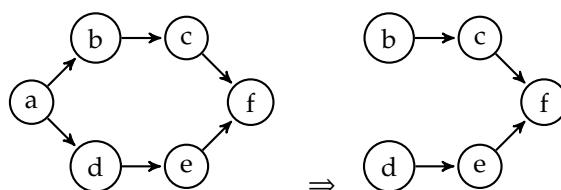


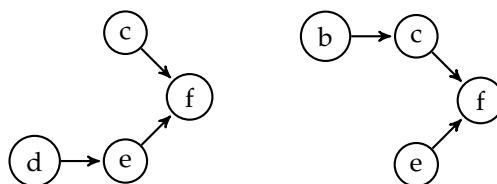
Assignment 3

Ch.3 - Ex.1

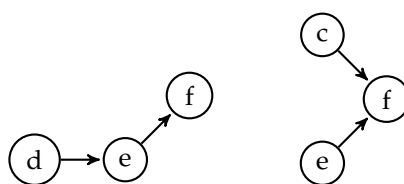
We start from the only node with a zero in degree, a . Remove it and add it to the first place of the topological ordering.



Let's remove another node in the second and third step. Now we have $a - b - ?$ or $a - d - ?$.



We take the first case as an example. We can either delete c first, then the left three can be only ordered as $d - e - f$; we can also first delete d , then use $c - e - f$ or $e - c - f$. There are three ordering for this subgraph, similarly, we find the other case has three ordering.



In summary, we have $2 \times 3 = 6$ possible topological orderings, listed as below.

$a - b - c - d - e - f$

$a - b - d - c - e - f$

$a - b - d - e - c - f$

$a - d - b - c - e - f$

$a - d - b - e - c - f$

$a - d - e - b - c - f$

Ch.3 - Ex.3

Algorithm 1 Extend Topological Sort

```

1: List<Node> topologicalOrdered
2: procedure EXTTOPOLOGICAL(GRAPH G)
3:   if G is empty then                                     ▶ prior recursions have cleared the DAG
4:     Return the topological order (list in line 1)
5:   Find a node  $v$  with  $\deg^-(v) = 0$  from G
6:   if  $v$  exists then
7:     Add  $v$  to list topologicalOrdered
8:     Delete  $v$  from G
9:     Recursively compute a topological ordering of  $G - \{v\}$ 
10:  else                                                     ▶ all nodes in G have income edge, G must contains cycle (Algorithm Design, pp. 100)
11:    Create a list to store the cycle Randomly pick a node  $u$  from G
12:    while  $u$  is not visited do                               ▶  $O(1)$  each loop
13:      Mark  $u$  as visited
14:      Add  $u$  to the list (stated in line 9)
15:       $u \leftarrow$  the first node in this node's income nodes' list
16:    We've find a whole cycle and store in the list, now return the answer

```

The above algorithm is modified base on the topological sort algorithm. If G is a DAG, then certainly it will never enter the *else* branch and runs as the normal topological sort algorithm, taking $O(m + n)$. If G is not a DAG, it may first run this algorithm as the normal condition, when it first meets a cycle (any node in a cycle), it takes $O(1)$ to check each node in the cycle, since the cycle is not larger than n nodes, the total cost to find the cycle is $O(n)$. With a prior part using $O(m + n)$, the total time complexity is still $O(m + n)$.