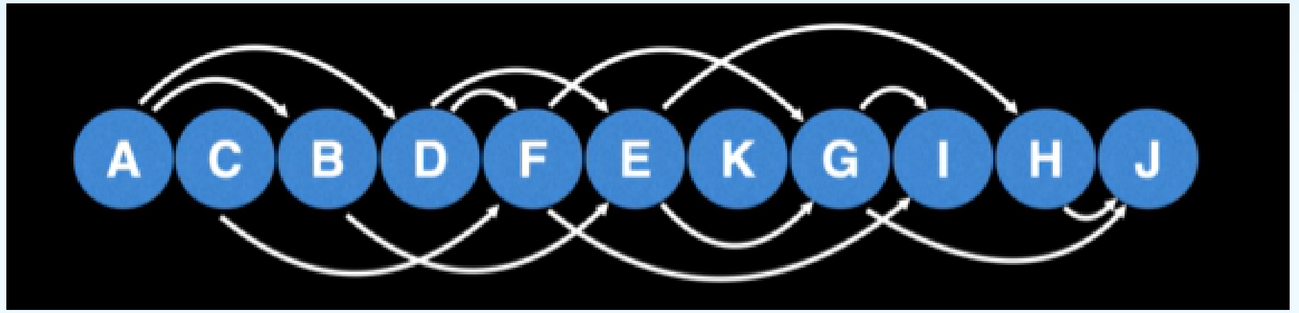
KAHN'S ALGORITHM

Topological Sorting Algorithm

Natanya Modi (21BAI1405)

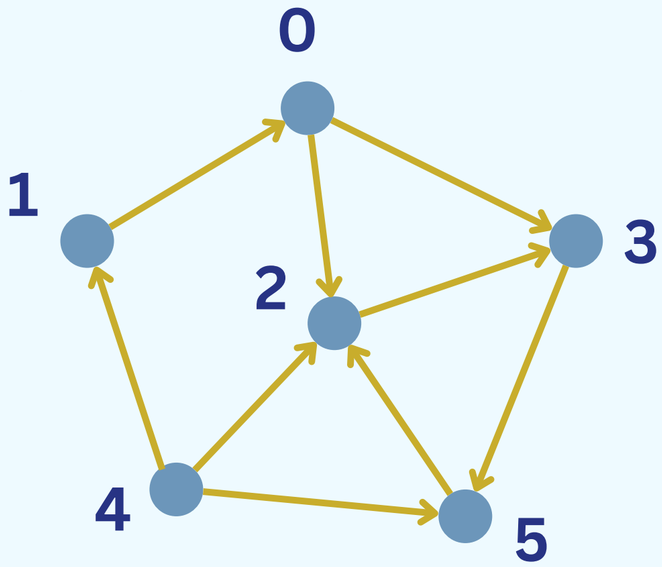


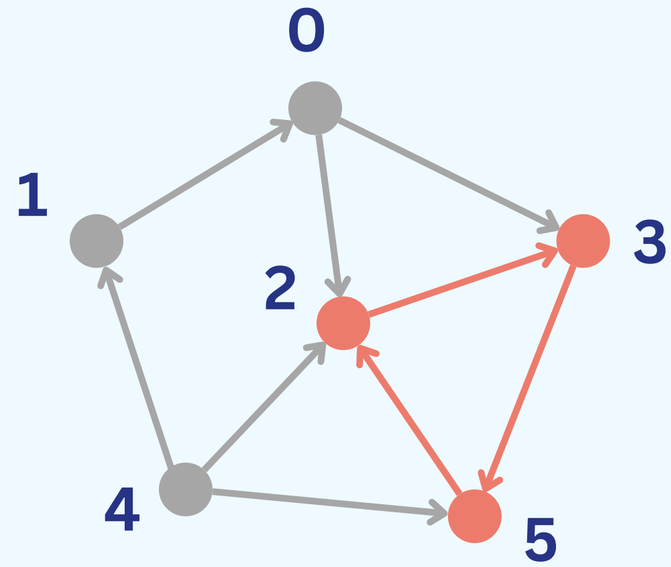
A **topological ordering** is an ordering of the nodes in a directed graph where for each directed edge from node X to node Y, node X appears before node Y in the ordering.

