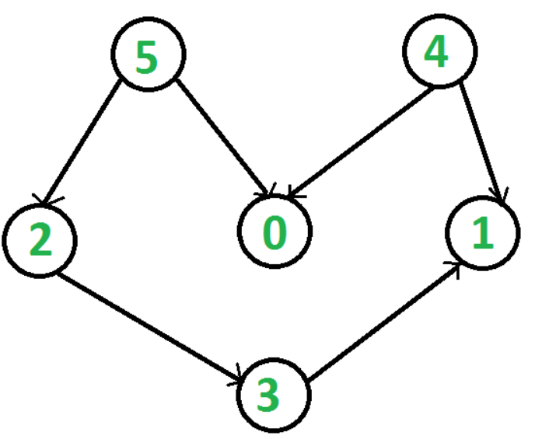
**Topological Sort**

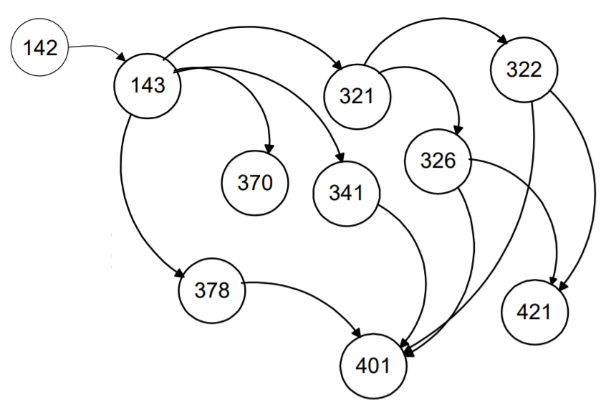
Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge (u,v), vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

For example, a topological sorting of the following graph is “5 4 2 3 1 0”. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is “4 5 2 3 1 0”. The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no in-coming edges).

**Topological Sorting vs Depth First Traversal (DFS):**

In [DFS](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/), we print a vertex and then recursively call DFS for its adjacent vertices. In topological sorting, we need to print a vertex before its adjacent vertices. For example, in the given graph, the vertex ‘5’ should be printed before vertex ‘0’, but unlike [DFS](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/), the vertex ‘4’ should also be printed before vertex ‘0’. So topological sorting is different from DFS. For example, a DFS of the shown graph is “5 2 3 1 0 4”, but it is not a topological sorting

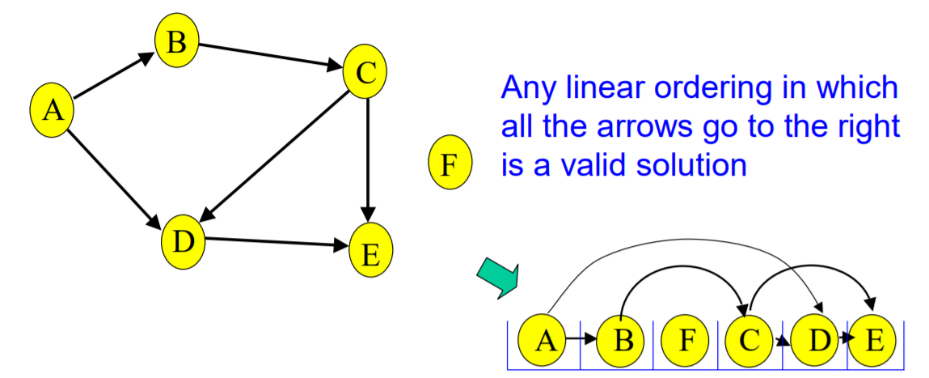
**Algorithm to find Topological Sorting:**We recommend to first see implementation of DFS .We can modify [DFS](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/)to find Topological Sorting of a graph. In [DFS](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/), we start from a vertex, we first print it and then recursively call DFS for its adjacent vertices. In topological sorting, we use a temporary stack. We don’t print the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack. Finally, print contents of stack. Note that a vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in stack.



