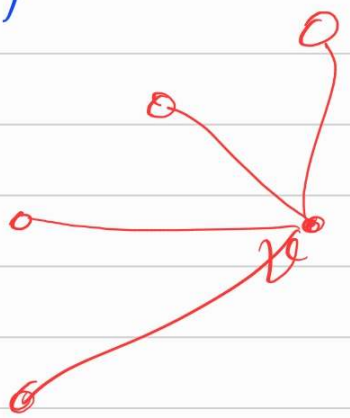
03/03/2022

To bound the runtime of the DFS algo. it suffices to bound the no. of additions and deletions to the Stack.

It suffices again to bound only the no. of additions to the Stack.

Q. How many times do you add a node to the Stack?

degree of that vertex many times.



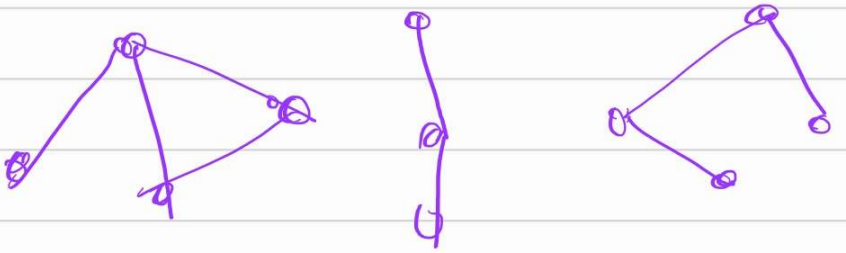
add only if Explored[v] = false.

Total runtime is at most

$$O\left(\underbrace{\left(\sum_{v \in V} \deg(v)\right)}_{\| \; 2 \cdot |Edges|} + n\right)$$

Fact:- $\sum_{v \in V} \deg(v) = 2 \cdot |Edges|$

All Connected Components:



Calling BFS/DFS gives you all the vertices reachable from the starting vertex, say s.

Connected Component of s

$$= \{ v \mid v \text{ is reachable from } s \}$$

Q: Consider $u \neq v \in V$.

Consider Conn-Component(u) & Conn-Component(v).

what is intersection of the two sets?

In other words, the sets are either identical or disjoint.

Q. How do you find all connected components?

Ans: Run BFS/DFS from every unexplored vertex.

Runtime will be $O(\#\text{edges} + \#\text{vertices})$

Directed graphs.

1) Adjacency matrix

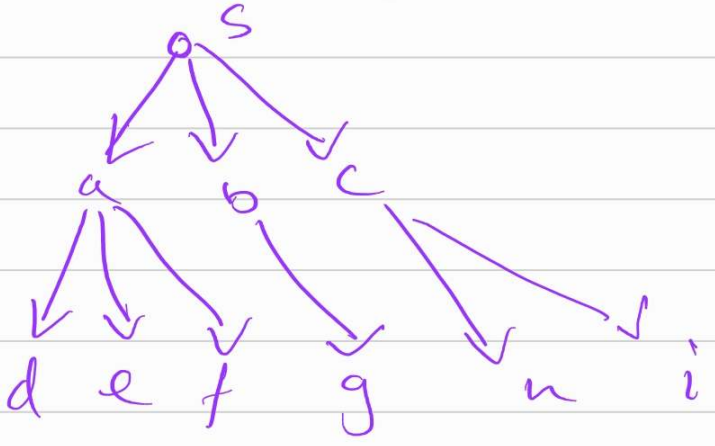
not a symmetric matrix.

$i \longrightarrow j$

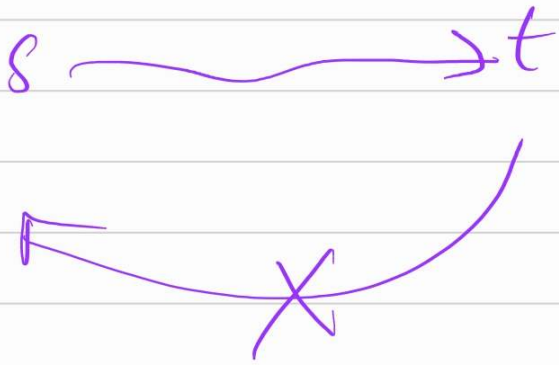
2) Adjacency list.

In-neighbors Out-neighbors

Q. what happens if you run BFS/ DFS on a directed graph?



It will give you all the nodes which are reachable from s.