Only certain types of graphs have topological orderings. These are Directed Acrylic Graphs which is basically a graph with no cycles.

Topological orderings are **NOT** unique

WHY CAN'T WE WORK WITH GRAPHS HAVING CYCLES?





4 , 1 , 0 , ?

Logic

The logic behind Kahn's algorithm is to repeatedly remove the nodes having no dependencies and then add them to the topological ordering.

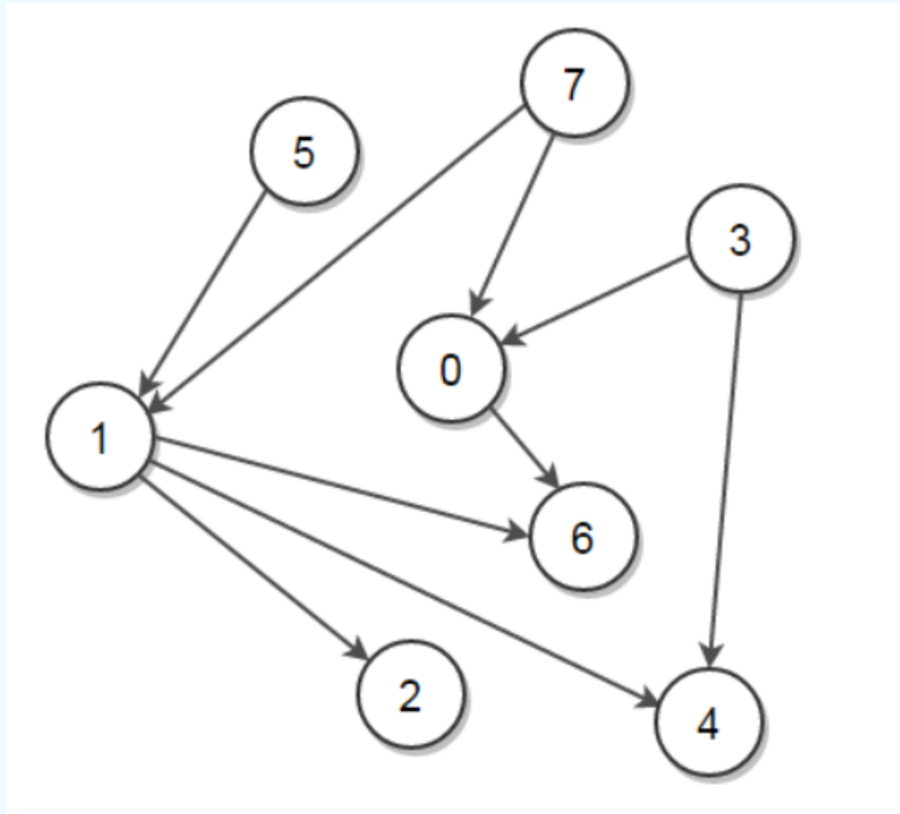
As we remove the nodes having no dependencies from other points, new nodes without incoming edges should become free.