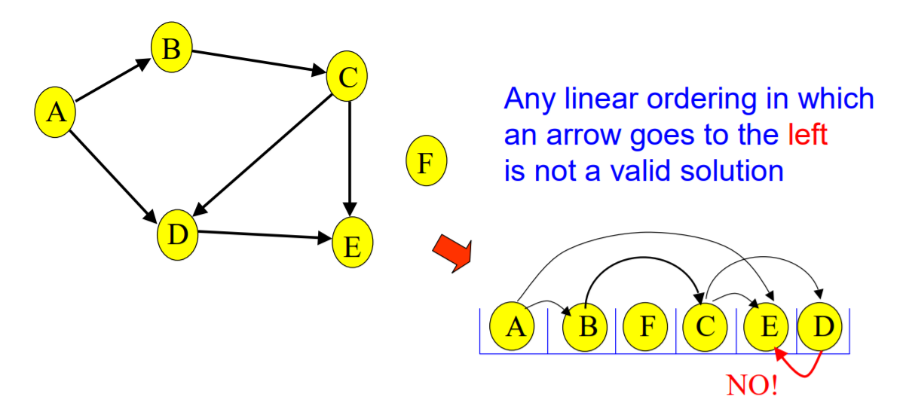
**Problem:** Find an order in which all these courses can be taken.   
Example: 142 143 378 370 321 341 322 326 421 401

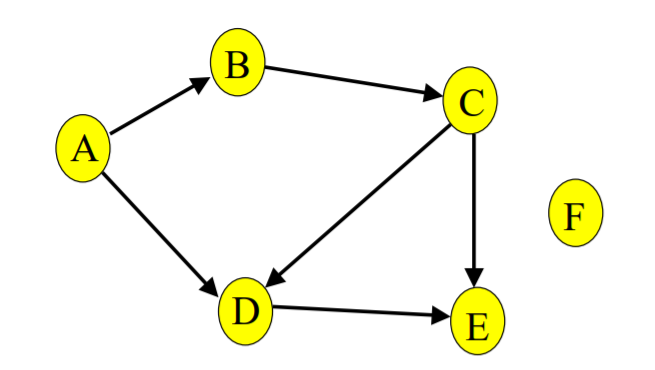
In order to take a course, you must take all of its prerequisites first

**Topo sort - Good example:**

Note that F can go anywhere in this list because it is not connected.

**Topo sort - Bad example:**



**Not all can be sorted:**

A directed graph with a cycle cannot be topologically sorted.

Given a digraph G = (V, E), find a linear ordering of its vertices such that: for any edge (v, w) in E, v precedes w in the ordering

**Cycles**

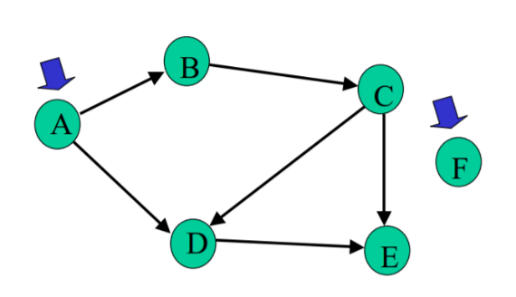
• Given a digraph G = (V,E), a cycle is a sequence of vertices v1,v2, …,vk such that

› k < 1

› v1 = vk

› (vi,vi+1) in E for 1 < i < k.

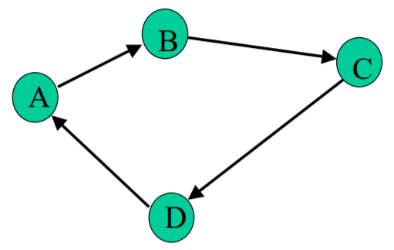
• G is acyclic if it has no cycles.

**Topo sort algorithm**

**Step 1:**

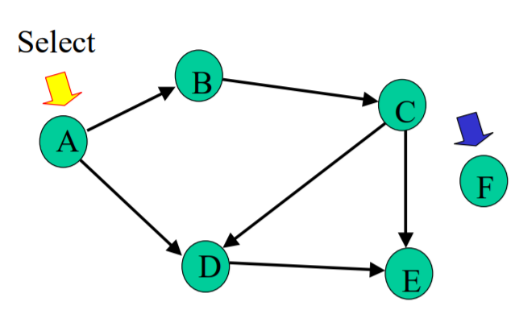
Identify vertices that have no incoming edges.

