8.8 POSETS; TOPOLOGICAL SORTING

Suppose S is a graph such that each node v_i of S represents a task and each edge (u, v) means that the completion of the task u is a prerequisite for starting the task v. Suppose such a graph S contains a cycle, such as

$$P = (u, v, w, u)$$

This means that we cannot begin v until completing u, we cannot begin w until completing v and we cannot begin u until completing w. Thus we cannot complete any of the tasks in the cycle. Accordingly, such a graph S, representing tasks and a prerequisite relation, cannot have cycles.

Suppose S is a graph without cycles. Consider the relation $<$ on S defined as follows:

$$u < v \qquad \text{if there is a path from } u \text{ to } v$$

This relation has the following three properties:

(1) For each element u in S, we have $u \not< u$. (Irreflexivity.)
(2) If $u < v$, then $v \not< u$. (Asymmetry.)
(3) If $u < v$ and $v < w$, then $u < w$. (Transitivity.)

Such a relation $<$ on S is called a *partial ordering* of S, and S with such an ordering is called a *partially ordered set*, or *poset*. Thus a graph S without cycles may be regarded as a partially ordered set.

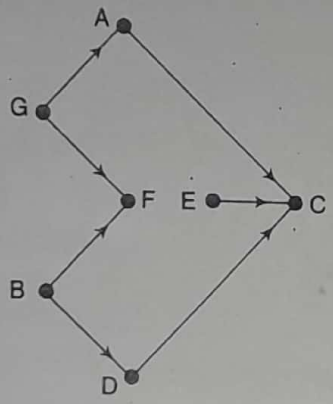
On the other hand, suppose S is a partially ordered set with the partial ordering denoted by $<$. Then S may be viewed as a graph whose nodes are the elements of S and whose edges are defined as follows:

(u, v) is an edge in S if $u < v$

Furthermore, one can show that a partially ordered set S, regarded as a graph, has no cycles.

Example 8.9

Let S be the graph in Fig. 8.15. Observe that S has no cycles. Thus S may be regarded as a partially ordered set. Note that G < C, since there is a path from G to C. Similarly, B < F and B < C. On the other hand, B ≮ A, since there is no path from B to A. Also, A ≮ B.



(a)

Adjacency lists	
A:	C
B:	D, F
C:	
D:	C
E:	C
F:	
G:	A, F

(b)

Fig. 8.15

Topological Sorting

Let S be a directed graph without cycles (or a partially ordered set). A topological sort T of S is a linear ordering of the nodes of S which preserves the original partial ordering of S. That is: *If $u < v$ in S (i.e., if there is a path from u to v in S), then u comes before v in the linear ordering T.* Figure 8.16 shows two different topological sorts of the graph S in Fig. 8.15. We have included the edges in Fig. 8.16 to indicate that they agree with the direction of the linear ordering.

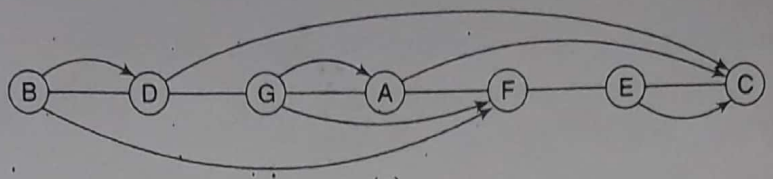
The following is the main theoretical result in this section.

Proposition 8.4

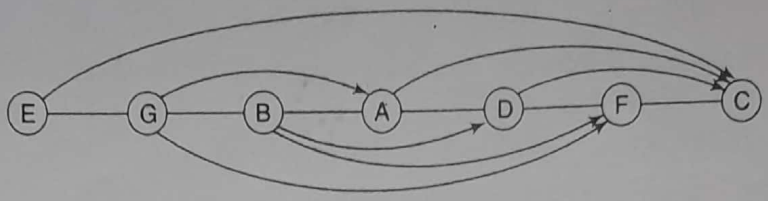
Let S be a finite directed graph without cycles or a finite partially ordered set. Then there exists a topological sort T of the set S.

Note that the proposition states only that a topological sort exists. We now give an algorithm which will find such a topological sort.

The main idea behind our algorithm to find a topological sort T of a graph S without cycles is that any node N with zero indegree, i.e., without any predecessors, may be chosen as the first element in the sort T. Accordingly, our algorithm will repeat the following two steps until the graph S is empty:



(a)



(b)

Fig. 8.16

(1) Finding a node N with zero indegree
(2) Deleting N and its edges from the graph *S*

The order in which the nodes are deleted from the graph *S* will use an auxiliary array QUEUE which will temporarily hold all the nodes with zero indegree. The algorithm also uses a field INDEG such that INDEG(N) will contain the current indegree of the node N. The algorithm follows.

