Kahn’s algorithm for Topological Sorting

**Solution: **In this article we will see another way to find the linear ordering of vertices in a directed acyclic graph (DAG). 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.
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
Algorithm: Steps involved in finding the topological ordering of a DAG:
Step-1: Compute in-degree (number of incoming edges) for each of the vertex present in the DAG and initialize the count of visited nodes as 0.
**Step-2: **Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue operation)
Step-3: Remove a vertex from the queue (Dequeue operation) and then.

1. Increment count of visited nodes by 1.
2. Decrease in-degree by 1 for all its neighbouring nodes.
3. If in-degree of a neighbouring nodes is reduced to zero, then add it to the queue.

Step 4: Repeat Step 3 until the queue is empty.
Step 5: **If count of visited nodes is **not equal to the number of nodes in the graph then the topological sort is not possible for the given graph.
How to find in-degree of each node?

1. 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)
2. **Auxillary Space:** O(V).
   The queue needs to store all the vertices of the graph. So the space required is O(V)

```
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -
> bool:
        graph = defaultdict(list)
        for u,v in prerequisites:
            graph[u].append(v)
        inDegrees = [0]*numCourses
        for i in range(numCourses):
            for nbr in graph[i]:
                inDegrees[nbr] = inDegrees[nbr]+1
```

```
        queue = []
        for i in range(numCourses):
            if inDegrees[i]==0:
                queue.append(i)

        count = 0
        while len(queue)>0:
            ele = queue.pop(0)
            count = count+1
            for nbr in graph[ele]:
                inDegrees[nbr] = inDegrees[nbr]-1

                if inDegrees[nbr]==0:
                    queue.append(nbr)

        return count==numCourses
        # print(count)
```

```
class Solution:
    def findOrder(self, numCourses: int, prerequisites: List[List[int]]) -
> List[int]:
        graph = defaultdict(list)
        for u,v in prerequisites:
            graph[u].append(v)
        inDegrees = [0]*numCourses
        for i in range(numCourses):
            for nbr in graph[i]:
                inDegrees[nbr] = inDegrees[nbr]+1
        queue = []
        for i in range(numCourses):
            if inDegrees[i]==0:
                queue.append(i)

        count = 0
        topSort = []
        while len(queue)>0:
            ele = queue.pop(0)
            count = count+1
            topSort.append(ele)
            for nbr in graph[ele]:
                inDegrees[nbr] = inDegrees[nbr]-1

                if inDegrees[nbr]==0:
```

```
                queue.append(nbr)

        if count==numCourses:
            return topSort[::-1]
        else:
            return []
```