**Problem 1 :** [**https://practice.geeksforgeeks.org/problems/alien-dictionary/1**](https://practice.geeksforgeeks.org/problems/alien-dictionary/1)

**Code :** [**https://practice.geeksforgeeks.org/viewSol.php?subId=97f7b20b799218aec06a739042851586&pid=700494&user=alaymehta1**](https://practice.geeksforgeeks.org/viewSol.php?subId=97f7b20b799218aec06a739042851586&pid=700494&user=alaymehta1)

**(imp) Problem 2 : (with indegree concept)** [**https://www.codingninjas.com/codestudio/problems/shortest-common-supersequence\_1377949?topList=top-graphs-interview-questions&leftPanelTab=0**](https://www.codingninjas.com/codestudio/problems/shortest-common-supersequence_1377949?topList=top-graphs-interview-questions&leftPanelTab=0)

**Approach :**

-> Make a directed graph with dependencies, **edge from arr[i] to arr[i+1].**

-> Store indegree of all nodes from 1 to N.

-> Add **nodes with indegree 0** in queue.

-> At every iteration , queue size should be 1 , **means nodes with indegree 0 should be EXACTLY 1 at every instance**, else there are 2 possibilities in topo sort which is invalid.

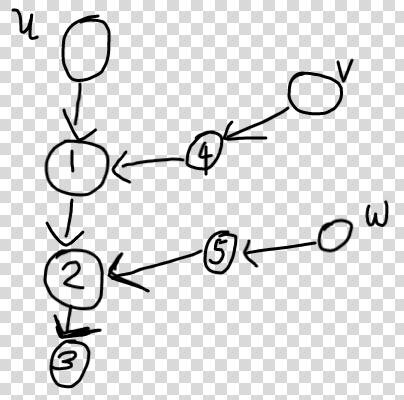
-> Now add that node to answer and remove outgoing edges from that node, and add new node with indegree 0 in queue, if more than 1 nodes are added , then the above condition will return false anyways.

-> Repeat until queue has element.

-> Now check **if size of answer == size of primary vector & the element order is same**, if both satisfied then return true.

**Problem 3(with Code) :** [**https://binarysearch.com/problems/Minimum-Starting-Nodes-to-Visit-Graph**](https://binarysearch.com/problems/Minimum-Starting-Nodes-to-Visit-Graph)

**Approach :**



-> If any node has indegree >=1 (e.g node 3), means it can be reached by it’s parent,if it’s parent also has indegree >=1,means it is also reachable from its parents.

-> Now by keeping on traversing parents like this you will find **Atleast 1 node** with indegree = 0,as there is no cycle.

-> So **Answer = No of nodes with indegree 0.**

=> **One follow up question is :**

**What are the minimum no of edges to be added to DAG , so that every vertex is reachable from some vertex ‘S’ .**

Ans : Select any node with indegree 0 in DAG as ‘S’ , suppose there are ‘n’ nodes with indegree=0, then by having ‘n-1’ edges from ‘S’ to other nodes with indegree 0, we make all nodes reachable from ‘S’.

**(imp)Problem** **4(topological sort with indegree concept + DP)**: <https://leetcode.com/problems/largest-color-value-in-a-directed-graph/>

**Approach** (pasting the indegree concept from theory part):

* **(imp)Topological Sorting(indegree concept)on DAG :**

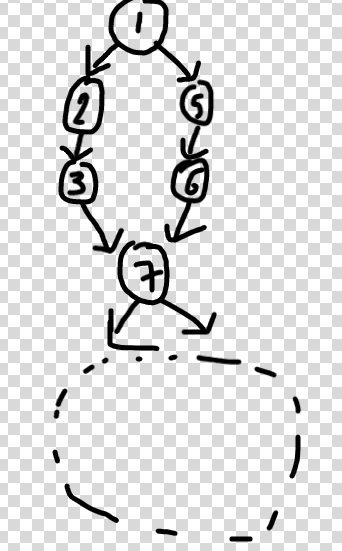
**-> A DAG will have atleast one node with indegree 0 and & atleast one node with outdegree 0.**

**PROCESS :**

1. So, you can start topological sort from a node with indegree 0 and end at node with outdegree 0.
2. After processing node with indegree 0, pass the information to it’s children and decrease the indegree of it’s children, so now there will again be atleast 1 new node with indegree 0 and this process repeats until it encounters the last node with outdegree 0.

**=> Topological Sort processes a node ‘u’ only after all it’s ancestors**(parent(node with incoming edge on ‘u’), parent’s parent(node with incoming edge on parent) and so on till node with indegree 0) **are being processed.**

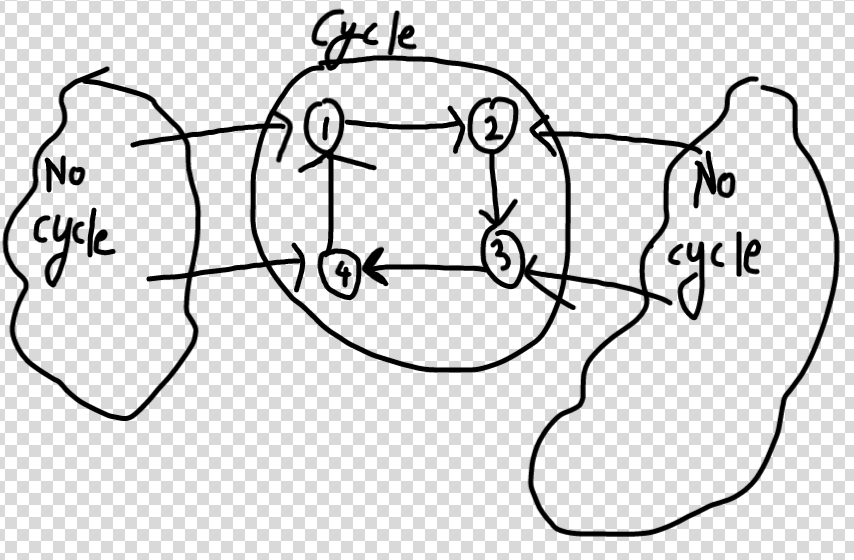
**=> (imp) This fact ensures that when it’s the turn of some node ‘u’ to get processed, ALL THE PATHS ENDING ON NODE ‘u’ would already have been processed.**



In above problem , one possible execution is 1 2 5 3 6 7 .

**-> (imp)So before we reach 7, we would have optimal answers for paths 1->2->3 and 1->5->6 which are incident on 7 , so now node 7 can be processed and this information can be passed to the children as we know that we already found optimal answer amongst all paths ending at 7.**

**=> Cycle Detection using Topological Sort :**

****

As per the topo-sort procedure, all the parts of graph where there will be no cycle will be processed without any problem initially, and also all the edges incident on the part with a cycle will be removed, **but it will be impossible to remove any edge which is a part of cycle** .

Why ? Because all nodes are dependent on each other (deadlock), to make their own indegree 0.

For e.g : 1 wants 4 to get processed, so that the edge incoming on it can be removed, but 4 can’t do it until 3 gets processed, but 3 is dependent on 2 and 2 back on 1, which creates a deadlock.

**=> So during topo sort procedure, make a boolean “visited” array, and if there is a cycle, it would be impossible for any node in the cycle to be marked as visited**(as indegree would never become 0). So if any node has visited = false after the whole procedure ends, then there definitely is a cycle.

**Code :** <https://leetcode.com/submissions/detail/623122702/>