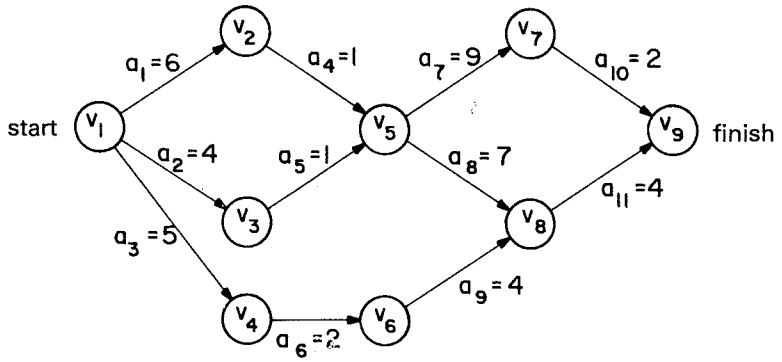
Critical Paths

An activity network closely related to the AOV network is the activity on edge or AOE network. The tasks to be performed on a project are represented by directed edges. Vertices in the network represent events. Events signal the completion of certain activities. Activities represented by edges leaving a vertex cannot be started until the event at that vertex has occurred. An event occurs only when all activities entering it have been completed. Figure 6.33(a) is an AOE network for a hypothetical project with 11 tasks or activities $a_1, \ldots, a_{11}$. There are nine events $v_1, v_2, \ldots, v_9$. The events v_1 and v_9 may be interpreted as "start project" and "finish project," respectively. Figure 6.33(b) gives interpretations for some of the nine events. The number associated with each activity is the time needed to perform that activity. Thus, activity a_1 requires 6 days while a_{11} requires 4 days. Usually, these times are only estimates. Activities a_1, a_2, and a_3 may be carried out concurrently after the start of the project. a_4, a_5, and a_6 cannot be started until events v_2, v_3, and v_4, respectively, occur. a_7 and a_8 can be carried out concurrently after the occurrence of event v_5 (i.e., after a_4 and a_5 have been completed). In case additional ordering constraints are to be put on the activities, dummy activities whose time is zero may be introduced. Thus, if we desire that activities a_7 and a_8 not start until both events v_5 and v_6 have occurred, a dummy activity a_{12} represented by an



(a) AOE Network. Activity Graph of a Hypothetical Project.

event	interpretation
v_1	start of project
v_2	completion of activity a_1
v_5	completion of activities a_4 and a_5
v_8	completion of activities a_8 and a_9
v_9	completion of project

(b) Interpretation for Some of the Events in the Activity Graph of (a).

Figure 6.33 An AOE network

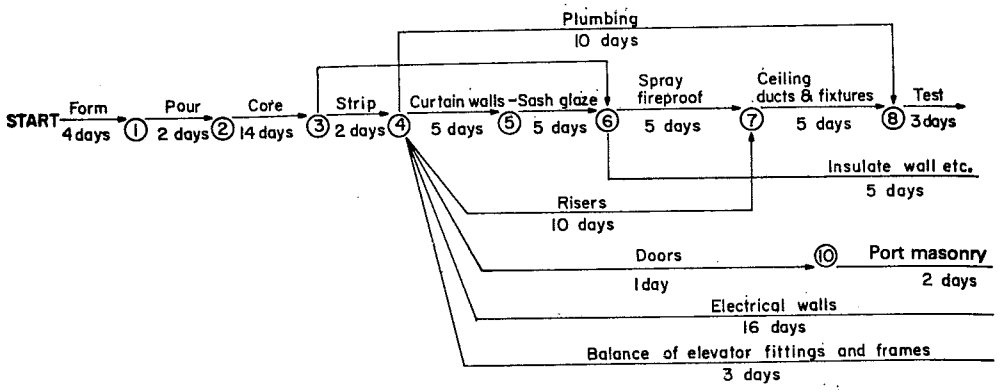
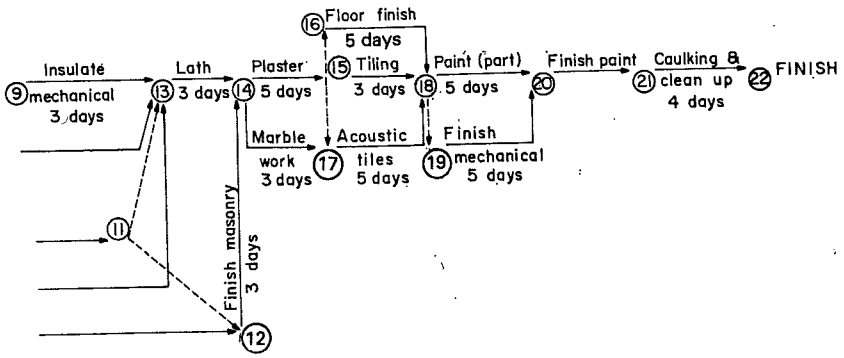