Q. How do we find all the vertices that have a path to s?

Construct a graph G^{rev} which is obtained by reversing the direction of edges.

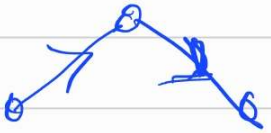
Running BFS on G starting at s gives you all the vertices that are reachable from s.

Running BFS on G^{rev} starting at s gives you all the vertices that have a path to s.

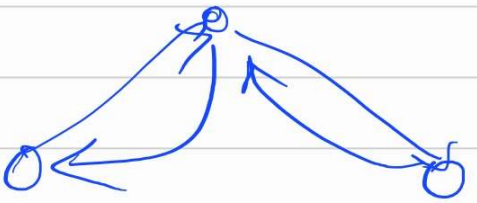
Q. What is the intersection of these sets. ?

The set of vertices which have paths to and from s.

Defn :- A directed graph is called Strongly connected. If for any two vertices there is a path from one to the other.



Not - Strongly connected.



Strongly connected.

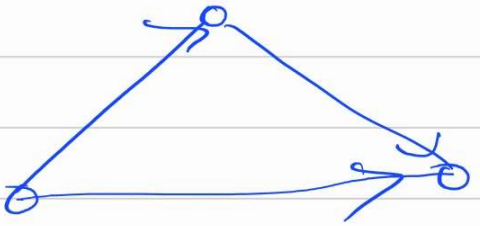
Defn :- For a vertex v in a directed graph

Strong - component (v)

$= \{ u \mid \exists$ a path from $u \rightsquigarrow v$ and $v \rightsquigarrow u \}$

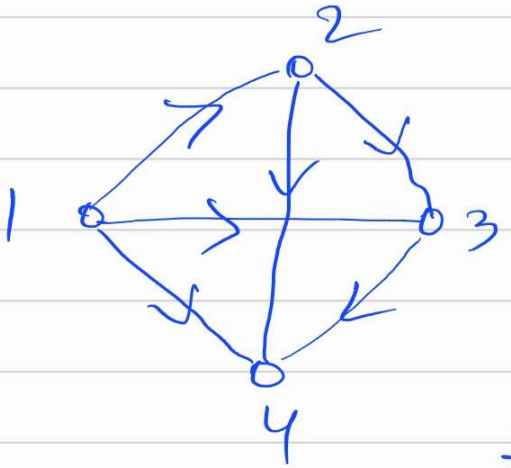
Directed Acyclic Graphs: (DAGs)

It's a directed graph with no directed cycles in it.



DAG

Connected undirected acyclic graphs = Trees



tail $\rightarrow (i \longrightarrow j)$ $\leftarrow$ head

On n vertices. (i, j)

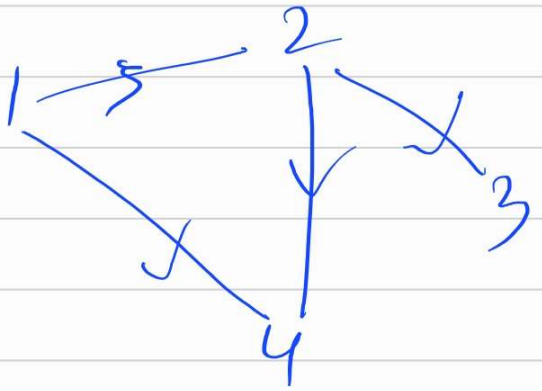
$\{1, \ldots, n\}$ for every $i < j$

$$\#\text{edges} = \binom{n}{2}$$

Set of tasks $\{1, \ldots, n\}$

s.t. 1 must be done before 4

2 ——————— 5.

