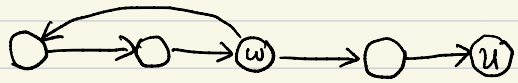
$m = |E|$.
$n = |V|$.

Correctness:

Claim (3.19) in textbook: If $G=(V,E)$ is a DAG, then there exist $u \in V$, such that u has no incoming edges.

Proof: Suppose by contradiction that every node in G has at least one incoming edge. We show how to find a cycle in G.

Pick any node $u \in V$. and walk backwards.



Repeating this process, after at most $n+1$ steps, we must have visited a node w twice, this form a cycle.

Topological Sort works!

Because removing a node from G does not introduce new cycles. (G remains acyclic)

Claim (3.20): If G is a DAG, then G has a topological ordering.

Runtime: $O(n^2)$ if G is given by its adjacency matrix.

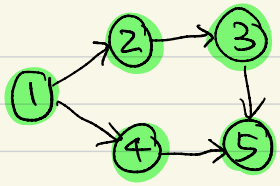
- WHILE loop runs for n iterations.
- Maintain the incoming degree of each node $deg[u]$.
- In each iteration of the WHILE loop:
  - Find a node with no incoming edges: loop over $u = 1..n$ and check if $deg[u]=0$. $O(n)$
  - Delete u and all edges incident to it when deleting an edge (u,v): $A[u][v]=0$; $deg[v]=deg[v]-1$. $O(n)$

$O(m+n)$ if G is represented by adjacency lists.

- Deleting edges can be slow for adj. lists. active[u].
  We say a node is active if it has not been added to the ordering. Initially all nodes are active.
- For every node u, we maintain the number of incoming edges that u has from active nodes. deg[u]
- We maintain S:
  $$S = \{u \in V : u \text{ is active}, deg[u]=0\}.$$
- At the beginning, we can set up active[], deg[], and S in $O(m+n)$ time
- In each iteration, we pick a node u from S. set active[u]=0. and for every edge $(u,v) \in E$:
  $deg[v] = deg[v] - 1$.
  if $deg[v]=0$ then add v to S.

In total, this takes $O(m)$ time because we delete each edge exactly once.

Example:



$S = [1]$

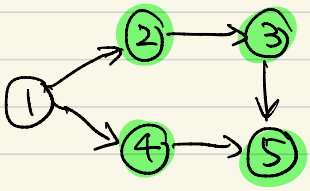
deg:

1	2	3	4	5
0	1	1	1	2

picked in the 1st iteration

deg[i] is the # of incoming edges to node i from active nodes.

active



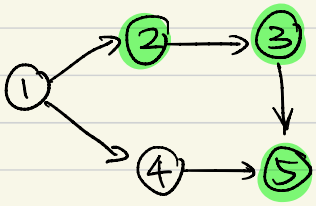
$S = [2, 4]$

deg:

1	2	3	4	5
0	0	1	0	2

suppose we picked 4.

can pick either in the 2nd iteration



deg:

1	2	3	4	5
0	0	1	0	1

$S = [2]$

only choice in the 3rd iteration.