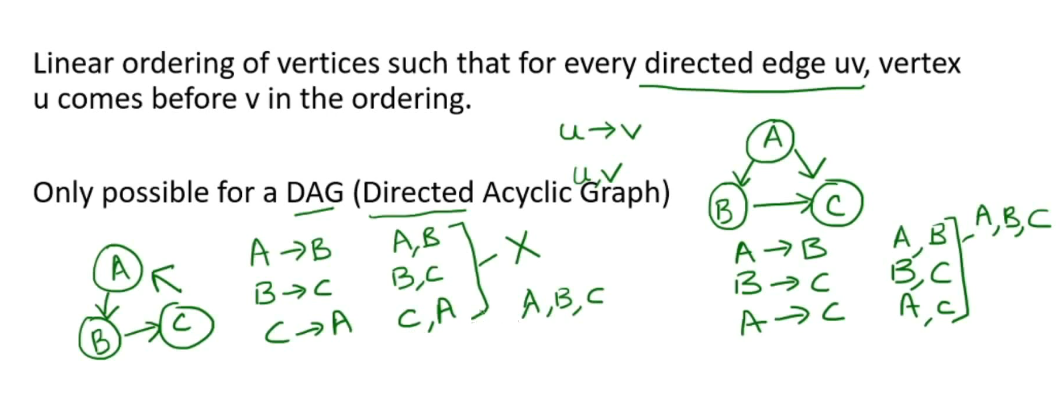
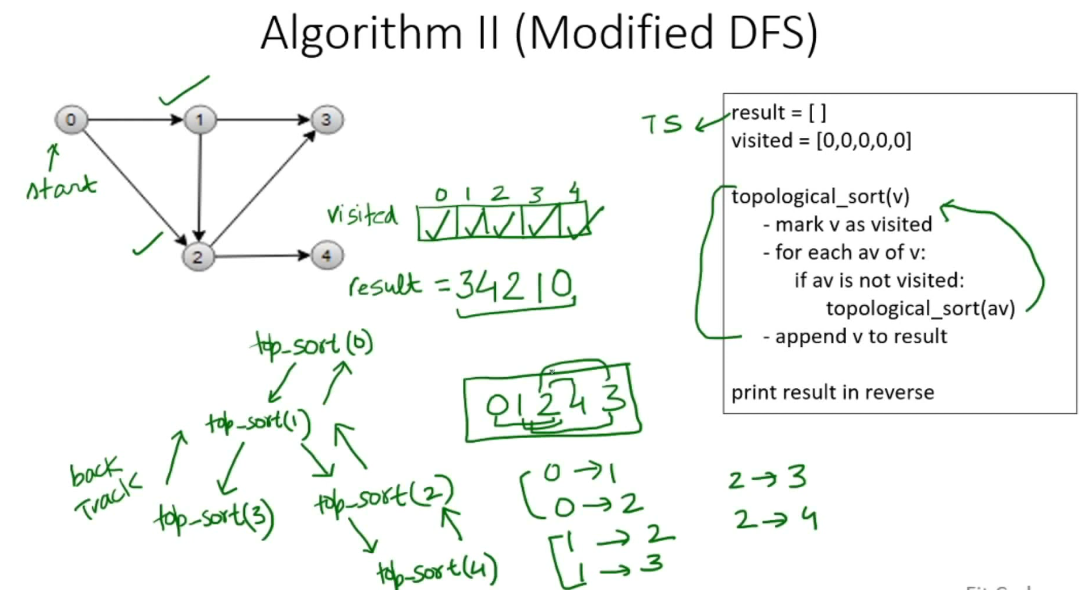
**Topological Sorting**

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.



**Algorithm:**

**Do the DFS traversal for the disconnected graph and in the last store the vertex in the stack and then print the stack data.**



**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.

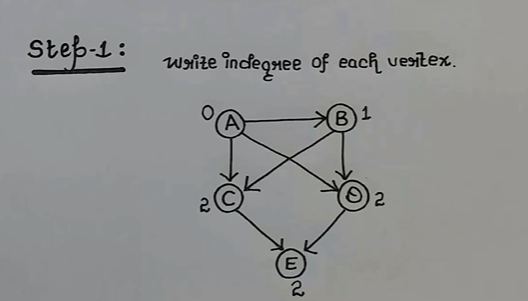
**Applications:**   
Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in make files, data serialization, and resolving symbol dependencies in linkers.