The (in-degree) of these vertices is zero



Identify vertices that have no incoming edges.

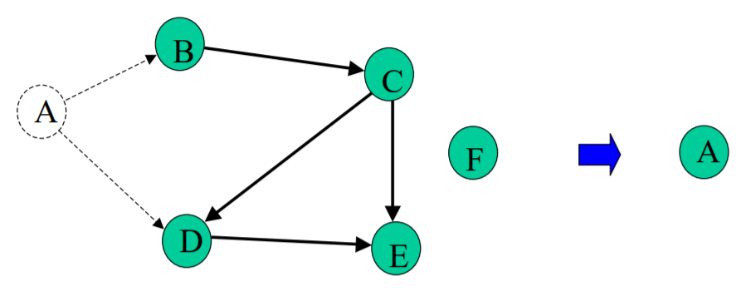
If no such vertices, graph has only cycle(s) (cyclic graph) Topological sort not possible – Halt.

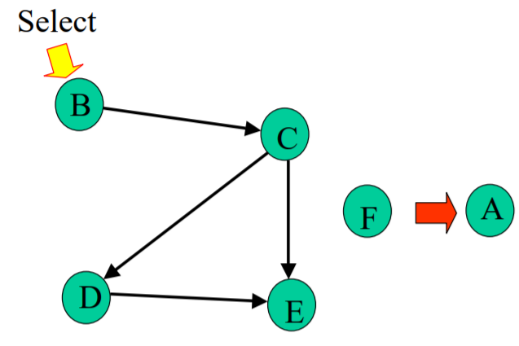


Identify vertices that have no incoming edges.

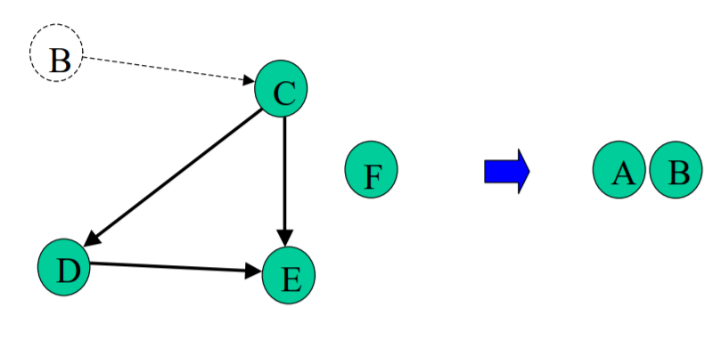
Select one such vertex

**Step 2:**

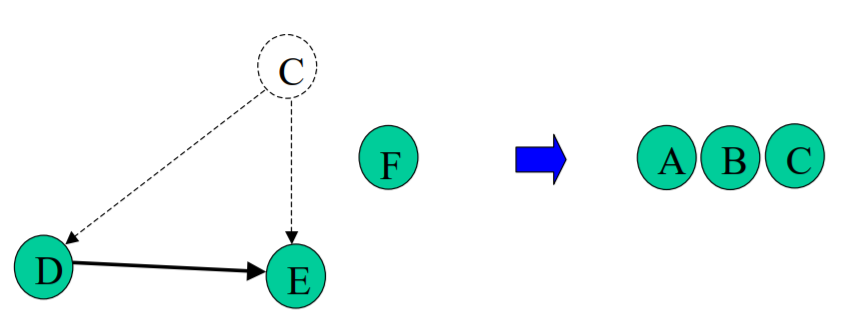
Delete this vertex of in-degree 0 and all its outgoing edges from the graph. Place it in the output.****



