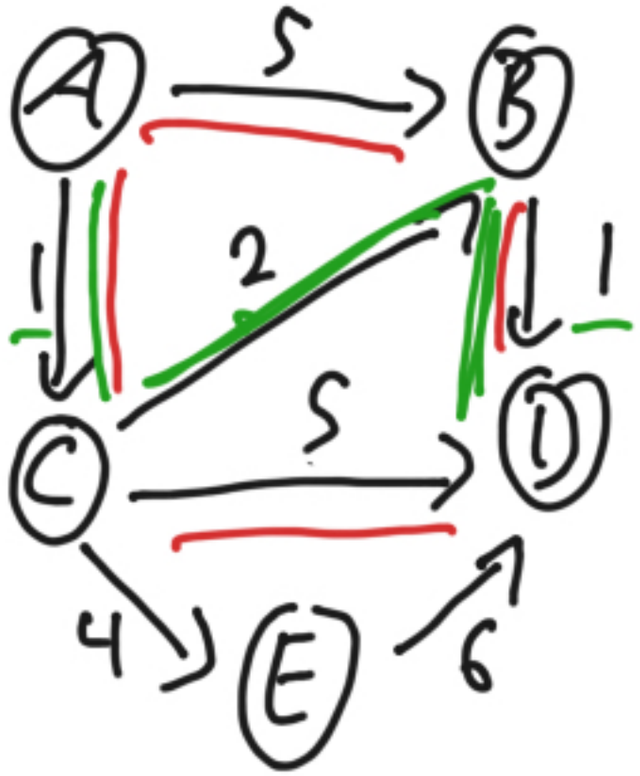
Graphs

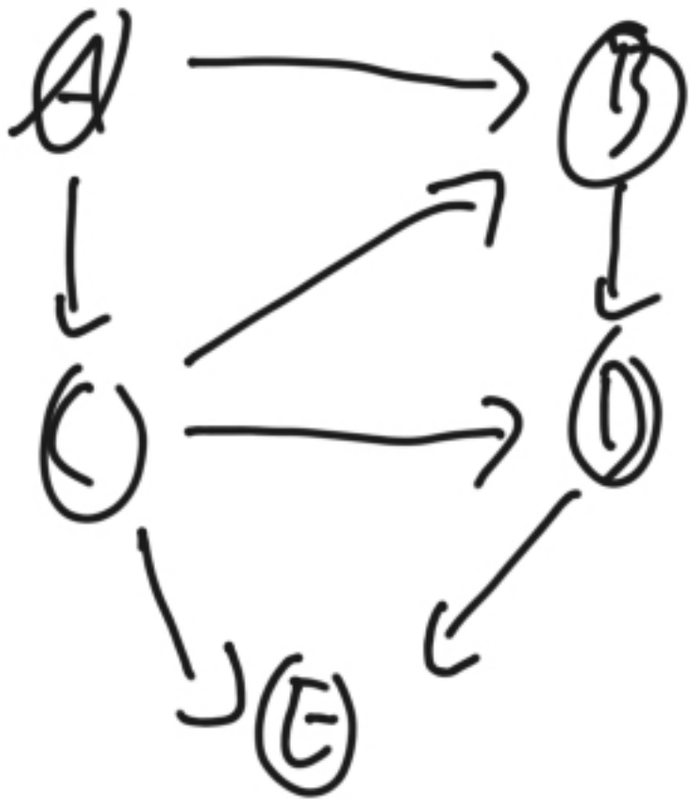


vertices $V = \{A, B, C, D, E\}$

Edges $E = \{(A,B), (A,C), (B,D), (C,D), (C,B), (C,E), (E,D)\}$

directed v. undirected graphs

weighted v. unweighted graphs



Path: sequence of vertices between vertices s and t

$\langle s, i, j, \ldots t \rangle$ such that there is an edge between each pair of vertices,