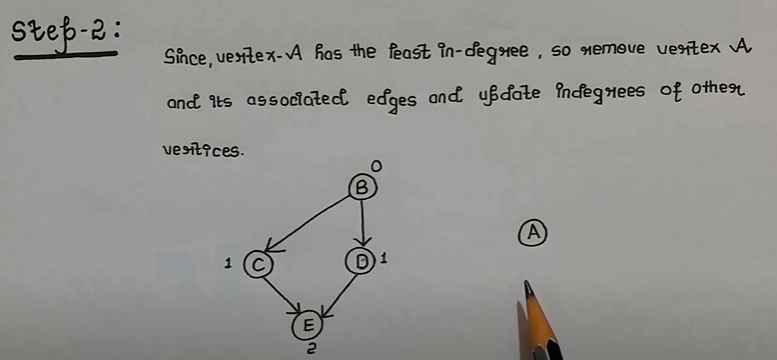
<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/topologicalsort/TopologicalSort.java>

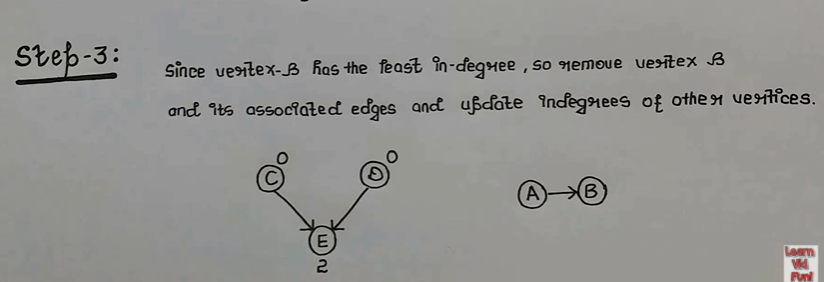
**All Topological Sorts of a Directed Acyclic Graph**

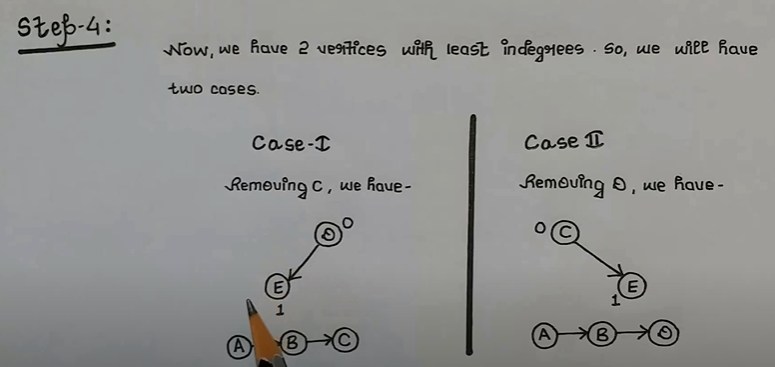
Topological sorting for **D**irected **A**cyclic **G**raph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.  
Given a DAG, print all topological sorts of the graph.

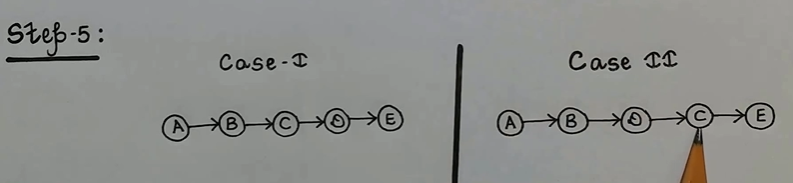
For example, consider the below graph.