Queue

7

5

3



0	1	2	3	4	5	6	7
2	2	1	0	2	0	2	0

Queue

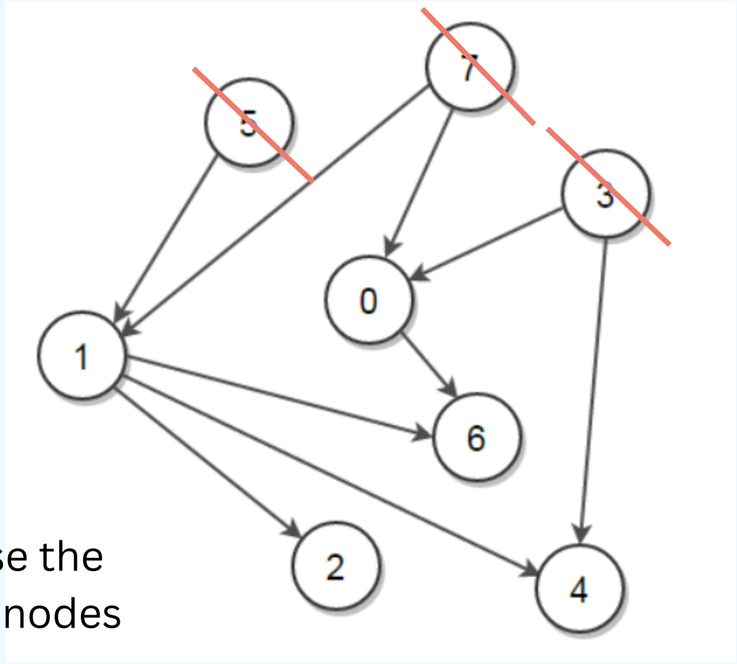
7

5

Queue

1

0



remove 3 and decrease the degree of all affected nodes

0	1	2	3	4	5	6	7
1	2	1	0	1	0	2	0

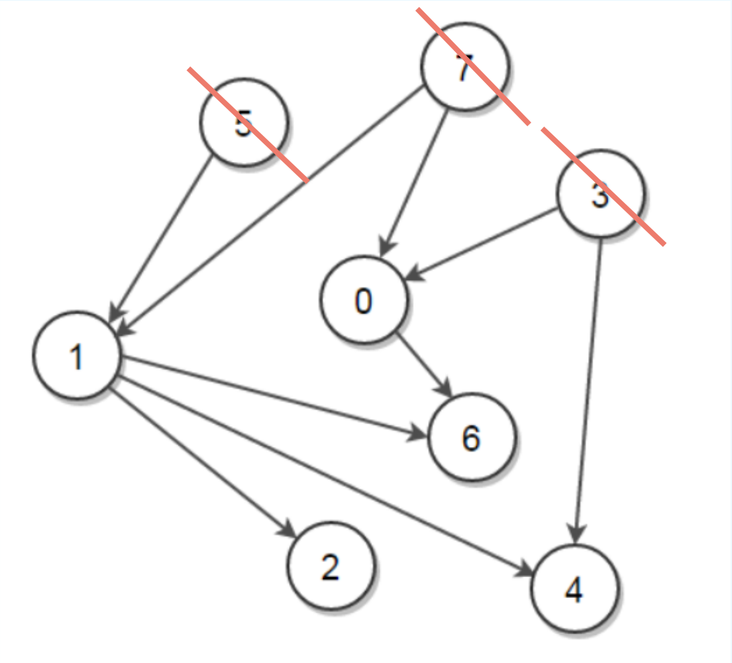
removing 5 and then 7

0	1	2	3	4	5	6	7
0	0	1	0	1	0	2	0

Topological ordering: 3 5 7

Queue
1
0

Queue
6
4
2



removing 0

0	1	2	3	4	5	6	7
0	2	1	0	1	0	1	0

removing 1

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

Topological ordering: 3 5 7 0 1 2 4 6

Pseudocode

```
function kahnsAlgorithm(graph, numNodes):
    initialize an array in Degree with size
numNodes, filled with zeros
    initialize an empty array topologicalOrder

    // Compute the in-degree of each node
    for each node in graph:
        for each neighbor of node:
            increment inDegree[neighbor]

    initialize an empty queue q

    // Enqueue nodes with an in-degree of 0
    for i = 0 to numNodes - 1:
        if inDegree[i] is 0:
            enqueue i into q

    while q is not empty:
            current = dequeue from q
            append current to topologicalOrder

            // Decrease the in-degree of neighbors
    and enqueue if their in-degree becomes 0
            for each neighbor of current:
                decrement inDegree[neighbor]
                if inDegree[neighbor] is 0:
                    enqueue neighbor into q
```

Pseudocode

```
// Check if a topological sort is possible
 if size of topologicalOrder is equal to
numNodes:
 return topologicalOrder
 else:
 return an empty array

// Example usage:
numNodes = number of nodes in the graph
graph = adjacency list representation of the
graph

topologicalOrder = kahnsAlgorithm(graph,
numNodes)

if topologicalOrder is empty:
    print "A topological sort is not possible
due to cycles."
else:
    print "Topological Order: "
 for each node in topologicalOrder:
    print node
```

TIME COMPLEXITY: O(V+E)

The time complexity of Kahn's Algorithm is O(V + E), where V is the number of vertices and E is the number of edges in the graph.

Computing the in-degree of each node: This step takes O(V + E) time, as it iterates over all nodes and their neighbors in the graph.

Performing the topological sorting: This step also takes O(V + E) time, as it iterates over all nodes and their neighbors in the graph.