EXAMPLE PROBLEM

For each pair of specimens i and j, they study them carefully side by side. If they're confident enough in their judgment, then they label the pair (i, j) either "same" (meaning they believe them both to come from the same species) or "different" (meaning they believe them to come from different species). They also have the option of rendering no judgment on a given pair, in which case we'll call the pair *ambiguous*.

So now they have the collection of n specimens, as well as a collection of m judgments (either "same" or "different") for the pairs that were not declared to be ambiguous. They'd like to know if this data is consistent with the idea that each butterfly is from one of species A or B. So more concretely, we'll declare the m judgments to be *consistent* if it is possible to label each specimen either A or B in such a way that for each pair (i, j) labeled "same," it is the case that i and j have the same label; and for each pair (i, j) labeled "different," it is the case that i and j have different labels. They're in the middle of tediously working out whether their judgments are consistent, when one of them realizes that you probably have an algorithm that would answer this question right away.

Give an algorithm with running time $O(m + n)$ that determines whether the m judgments are consistent.

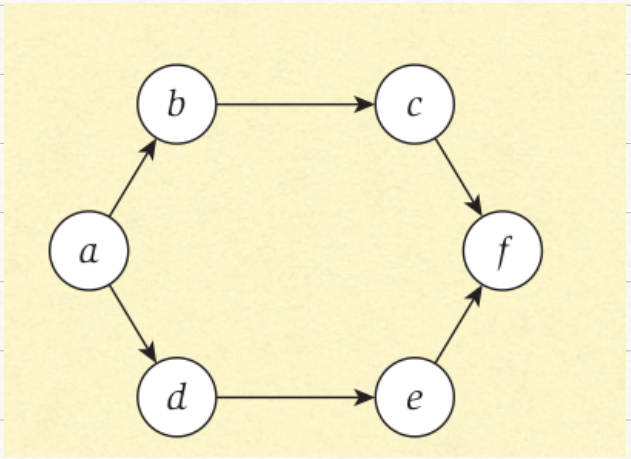
5. A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

6. We have a connected graph $G = (V, E)$, and a specific vertex $u \in V$. Suppose we compute a depth-first search tree rooted at u, and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at u, and obtain the same tree T. Prove that $G = T$. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T.)

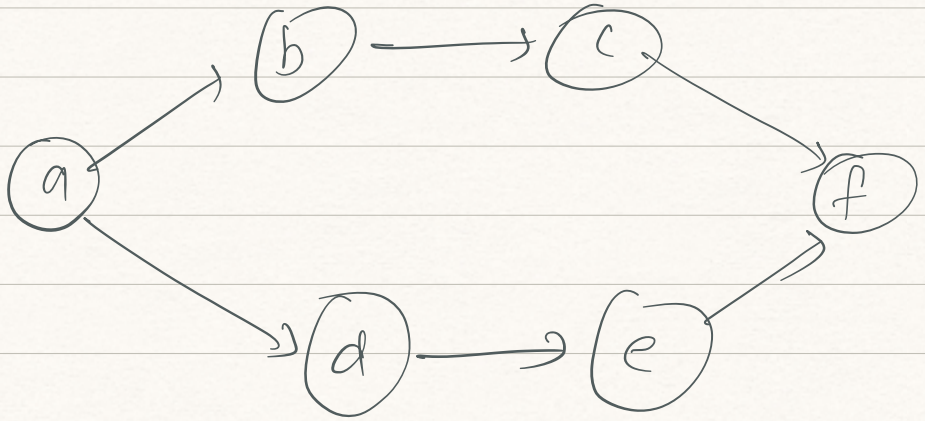
7. Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around (actually, as their human owners move around), they define a graph at any point in time as follows: there is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

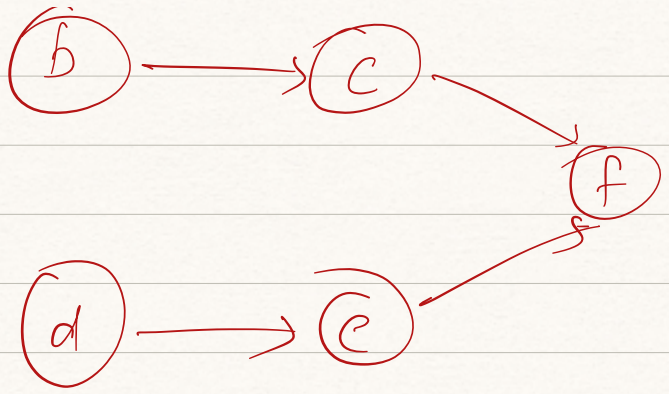
They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy

1. Consider the directed acyclic graph G in Figure 3.10. How many topological orderings does it have?



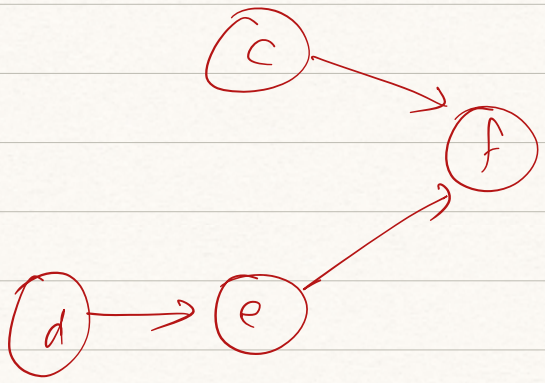
- This problem is very straight forward as we have already discussed in previous video.





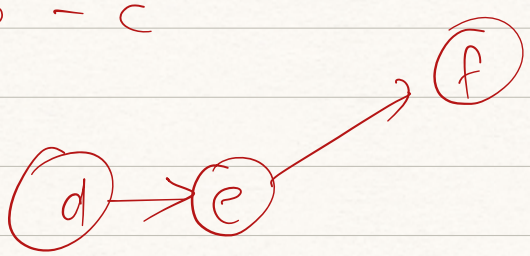
1. a - b
2. a - d

if a - b



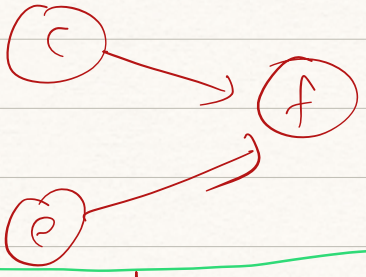
3. a - b - c
4. a b - d

3. a - b - c



a - b - c - d - e - f

4 a - b - d



4 — 1 a - b - d - e - c - f

a - b - d - c - e - f

ii) a - d

⇒ find, similar 3 topological ordering for these combination

2. Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output one. (It should not output all cycles in the graph, just one of them.) The running time of your algorithm should be $O(m+n)$ for a graph with n nodes and m edges.

- We can take help of DFS or BFS kind of algorithm to check whether graph contains cycle or not.
- Let's run DFS from any node & we can start visiting nodes.
- If we encounter same node again, we can say that graph contains cycle.
- Complexity :
  - This would run with same time complexity as DFS.

Time Complexity : $O(m+n)$

3. The algorithm described in Section 3.6 for computing a topological ordering of a DAG repeatedly finds a node with no incoming edges and deletes it. This will eventually produce a topological ordering, provided that the input graph really is a DAG.

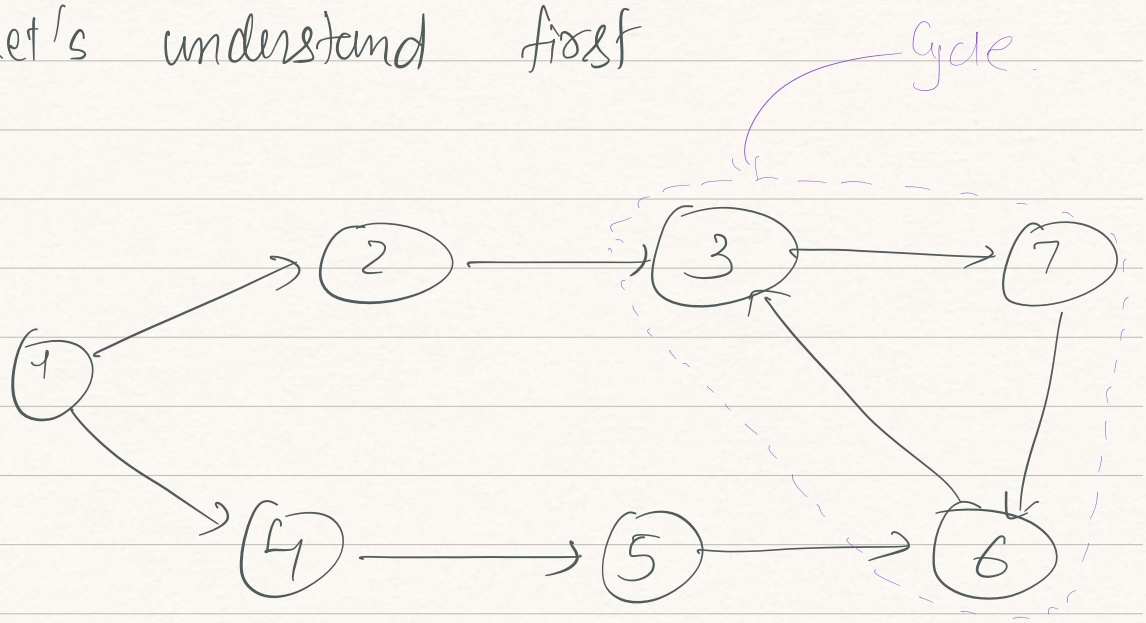
But suppose that we're given an arbitrary graph that may or may not be a DAG. Extend the topological ordering algorithm so that, given an input directed graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG; or (b) a cycle in G, thus establishing that G is not a DAG. The running time of your algorithm should be $O(m+n)$ for a directed graph with n nodes and m edges.

