Chapter 3 1 & 3

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

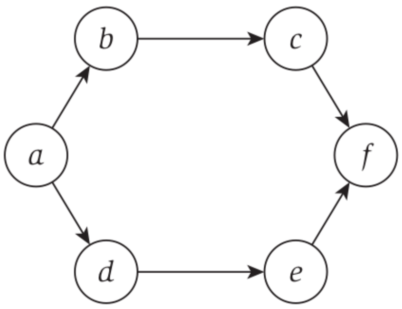
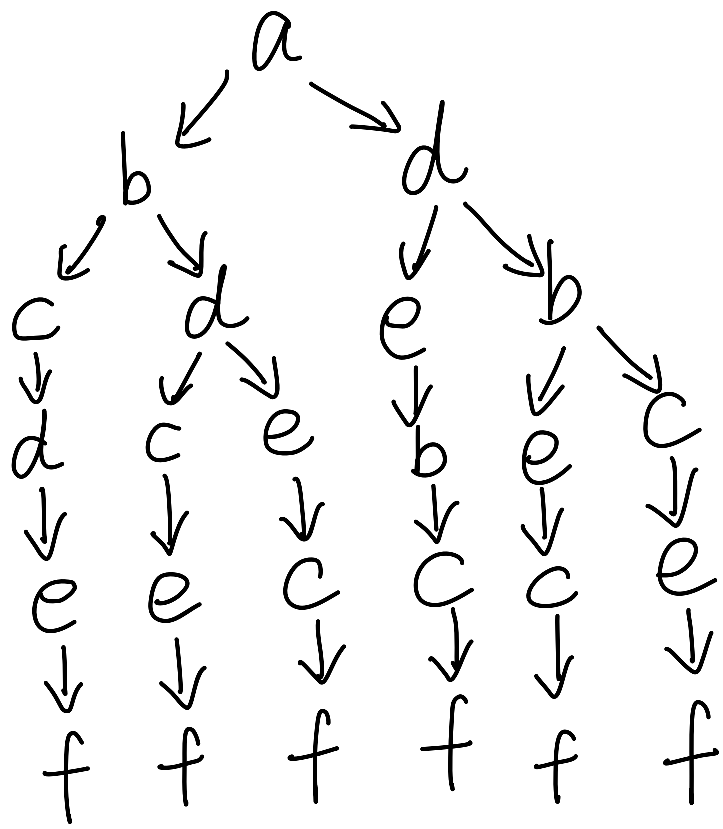


Figure 3.10 How many topological orderings does this graph have?



So there are total 6 topological ordering.

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.

To solve the problem:

① try to find a node v with no incoming edges and order it first or next, and delete v from graph and go back to ①

② if cannot find such node, we recursively choose an arbitrary edge delete it until we get a node with no incoming edge; let it as the new first node and delete it from graph and go to ①
also, record the last edge we deleted.

③ when finish, we'll get a topological order for DAG and we add the edge we deleted in (2) then get a cycle for graphs that is not DAG.

since ② is still $O(m+n)$ so my algorithm is $O(m+n)$