Figure 6.34 AOE network for the construction of a typical floor in a multistory building [Engineering News-Record (McGraw-Hill Book Company, Inc., Jan. 26, 1961).]

edge $<v_6,v_5>$ may be introduced. Activity networks of the AOE type have proved very useful in the performance evaluation of several types of projects. This evaluation includes determining such facts about the project as what is the least amount of time in which the project may be completed (assuming there are no cycles in the network); which activities should be speeded up in order to reduce completion time; etc.

Several sophisticated techniques such as PERT (performance evaluation and review technique), CPM (critical path method), and RAMPS (resource allocation and multiproject scheduling) have been developed to evaluate network models of projects. CPM was originally developed in connection with maintenance and construction projects. Figure 6.34 shows a network used by the Perinia Corporation of Boston in 1961 to model the construction of a floor in a multistory building. PERT was originally designed for use in the development of the Polaris missile system.

Since the activities in an AOE network can be carried out in parallel the minimum time to complete the project is the length of the longest path from the start vertex to the finish vertex (the length of a path is the sum of the times of activities on this path). A path of longest length is a *critical path*. The path v_1,v_2,v_5,v_7,v_9 is a critical path in the network of Figure 6.33(a). The length of this critical path is 18. A network may have more than one critical path (the path v_1,v_2,v_5,v_8,v_9 is also critical). The *earliest time* an event v_i can occur is the length of the longest path from the start vertex v_1 to the vertex v_i. The earliest time event v_5 can occur is 7. The earliest time an event can occur determines the *earliest start time* for all activities represented by edges leaving that vertex. Denote this time by $e(i)$ for activity a_i. For example, $e(7)=e(8)=7$. For every



activity a_i, we may also define the *latest time*, $l(i)$, an activity may start without increasing the project duration (i.e., length of the longest path from start to finish). In Figure 6.33(a) we have $e(6)=5$ and $l(6)=8$, $e(8)=7$ and $l(8)=7$. All activities for which $e(i)=l(i)$ are called *critical activities*. The difference $l(i)-e(i)$ is a measure of the criticality of an activity. It gives the time by which an activity may be delayed or slowed without increasing the total time needed to finish the project. If activity a_6 is slowed down to take 2 extra days, this will not affect the project finish time. Clearly, all activities on a critical path are critical and speeding noncritical activities will not reduce the project duration.

The purpose of critical path analysis is to identify critical activities so that resources may be concentrated on these activities in an attempt to reduce project finish time. Speeding a critical activity will not result in a reduced project length unless that activity is on all critical paths. In Figure 6.33(a) the activity a_{11} is critical but speeding it up so that it takes only 3 days instead of 4 does not reduce the finish time to 17 days. This is so because there is another critical path v_1,v_2,v_5,v_7,v_9 that does not contain this activity. The activities a_1 and a_4 are on all critical paths. Speeding a_1 by 2 days reduces the critical path length to 16 days. Critical path methods have proved very valuable in evaluating project performance and identifying bottlenecks.

Critical path analysis can also be carried out with AOV networks. The length of a path would now be the sum of the activity times of the vertices on that path. For each activity or vertex, we could analogously define the quantities $e(i)$ and $l(i)$. Since the activity times are only estimates, it is necessary to re-evaluate the project during several

stages of its completion as more accurate estimates of activity times become available. These changes in activity times could make previously noncritical activities critical and vice versa. Before ending our discussion on activity networks, let us design an algorithm to evaluate $e(i)$ and $l(i)$ for all activities in an AOE network. Once these quantities are known, then the critical activities may be easily identified. Deleting all noncritical activities from the AOE network, all critical paths may be found by just generating all paths from the start to finish vertex (all such paths will include only critical activities and so must be critical, and since no noncritical activity can be on a critical path, the network with noncritical activities removed contains all critical paths present in the original network).