- We can run same algorithm as mentioned in text book

But

We will have slight modification

- Let's understand first

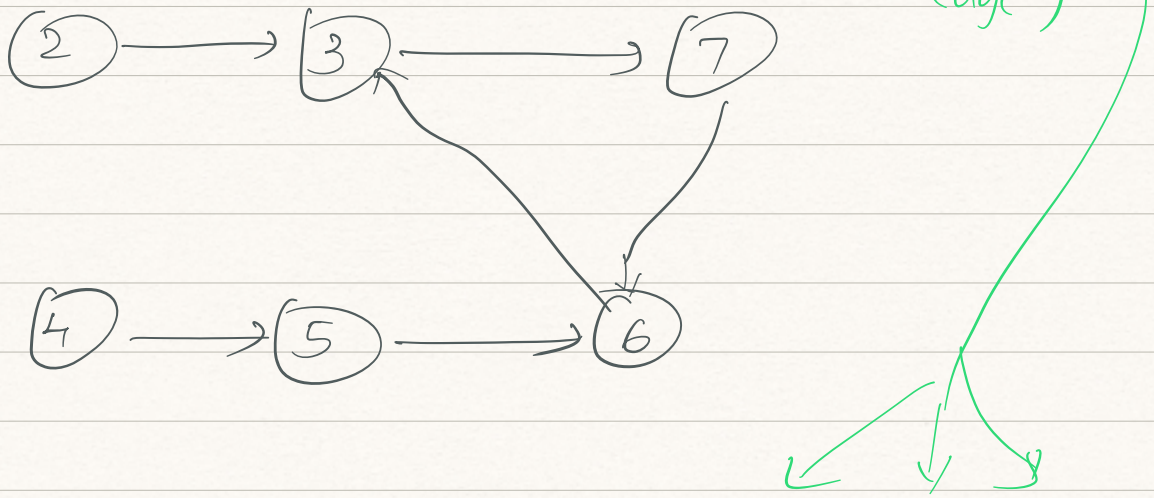


Visited

Step-1

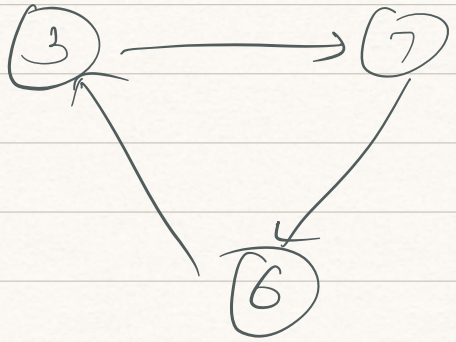
1	2	3	4	5	6	7

Remove - (1) (Node with no incoming edge)



- Subsequently remove (2), (4), (5)

- Graph



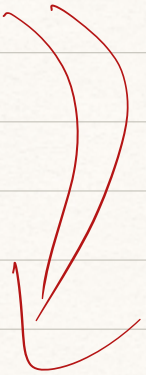
0	0	3	0	0	6	7

Visited

- We can see, there are few nodes, not visited yet

& even though

We cannot find a node with 0 incoming edge



there is a cycle.