Algorithm C: This algorithm finds a topological sort T of a graph S without cycles.

1. Find the indegree INDEG(N) of each node N of S. (This can be done by traversing each adjacency list as in Problem 8.15.)
2. Put in a queue all the nodes with zero indegree.
3. Repeat Steps 4 and 5 until the queue is empty.
4. Remove the front node N of the queue (by setting FRONT := FRONT + 1).
5. Repeat the following for each neighbor M of the node N:
   (a) Set INDEG(M) := INDEG(M) – 1.
   [This deletes the edge from N to M.]
   (b) If INDEG(M) = 0, then: Add M to the rear of the queue.
   [End of loop.]
   [End of Step 3 loop.]
6. Exit.

Example 8.10

Consider the graph *S* in Fig. 8.15(a). We apply our Algorithm C to find a topological sort *T* of the graph *S*. The steps of the algorithm follow.

1. Find the indegree INDEG(N) of each node N of the graph S. This yields:

INDEG(A) = 1 INDEG(B) = 0 INDEG(C) = 3 INDEG(D) = 1
INDEG(E) = 0 INDEG(F) = 2 INDEG(G) = 0

[This can be done as in Problem 8.15.]

2. Initially add to the queue each node with zero indegree as follows:

FRONT = 1, REAR = 3, QUEUE: B, E, G

3a. Remove the front element B from the queue by setting FRONT := FRONT + 1, as follows:

FRONT = 2, REAR = 3 QUEUE: B, E, G

3b. Decrease by 1 the indegree of each neighbor of B, as follows:

INDEG(D) = 1 − 1 = 0 and INDEG(F) = 2 − 1 = 1

[The adjacency list of B in Fig. 8.15(b) is used to find the neighbors D and F of the node B.] The neighbor D is added to the rear of the queue, since its indegree is now zero:

FRONT = 2, REAR = 4 QUEUE: B, E, G, D

[The graph S now looks like Fig. 8.17(a), where the node B and the edges from B have been deleted, as indicated by the dotted lines.]

4a. Remove the front element E from the queue by setting FRONT := FRONT + 1, as follows:

FRONT = 3, REAR = 4 QUEUE: B, E, G, D

4b. Decrease by 1 the indegree of each neighbor of E, as follows:

INDEG(C) = 3 − 1 = 2

[Since the indegree is nonzero, QUEUE is not changed. The graph S now looks like Fig. 8.17(b), where the node E and its edge have been deleted.]

5a. Remove the front element G from the queue by setting FRONT := FRONT + 1, as follows:

FRONT = 4, REAR = 4 QUEUE: B, E, G, D

5b. Decrease by 1 the indegree of each neighbor of G, as follows:

INDEG(A) = 1 − 1 = 0 and INDEG(F) = 1 − 1 = 0

Both A and F are added to the rear of the queue, as follows:

FRONT = 4, REAR = 6 QUEUE: B, E, G, D, A, F

[The graph S now looks like Fig. 8.17(c), where G and its two edges have been deleted.]

6a. Remove the front element D from the queue by setting FRONT := FRONT + 1, as follows:

FRONT = 5, REAR=6 QUEUE: B, E, G, D, A, F

6b. Decrease by 1 the indegree of each neighbor of D, as follows:

INDEG(C) = 2 − 1 = 1

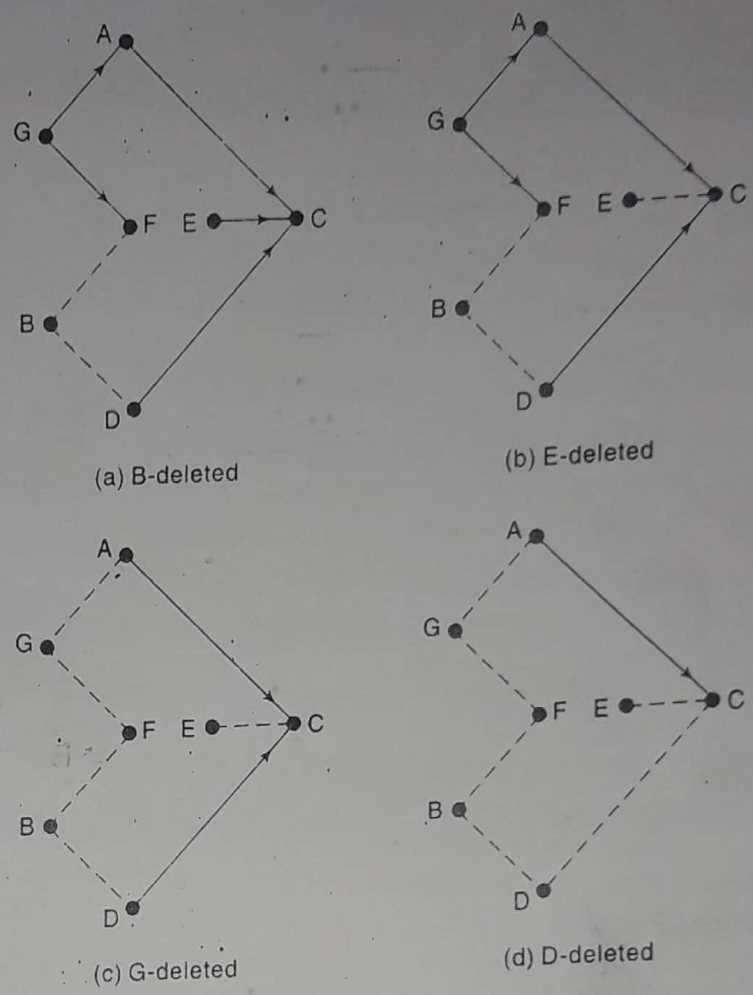


Fig. 8.17

[Since the indegree is nonzero, QUEUE is not changed. The graph S now looks like Fig. 8.17(d), where D and its edge have been deleted.]

7a. Remove the front element A from the queue by setting FRONT := FRONT + 1, as follows:

FRONT = 6, REAR = 6 QUEUE: B, E, G, D, A, F

7b. Decrease by 1 the indegree of each neighbor of A, as follows:

INDEG(C) = 1 – 1 = 0

Add C to the rear of the queue, since its indegree is now zero:

FRONT = 6, REAR = 7 QUEUE: B, E, G, D, A, F, C

8a. Remove the front element F from the queue by setting FRONT := FRONT + 1, as follows:

FRONT = 7, REAR = 7 QUEUE: B, E, G, D, A, F, C

8b. The node F has no neighbors, so no change takes place.

9a. Remove the front element C from the queue by setting FRONT := FRONT + 1 as follows:

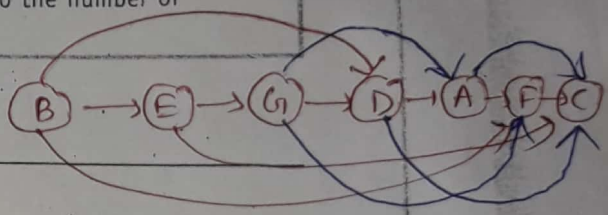
FRONT = 8, REAR = 7 QUEUE: B, E, G, D, A, F, C

9b. The node C has no neighbors, so no other changes take place.

The queue now has no front element, so the algorithm is completed. The elements in the array QUEUE give the required topological sort T of S as follows:

T: B, E, G, D, A, F, C

The algorithm could have stopped in Step 7b, where REAR is equal to the number of nodes in the graph S.



SOLVED PROBLEMS

Graph Terminology

8.1 Consider the (undirected) graph G in Fig. 8.18. (a) Describe G formally in terms of its set V of nodes and its set E of edges. (b) Find the degree of each node.

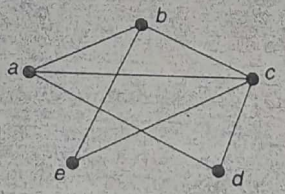


Fig. 8.18

(a) There are 5 nodes, a, b, c, d and e; hence $V = \{a, b, c, d, e\}$. There are 7 pairs $[x, y]$ of nodes such that node x is connected with node y; hence

$$E = \{[a, b], [a, c], [a, d], [b, c], [b, e], [c, d], [c, e]\}$$

(b) The degree of a node is equal to the number of edges to which it belongs; for example, $\deg(a) = 3$, since a belongs to three edges, $[a, b]$, $[a, c]$ and $[a, d]$. Similarly, $\deg(b) = 3$, $\deg(c) = 4$, $\deg(d) = 2$ and $\deg(e) = 2$.

8.2 Consider the multigraphs in Fig. 8.19. Which of them are (a) connected; (b) loop-free (i.e., without loops); (c) graphs?