In obtaining the $e(i)$ and $l(i)$ for the activities of an AOE network, it is easier to first obtain the earliest event occurrence time, $ee[j]$, and latest event occurrence time, $le[j]$, for all events, j, in the network. Then if activity a_i is represented by edge $<k,l>$, we can compute $e(i)$ and $l(i)$ from the formulas

$$e(i)=ee[k]$$

and (6.1)

$$l(i)=le[l]-\text{duration of activity } a_i$$

The times $ee[j]$ and $le[j]$ are computed in two stages: a forward stage and a backward stage. During the forward stage we start with $ee[1]=0$ and compute the remaining early start times, using the formula

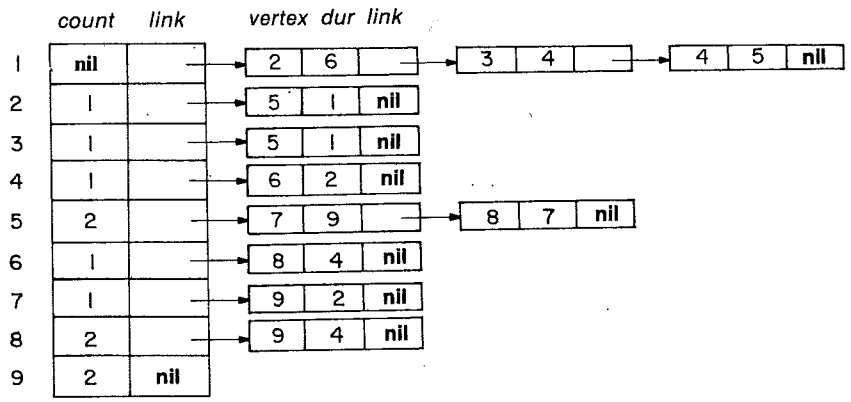
$$ee[j]=\max_{i \in P(j)}\{ee[i] + \text{duration of} <i,j>\} \tag{6.2}$$

where $P(j)$ is the set of all vertices adjacent to vertex j. In case this computation is carried out in topological order, the early start times of all predecessors of j would have been computed prior to the computation of $ee[j]$. The algorithm to do this is obtained easily from algorithm *TopologicalOrder* by inserting the step

```
if ee[k] < ee[j] + ptr↑.dur
    then ee[k] := ee[j] + ptr↑.dur;
```

between lines 38 and 39. It is assumed that the array *ee* is initialized to zero and that *dur* is another field in the adjacency list nodes which contains the activity duration. This modification results in the evaluation of Eq. (6.2) in parallel with the generation of a topological order. $ee(j)$ is updated each time the $ee(i)$ of one of its predecessors is known (i.e., when i is ready for output). The step **writeln**(j) of line 33 may be omitted.

To illustrate the working of the modified *TopologicalOrder* algorithm let us try it out on the network of Figure 6.33(a). The adjacency lists for the network are shown in Figure 6.35(a). The order of nodes on these lists determines the order in which vertices will be considered by the algorithm. At the outset the early start time for all vertices is 0, and the start vertex is the only one in the stack. When the adjacency list for this vertex is



(a) Adjacency Lists for Figure 6.34(a)

ee	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	stack
initial	0	0	0	0	0	0	0	0	0	1
output v_1	0	6	4	5	0	0	0	0	0	4 3 2
output v_4	0	6	4	5	0	7	0	0	0	6 3 2
output v_6	0	6	4	5	0	7	0	11	0	3 2
output v_3	0	6	4	5	5	7	0	11	0	2
output v_2	0	6	4	5	7	7	0	11	0	5
output v_5	0	6	4	5	7	7	16	14	0	8 7
output v_8	0	6	4	5	7	7	16	14	16	7
output v_7	0	6	4	5	7	7	16	14	18	9
output v_9										

(b) Computation of *ee*

Figure 6.35 Action of modified topological order

processed, the early start time of all vertices adjacent from v_1 is updated. Since vertices 2, 3 and 4 are now in the stack, all their predecessors have been processed and Eq. (6.2) evaluated for these three vertices. *ee* [6] is the next one determined. When vertex v_6 is being processed, *ee* [8] is updated to 11. This, however, is not the true value for *ee* [8] since Eq. (6.2) has not been evaluated over all predecessors of v_8 (v_5 has not yet been considered). This does not matter since v_8 cannot get stacked until all its predecessors have been processed. *ee* [5] is next updated to 5 and finally to 7. At this point *ee* [5] has been determined as all the predecessors of v_5 have been examined. The values of *ee* [7] and *ee* [8] are next obtained. *ee* [9] is ultimately determined to be 18, the length of a critical path. You may readily verify that when a vertex is put into the stack its early time has been correctly computed. The insertion of the new statement does not change the asymptotic computing time; it remains $O(e+n)$.