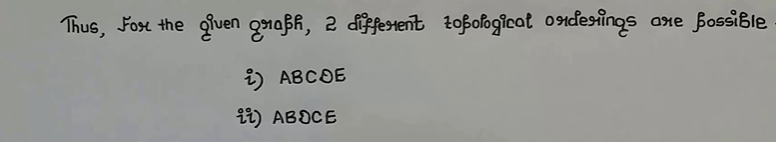










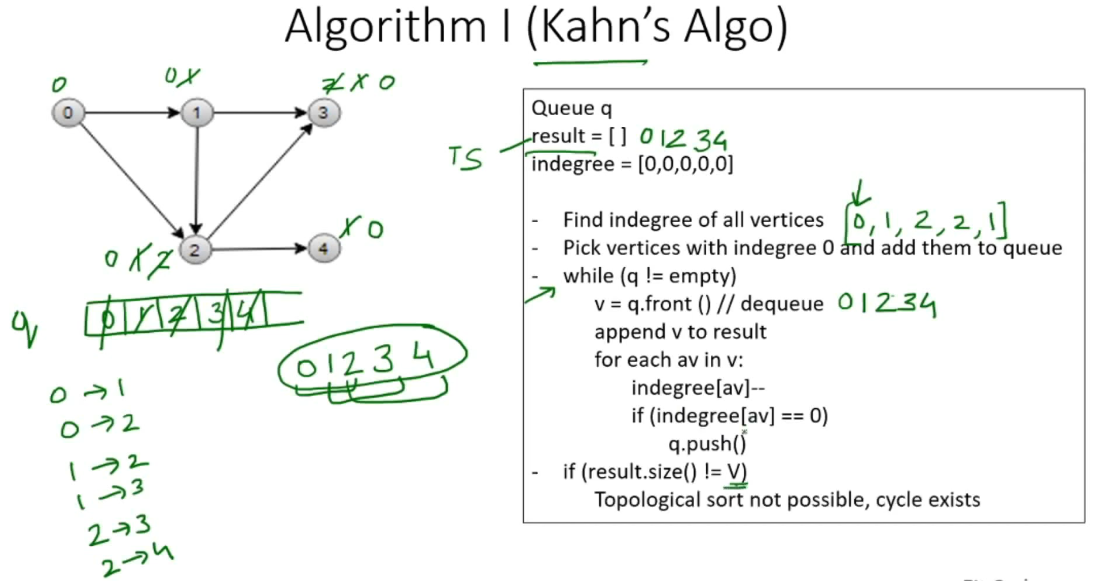


Below is implementation of above steps.

<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/topologicalsort/AllTopologicalSortOfDirectedAcyclicGraph.java>

**Kahn’s algorithm for Topological Sorting**

Topological sorting for **D**irected **A**cyclic **G**raph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.



**A DAG G has at least one vertex with in-degree 0 and one vertex with out-degree 0**.   
**Proof:** There’s a simple proof to the above fact is that a DAG does not contain a cycle which means that all paths will be of finite length. Now let S be the longest path from u(source) to v(destination). Since S is the longest path there can be no incoming edge to u and no outgoing edge from v, if this situation had occurred then S would not have been the longest path   
=> indegree(u) = 0 and outdegree(v) = 0

**Time Complexity:** The outer for loop will be executed V number of times and the inner for loop will be executed E number of times, Thus overall time complexity is O(V+E).  
The overall time complexity of the algorithm is O(V+E)

<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/topologicalsort/KahnAlgoForTopologicalSorting.java>

**Longest path between any pair of vertices**

We are given a map of cities connected with each other via cable lines such that there is no cycle between any two cities. We need to find the maximum length of cable between any two cities for given city map.

Input : n = 6

1 2 3 // Cable length from 1 to 2 (or 2 to 1) is 3

2 3 4

2 6 2

6 4 6

6 5 5

Output: maximum length of cable = 12

**Algorithm:**

We create undirected graph for given city map and do [DFS](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/) from every city to find maximum length of cable. While traversing, we look for total cable length to reach the current city and if it’s adjacent city is not visited then call [DFS](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/) for it but if all adjacent cities are visited for current node, then update the value of max\_length if previous value of max\_length is less than current value of total cable length.

<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/topologicalsort/LongestPathBetweenAnyPairOfVertices.java>

**Longest Path in a Directed Acyclic Graph**

Given a Weighted **D**irected **A**cyclic **G**raph (DAG) and a source vertex s in it, find the longest distances from s to all other vertices in the given graph.

The longest path problem for a general graph is not as easy as the shortest path problem because the longest path problem doesn’t have [optimal substructure property](https://www.geeksforgeeks.org/dynamic-programming-set-2-optimal-substructure-property/). In fact, [the Longest Path problem is NP-Hard for a general graph](http://en.wikipedia.org/wiki/Longest_path_problem). However, the longest path problem has a linear time solution for directed acyclic graphs. The idea is similar to [linear time solution for shortest path in a directed acyclic graph.](https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/), we use [Topological Sorting](https://www.geeksforgeeks.org/topological-sorting/).

We initialize distances to all vertices as minus infinite and distance to source as 0, then we find a [topological sorting](https://www.geeksforgeeks.org/topological-sorting/) of the graph. Topological Sorting of a graph represents a linear ordering of the graph. Once we have topological order (or linear representation), we one by one process all vertices in topological order. For every vertex being processed, we update distances of its adjacent using distance of current vertex.  
Following figure shows step by step process of finding longest paths.

LongestPath

Following is complete algorithm for finding longest distances.   
**1)** Initialize dist[] = {NINF, NINF, ….} and dist[s] = 0 where s is the source vertex. Here NINF means negative infinite.   
**2)** Create a topological order of all vertices.   
**3)** Do following for every vertex u in topological order.   
………..Do following for every adjacent vertex v of u   
………………if (dist[v] < dist[u] + weight(u, v))   
………………………dist[v] = dist[u] + weight(u, v)

<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/topologicalsort/LongestPathInDirectedAcyclicGraph.java>