(auto u:adj[front]){
            --indegree[u];
            if(indegree[u]==0){
                ans.push_back(u);
                q.push(u);
            }
        }
    }

    for(int i=0;i<ans.size();i++){
        cout<<ans[i]<<" ";
    }
}

int main(){
    int n;
    cin>>n;
    vector<vector<int>>adj(n+1);
    int edge;
    cin>>edge;
    for(int i=0;i<edge;i++){
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
    }
    bfs(adj,n+1);
}
```

Application

- **Finding cycle in a graph** : A topological ordering is possible only for directed acyclic graph(DAG). For a cyclic graph topological ordering is not possible.
- **Finding Deadlock in OS** : Deadlock is a state in which a process in a waiting state and another waiting process is holding the demanded resource. If the wait-for graph has a cycle, then there is deadlock.
- **Dependency resolution** : Suppose, A class extends B class. Then B has a dependency on A, and A must be compiled before B.
- **Task Schedule or Instruction Schedule or Scheduling Problems**
    - Finding out possible sequence to finish task
    - Installing of packages in a Linux System,all dependencies are installed first in the order generated by topological sort.
    - Pre-requisite Problems

Complexity Analysis:

- **Time Complexity:** O(V+E).
  The above algorithm is simply DFS with an extra stack. So time complexity is the same as DFS which is.
- **Auxiliary space:** O(V).
  The extra space is needed for the stack.