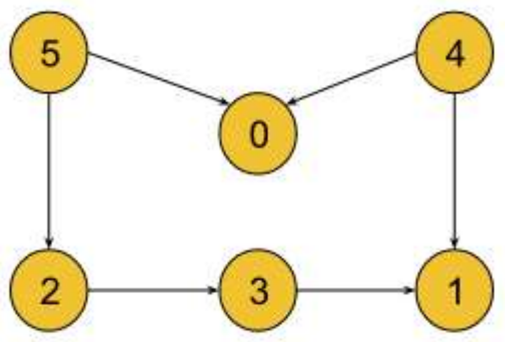
Kahn’s Algorithm | Topological Sort Algorithm

Example 1:
Input Format: V = 6, E = 6



Result: 5, 4, 0, 2, 3, 1

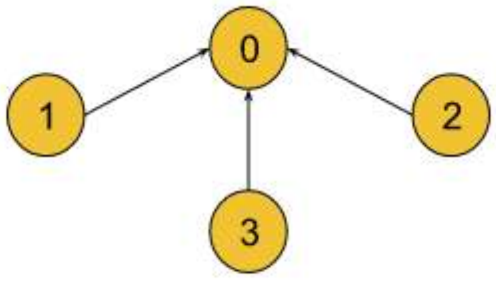
Explanation: A graph may have multiple topological sortings.

The result is one of them. The necessary conditions for the ordering are:

- According to edge **5 -> 0**, node 5 must appear before node 0 in the ordering.
- According to edge **4 -> 0**, node 4 must appear before node 0 in the ordering.
- According to edge **5 -> 2**, node 5 must appear before node 2 in the ordering.
- According to edge **2 -> 3**, node 2 must appear before node 3 in the ordering.
- According to edge **3 -> 1**, node 3 must appear before node 1 in the ordering.
- According to edge **4 -> 1**, node 4 must appear before node 1 in the ordering.

The above result satisfies all the necessary conditions.
[4, 5, 2, 3, 1, 0] and [4, 5, 0, 2, 3, 1] are also such topological sortings that satisfy all the conditions.

Example 2:
Input Format: V = 4, E = 3



Result: 1, 2, 3, 0

Explanation: The necessary conditions for the ordering are:

- For edge **1 -> 0** node 1 must appear before node 0.
- For edge **2 -> 0** node 1 must appear before node 0.
- For edge **3 -> 0** node 1 must appear before node 0.

[2, 3, 1, 0] is also another topological sorting for the graph.

Topological sorting only exists in Directed Acyclic Graph (DAG). If the nodes of a graph are connected through directed edges and the graph does not contain a cycle, it is called a **directed acyclic graph(DAG)**.

The ***topological sorting*** of a directed acyclic graph is nothing but the linear ordering of vertices such that if there is an edge between node u and v(u -> v), node u appears before v in that ordering.

Approach:

Breadth First Search or BFS is a traversal technique where we visit the nodes level-wise, i.e., it visits the same level nodes simultaneously, and then moves to the next level.

Initial Configuration:

Indegree Array: Initially all elements are set to 0. Then, We will count the incoming edges for a node and store it in this array. For example, if indegree of node 3 is 2, indegree[3] = 2.

Queue: As we will use BFS, a queue is required. Initially, the node with indegree 0 will be pushed into the queue.

Answer array: Initially empty and is used to store the linear ordering.

The algorithm steps are as follows:

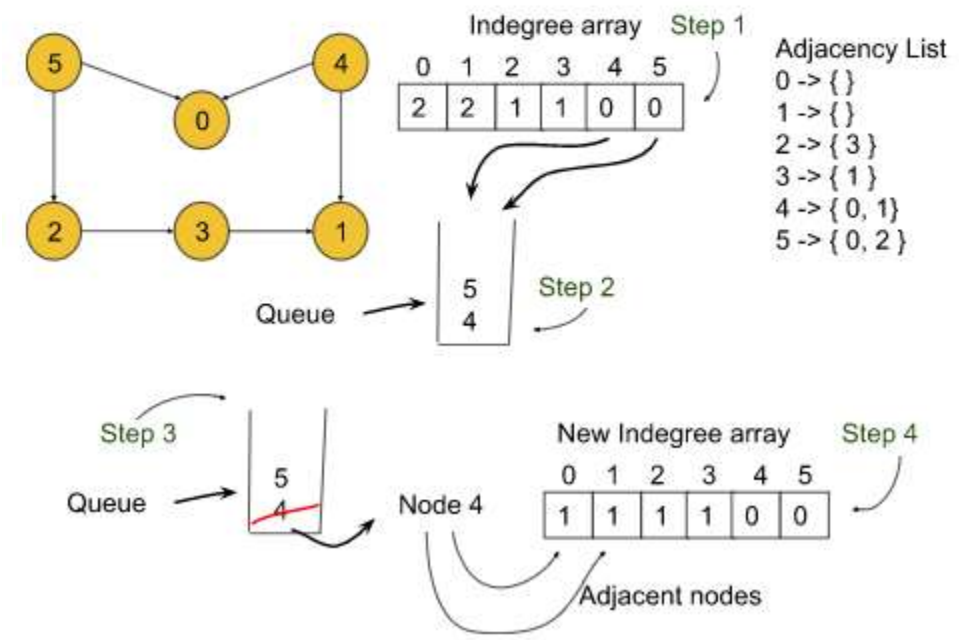
1. First, we will calculate the indegree of each node and store it in the indegree array. We can iterate through the given adj list, and simply for every node u->v, we can increase the indegree of v by 1 in the indegree array.
2. Initially, there will be always at least a single node whose indegree is 0. So, we will push the node(s) with indegree 0 into the queue.
3. Then, we will pop a node from the queue including the node in our answer array, and for all its adjacent nodes, we will decrease the indegree of that node by one. For example, if node u that has been popped out from the queue has an edge towards node v(u->v), we will decrease indegree[v] by 1.
4. After that, if for any node the indegree becomes 0, we will push that node again into the queue.
5. We will repeat steps 3 and 4 until the queue is completely empty. Finally, completing the BFS we will get the linear ordering of the nodes in the answer array.

Let's understand ***how to find the indegree(s)***:

By visiting the adjacency list, we can find out the indegrees for each node. For example, if node 3 is an adjacent node of node 2, we will just increase indegree[3] by 1 as the adjacency list suggests that node 3 has an incoming edge from node 2.

Note: *If you wish to see the dry run of the above approach, you can watch the video attached to this article.*

Let's quickly understand the algorithm using the below graph:



- First, we will calculate the indegrees for all 6 nodes. For node 0, it will be 2, for node 1, it will also be 2 and similarly, we will calculate for other nodes. The indegree array will look like this: {2, 2, 1, 1, 0, 0}.
- Next, the queue will be pushed with nodes 4 and 5 as their indegrees are 0. Then we will start the BFS.
- First, **node 4** will be popped out and kept in the answer array, and for all its adjacent nodes 0 and 1, the indegrees will be decreased by 1(**indegree array:** {**1**, **1**, 1, 1, 0, 0}). No nodes will be pushed into the queue, as there are no other nodes with indegree 0.
- Now, similarly, **node 5** will be popped out and indegree[0] and indegree[2] will decrease by 1 keeping node 5 in the answer array. Now, we will push nodes 0 and 2 into the queue as their indegree has become 0(**indegree array:** {**0**, 1, **0**, 1, 0, 0}).

- Then **node 0** will be popped out and as node 0 has no adjacent nodes it will be simply kept in the answer array.
- Next, **node 2** will be popped out and kept in the answer array while decreasing indegree[3] by 1(**indegree array:** {0, 1, 0, **0**, 0, 0}). Now indegree[3] is 0 and so node 3 is pushed into the queue.
- Next, **node 3** is popped out and kept in the answer array and indegree[1] is decreased by 1 as node 1 is the adjacent node of node 3(**indegree array:** {0, **0**, 0, 0, 0, 0}). Now indegree[1] is 0 and so node 1 is pushed into the queue.
- Lastly, **node 1** will be popped out and kept in the answer array, as node 1 has no adjacent nodes. Now, the BFS is completed for the graph.
- Finally, the answer array will look like: **{4, 5, 0, 2, 3, 1}.**