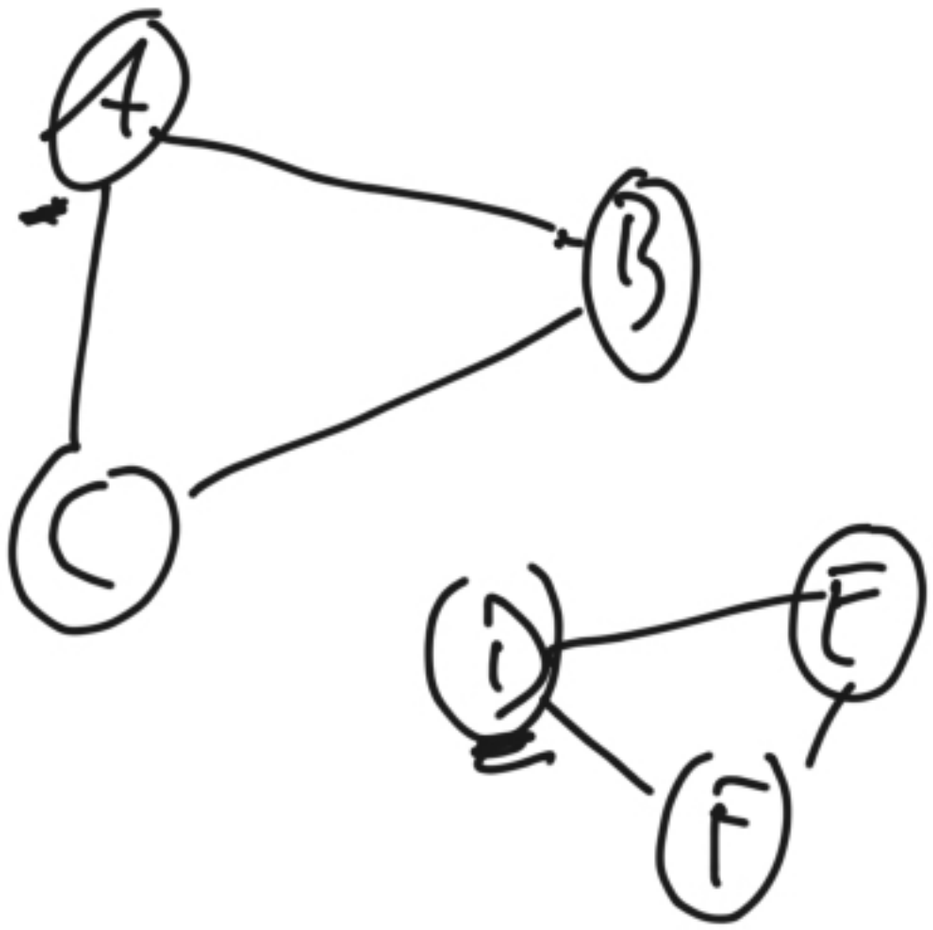
$(s,i) \in E$, $(i,j) \in E$, ...

Length of path: # edges = # vertices − 1

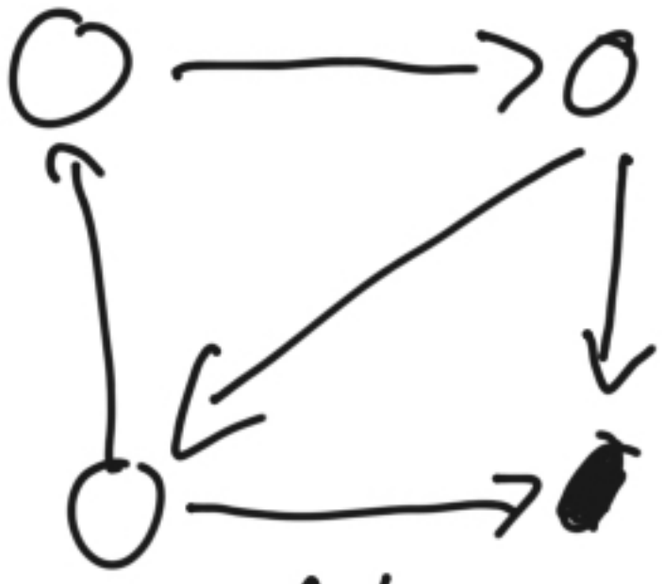
$\langle A, B, D \rangle$ has length 2

Connectedness

a graph is connected if there is some path between any pair of vertices.