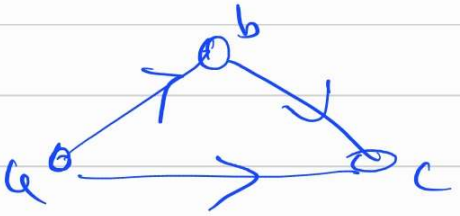


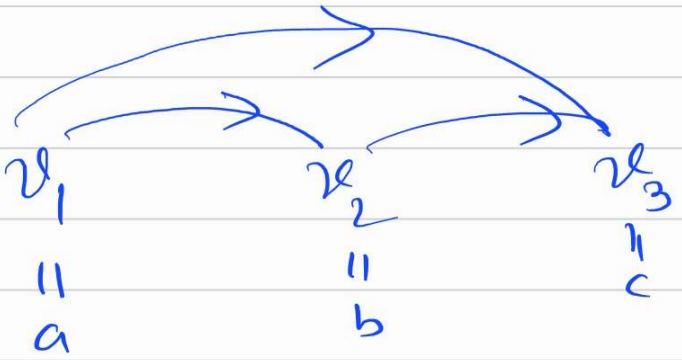
Goal:-

Given a DAG,

you want to find a order on the vertices as $v_1, \ldots, v_n$ s.t. for every edge (v_i, v_j) in the graph $i < j$.

This ordering is called "Topological ordering" of a directed graph.





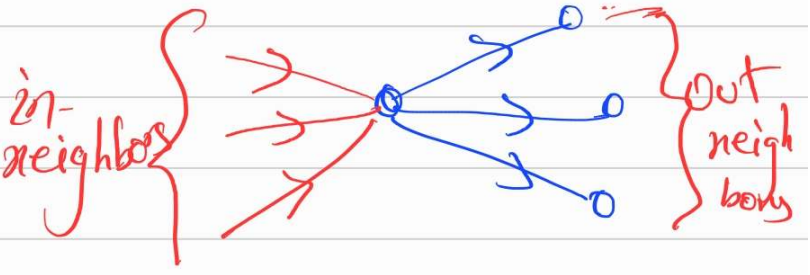
edges go from left to right.

Obs! A directed graph with topological ordering is in fact a DAG.

Goal :- To find a topological ordering.

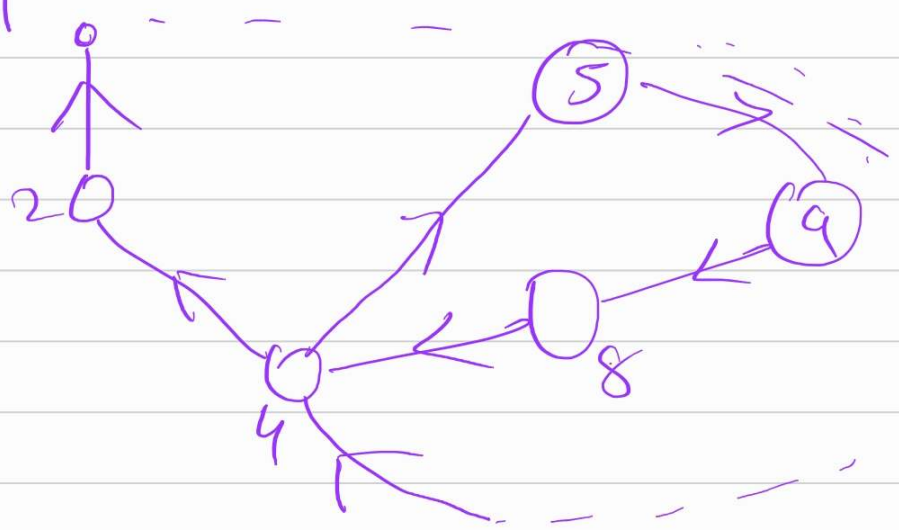
out-degree

in-degree



Lemma :- In a DAG, there always exist a node with in-degree 0.

Proof :- Suppose not.

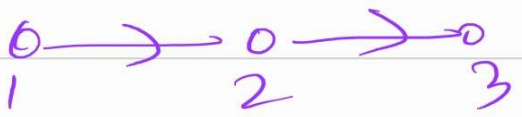


I continue this process for $n+1$ times.

↑ # nodes

then you have visited some vertex at least twice.

this gives the directed cycle ∎



Topological Ordering (G):

Find a vertex v with indegree 0 and make it the first vertex.

Recursively call Topological Ordering (G \ {v})

Proof of Correctness: induction on the number of vertices.

Runtime: Obvious upper bound. $O(n^2)$

But a better implementation.

will give $O(m+n)$.

Maintain : (1) Set of indegree 0 vertices

(2) Array listing indegree of vertices.

Post-Scriptum after the class:

In the algorithm to find all connected components maintain an array Discovered $[1, \ldots, n]$ with initial values set to False.

Set Discovered $[i]$ = True once this vertex has been explored.

To find a new vertex to start BFS scan through the array to find the first vertex that has not been discovered. Remember this index so that next scan starts from here.