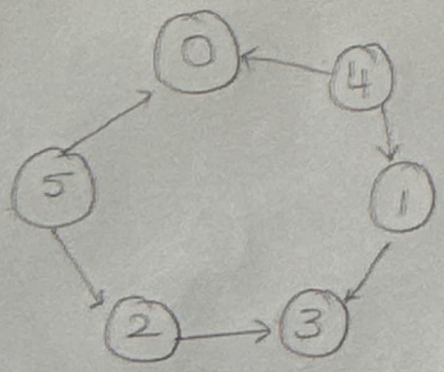
Topo Sort



Dfs : Explore all the neighbor, If no node is Left then it is not prereq to any other node.

Dfs(0) : add to op since it has no nei op = [0]

Mark the node as visited

Dfs(1) :

↳ nei = 3

Dfs(3) : add to op op = [0, 3]

mark as visited

↳ It has no more neighbor Left

It can be added to op op = [0, 3, 1]

Mark as Visited

Dfs(2)

↳ nei = 3

since this was already Explore, We can skip

Why? Because 3 would alread be in op list Before 2, → so when we reverse it 3 will appear after 2 for sure

↳ It has no more neighbor left

add to op → op: [0, 3, 1, 2]

Mark as visited

Dfs(3):

↳ Already visited. so no need to revisit again

Dfs(4)

↳ nei = 0
already visited

↳ nei = 1
already visited

↳ Since there is no more neighbor

Add to op → op: [0, 3, 1, 2, 4]

mark as visited

Dfs(5)

↳ nei = 0
already visited

↳ nei = 2
already visited

↳ Since there is no more neighbor

Add to op → op: [0, 3, 1, 2, 4, 5]

Mark as visited

Return the reversed op: [5, 4, 2, 1, 3, 0]

why? A node gets pushed only after all its dependant neighbors are explored, so the list is built in reverse order.

Kahn's Algorithm

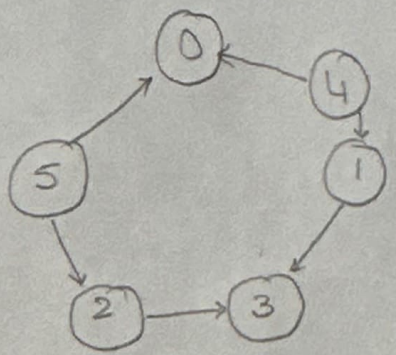
Intution is:

- A node can only be done when all of its pre req are finished
- pre req = No of incoming node
- so we repeatedly pick task with indegree zero & mark them as completed.

Steps:

- Calculate Indegree :
  A Node with Indegree Zero has no prereq
- Iterate Over the node with Indegree Zero
  * Visit its neighbor & reduce the count of indegree
    ↳ Meaning the current prereq for nei node is complete | satisfied.

Eg:



Indegree :

0	1	2	3	4	5
2	1	1	2	0	0

Grab the node with no pre-req

St= [4, 5]

pop (5)

- nei = 0, 2
- reduce the indegree of nei

0	1	2	3	4	5
~~1~~	1	~~0~~	2	0	0

2
4
~~5~~

Since Indegree of (2) = 0, add this to stack,

This means all the prereq to unlock (2) is now complete

pop (4)

- nei = 0, 1
- reduce the indegree of nei

0	1	2	3	4	5
~~0~~	~~0~~	0	2	0	0

Indegree of (0) = 0

Indegree of (1) = 0

Add to stack

0
1
2
~~4~~
~~5~~

pop (2)

- nei = 3
- reduce Indegree of (3).

0	1	2	3	4	5
0	0	0	~~1~~	0	0

pop (1):

nei = 3

reduce Indegree of 3 :

0	1	2	3	4	5
0	0	0	~~0~~	0	0

Indegree of (3) = 0

Add to stack

3
0
~~1~~
~~2~~
~~4~~
~~5~~

pop (0) - no neighbors

pop (3) no neighbors

So the order was

pop (5), pop (4), pop (2), pop (1), pop (0), pop (3)

This is the op: [5, 4, 2, 1, 0, 3]