directed graph

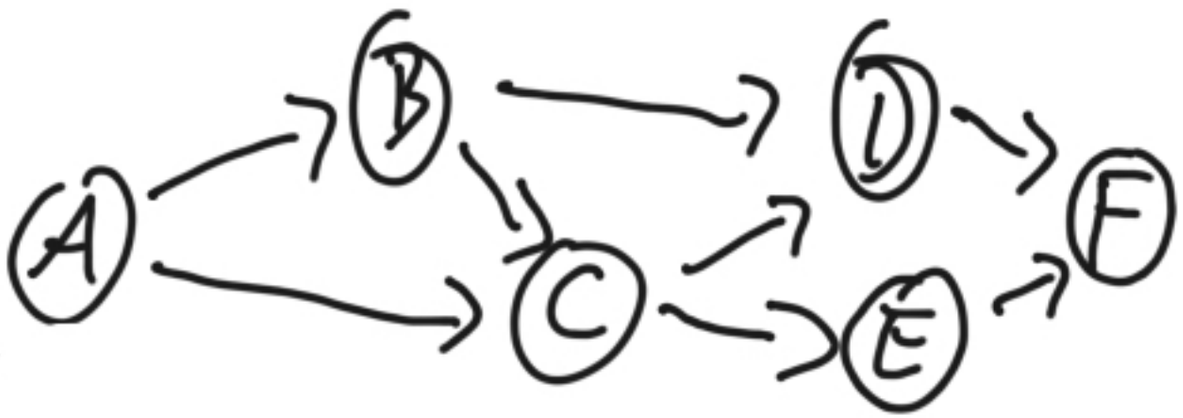


weakly connected

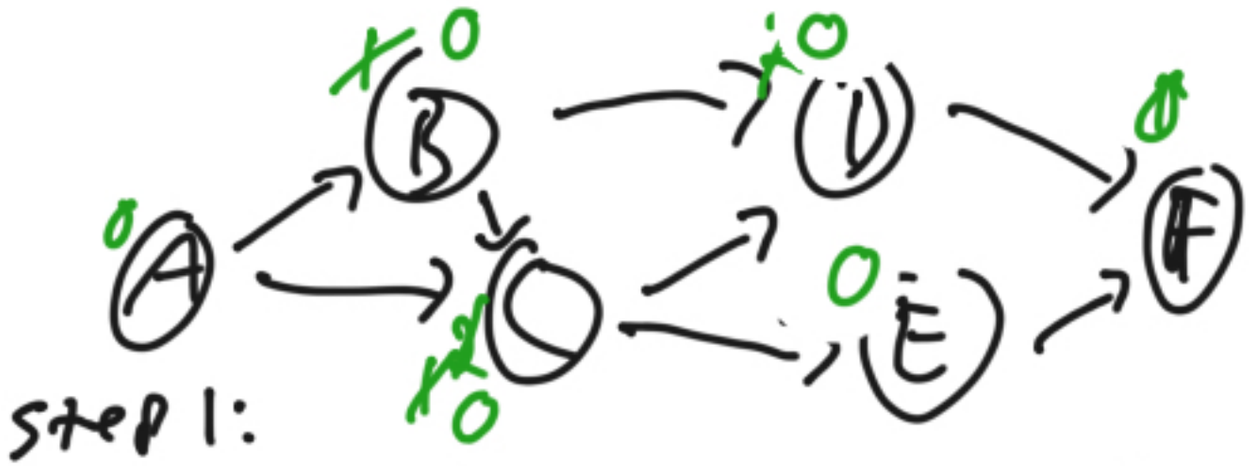
not strongly connected

A directed graph contains a cycle if there is some path $\langle A \ldots A \rangle$ of length ≥ 2.

A directed graph without cycles is called a Directed Acyclic Graph (DAG)



Topological Sort on a DAG



Queue:

A B C D E F

step 1:

Compute in-degree of each vertex

step 2:

use a queue and enque all vertices with in-degree 0.

while queue not empty:

s ← queue.dequeue()

add s to output

for all neighbors t of s:

t.indegree -= 1

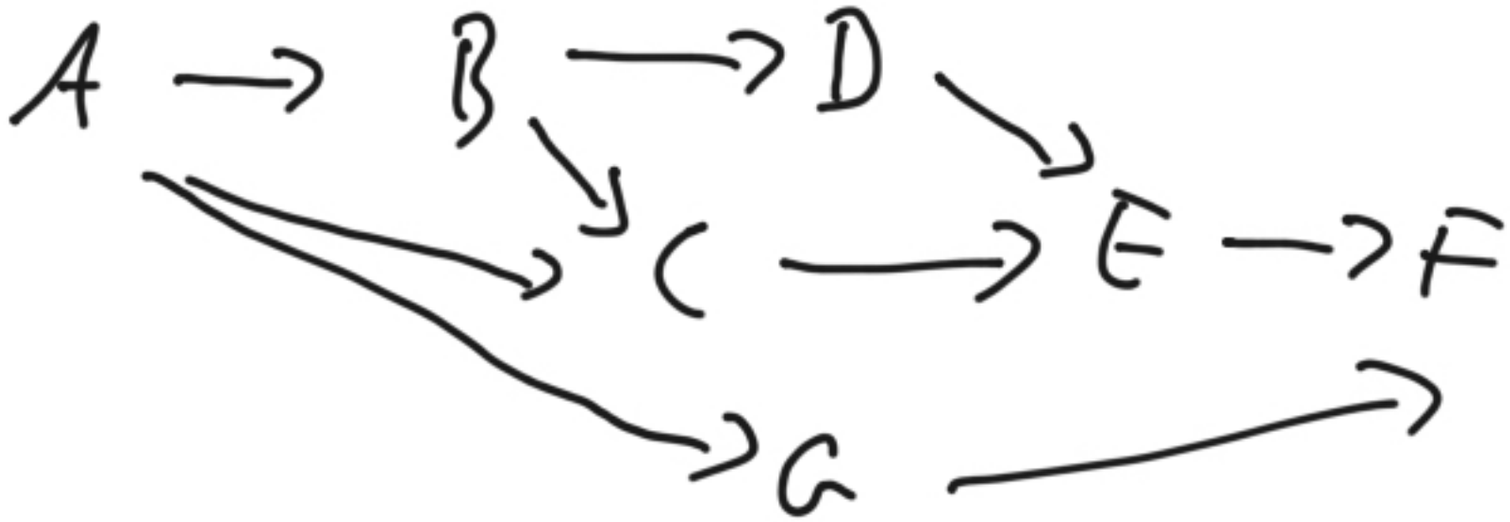
if t.indegree == 0:

queue.enque(t)

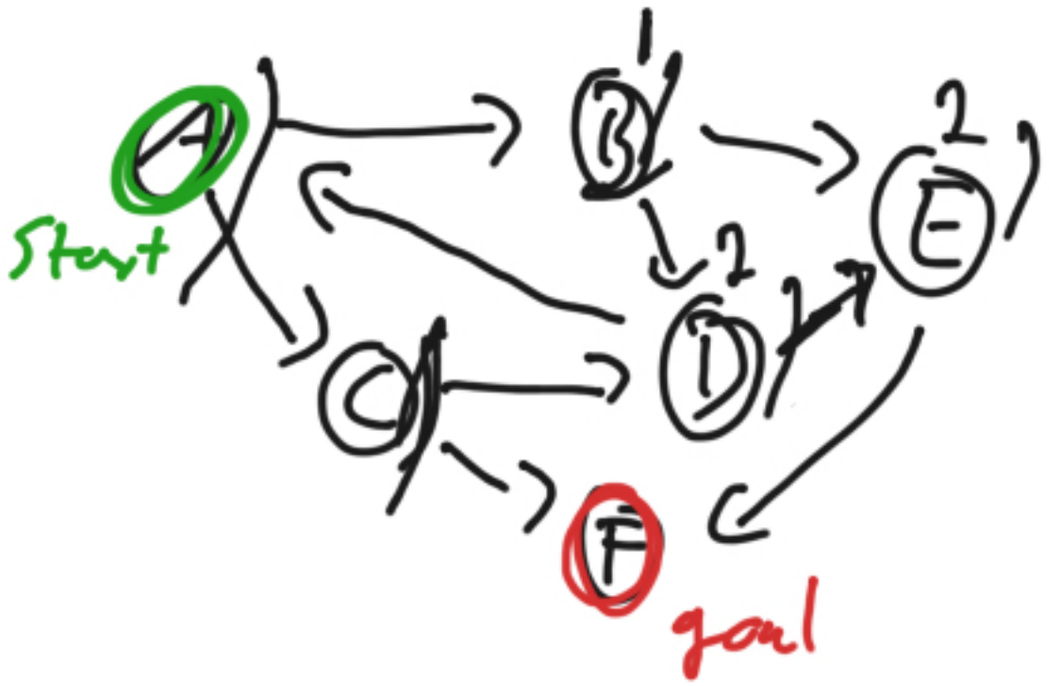
If there is a path between s and t in the graph, then s must appear before t in the topo sort.

A B C D E F

A B C E D F



BFS Breadth first search cost={'A': 0}



use a queue q
Set discovered
q.enque (start)

while not found and q not empty:

VISIT s → s = q.deque()

for each neighbor t of s:

if s not in discovered

if not t in discovered:
discovered.add(t)
if t is goal:
stop
enque (t)

DISCOVER t → cost[t] = cost[s]+1