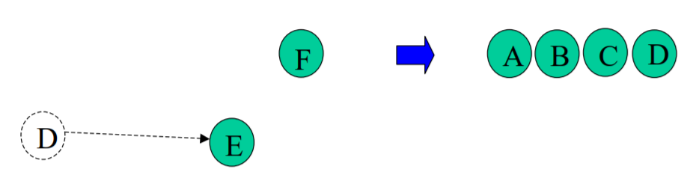
Repeat Step 1 and Step 2 until graph is empty



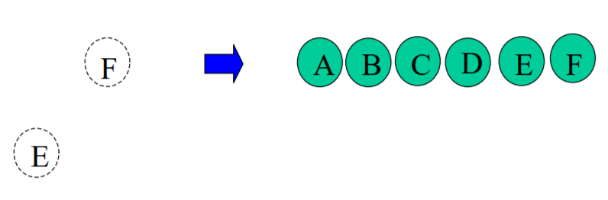
Select B. Copy to sorted list. Delete B and its edges



Select C. Copy to sorted list. Delete C and its edges.

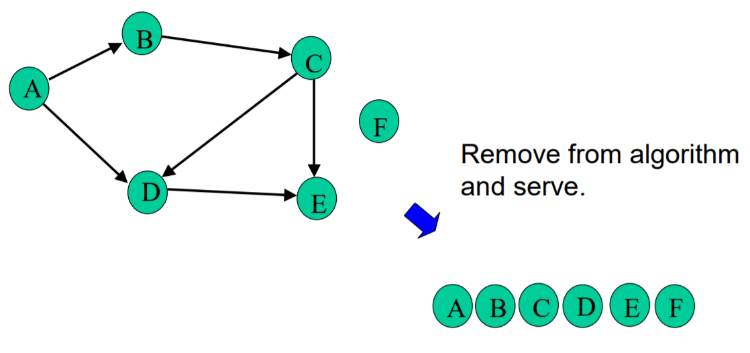


Select D. Copy to sorted list. Delete D and its edges



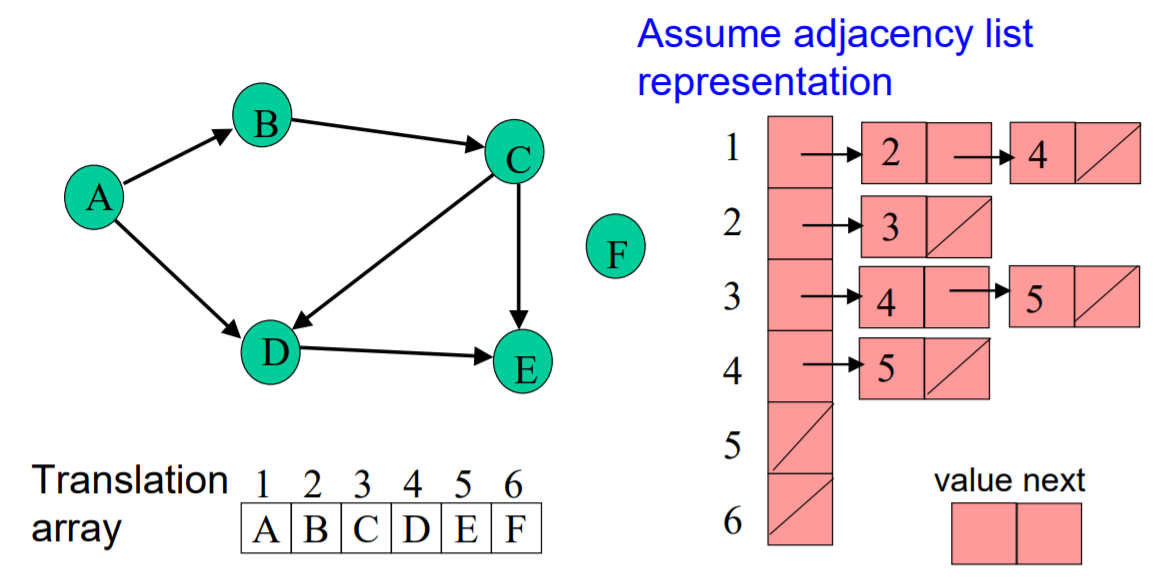
Select E. Copy to sorted list. Delete E and its edges.

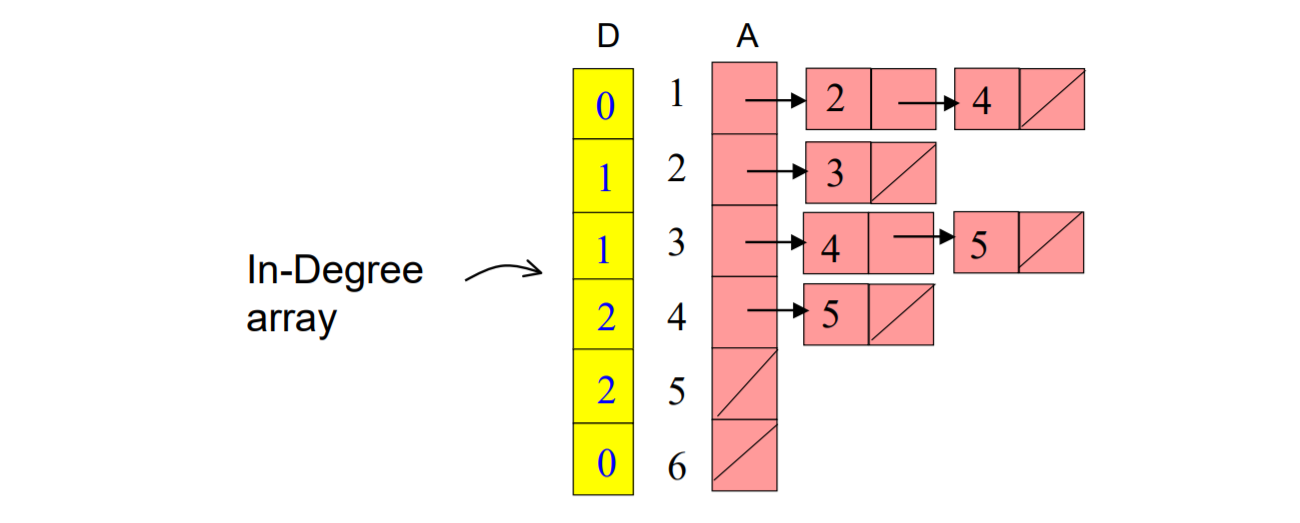
Select F. Copy to sorted list. Delete F and its edges.

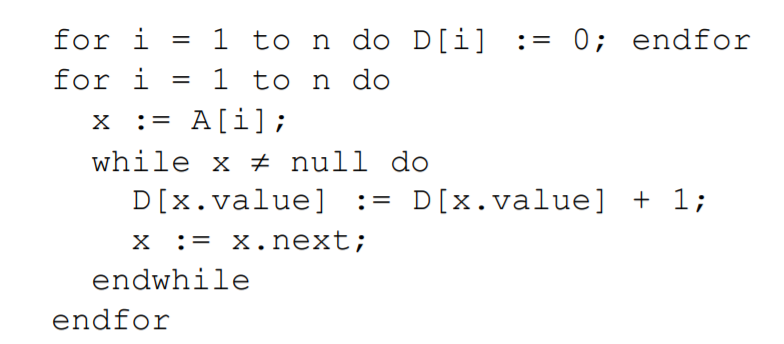


**All Selected:**

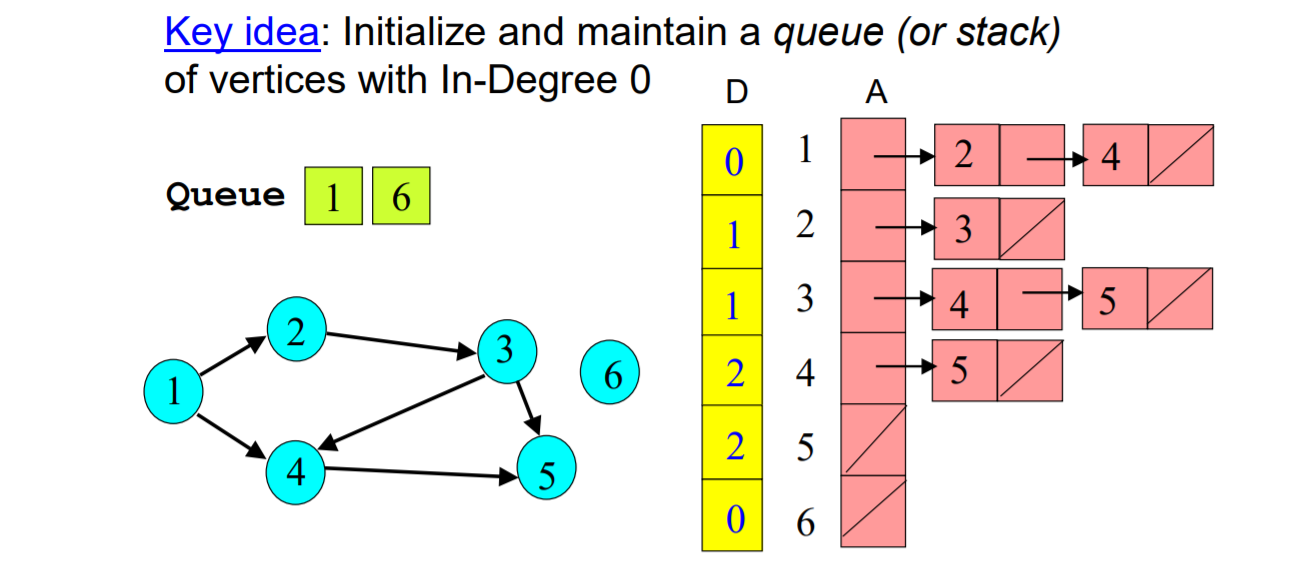
**Implementation:**



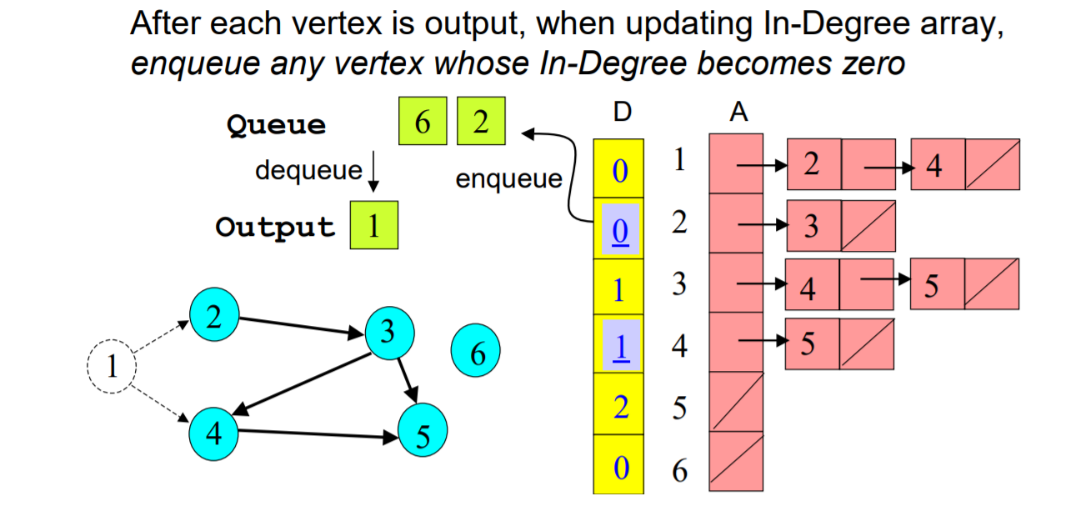
**Calculate In-degrees:**

**Calculate In-degrees:**

**Maintaining Degree 0 Vertices:**



**Topo Sort using a Queue:**

****

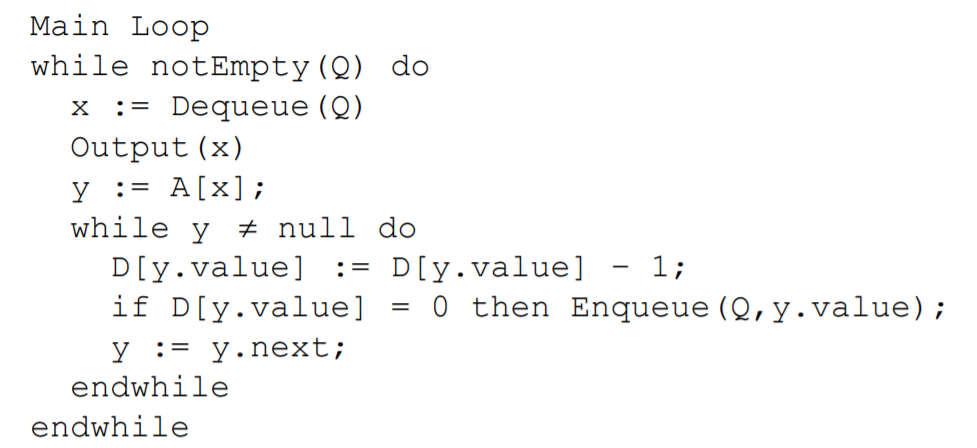
**Topological Sort Algorithm:**

1. Store each vertex’s In-Degree in an array D

2. Initialize queue with all “in-degree=0” vertices

3. While there are vertices remaining in the queue: (a) Dequeue and output a vertex (b) Reduce In-Degree of all vertices adjacent to it by 1 (c) Enqueue any of these vertices whose In-Degree   
became zero .

4. If all vertices are output then success, otherwise there is a cycle.

**Some Detail:**

**Topological Sort Analysis:**

Initialize In-Degree array: O(|V| + |E|)

• Initialize Queue with In-Degree 0 vertices: O(|V|)

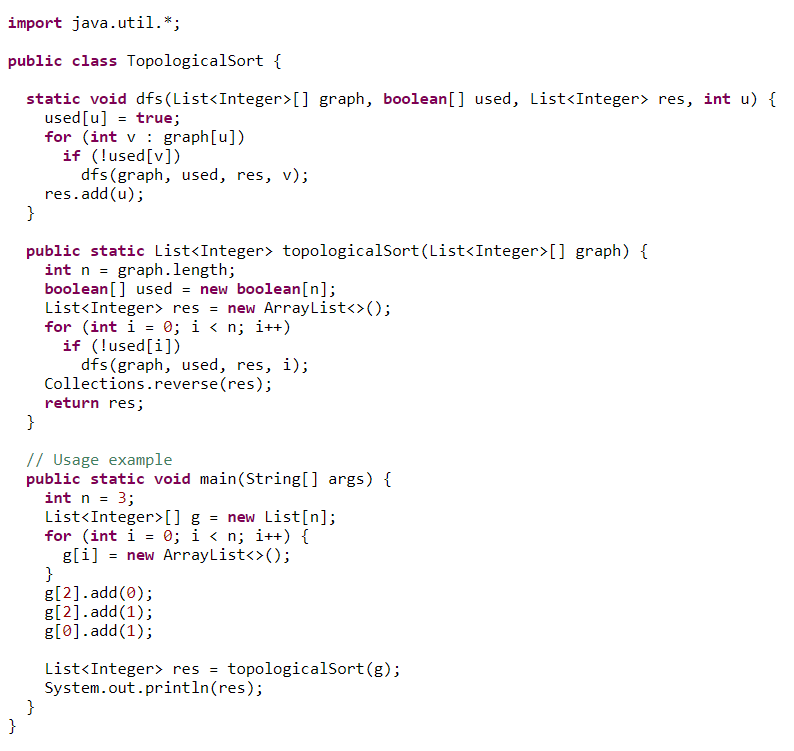
• Dequeue and output vertex:

› |V| vertices, each takes only O(1) to dequeue and output: O(|V|)

• Reduce In-Degree of all vertices adjacent to a vertex and Enqueue any In-Degree 0 vertices:

› O(|E|)

• For input graph G=(V,E) run time = O(|V| + |E|) › Linear time!

**Sample Code:**