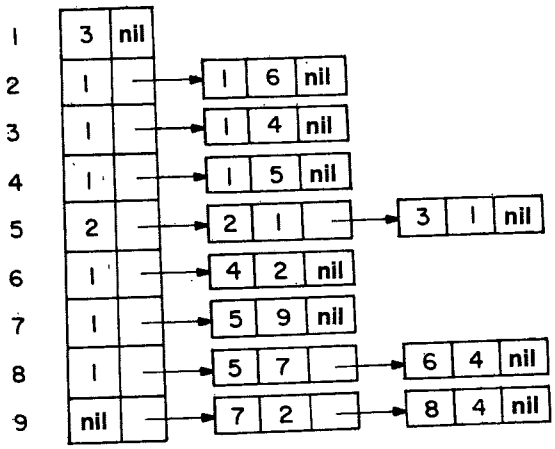
In the backward stage the values of *le* [*i*] are computed using a procedure analogous to that used in the forward stage. We start with *le* [*n*]=*ee* [*n*] and use the equation

$$le[j] = \min_{i \in S(j)} \{le[i] - \text{duration of } <j,i>\} \tag{6.3}$$

where $S(j)$ is the set of vertices adjacent from vertex *j*. The initial values for *le* [*i*] may be set to *ee* [*n*]. Basically, Eq. (6.3) says that if $<j,i>$ is an activity and the latest start time for event *i* is *le* [*i*], then event *j* must occur no later than *le* [*i*] – duration of $<j,i>$. Before *le* [*j*] can be computed for some event *j*, the latest event time for all successor events (i.e., events adjacent from *j*) must be computed. These times can be obtained in a manner identical to the computation of the early times by using inverse adjacency lists and inserting the step *le* [*k*]:=min{*le* [*k*],*le* [*j*]–*ptr*↑.*dur*} at the same place as before in algorithm *TopologicalOrder*. The *count* field of a head node will initially be the out-degree of the vertex.

Figure 6.36 describes the process on the network of Figure 6.33(a). In case the forward stage has already been carried out and a topological ordering of the vertices obtained, then the values of *le* [*i*] can be computed directly, using Eq. (6.3), by performing the computations in the reverse topological order. The topological order generated in Figure 6.35(b) is v_1, v_4, v_6, v_3, v_2, v_5, v_8, v_7, v_9. We may compute the values of *le* [*i*] in the order 9, 7, 8, 5, 2, 3, 6, 4, 1 as all successors of an event precede that event in this order. In practice, one would usually compute both *ee* and *le*. The procedure would then be to compute *ee* first using algorithm *TopologicalOrder* modified as discussed for the forward stage and to then compute *le* directly from Eq. (6.3) in reverse topological order.

Using the values of *ee* (Figure 6.35) and of *le* (Figure 6.36 and Eq. (6.1)) we may compute the early and late times $e(i)$ and $l(i)$ and the degree of criticality of each task. Figure 6.37 gives the values. The critical activities are a_1, a_4, a_7, a_8, a_{10}, and a_{11}. Deleting all noncritical activities from the network we get the directed graph of Figure 6.38. All paths from v_1 to v_9 in this graph are critical paths and there are no critical paths in the original network that are not paths in the graph of Figure 6.38.



(a) Inverted Adjacency Lists for AOE Network of Figure 6.34(a).

le	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	stack
initial	18	18	18	18	18	18	18	18	18	9
output v_9	18	18	18	18	18	18	16	14	18	8, 7
output v_8	18	18	18	18	7	10	16	14	18	6, 7
output v_6	18	18	18	18	7	10	16	14	18	4, 7
output v_4	3	18	18	8	7	10	16	14	18	7
output v_7	3	18	18	8	7	10	16	14	18	5
output v_5	3	6	6	8	7	10	16	14	18	3, 2
output v_3	2	6	6	8	7	10	16	14	18	2
output v_2	0	6	6	8	7	10	16	14	18	1

(b) Computation of *topologicalorder* Modified to Compute Latest Event Times.

$$
\begin{aligned}
le[9] &= ee[9] = 18 \\
le[7] &= \min\{le[9] - 2\} = 16 \\
le[8] &= \min\{le[9] - 4\} = 14 \\
le[5] &= \min\{le[7] - 9,\ le[8] - 7\} = 7 \\
le[2] &= \min\{le[5] - 1\} = 6 \\
le[3] &= \min\{le[5] - 1\} = 6 \\
le[6] &= \min\{le[8] - 4\} = 10 \\
le[4] &= \min\{le[6] - 2\} = 8 \\
le[1] &= \min\{le[2] - 6,\ le[3] - 4,\ le[4] - 5\} = 0
\end{aligned}
$$

(c) Computation of *le* Directly from Equation (6.3) Using a Reverse Topological Order.

Figure 6.36 Computing *le* for AOE network of Figure 6.33(a)

activity	e	l	$l - e$
a_1	0	0	0
a_2	0	2	2
a_3	0	3	3
a_4	6	6	0
a_5	4	6	2
a_6	5	8	3
a_7	7	7	0
a_8	7	7	0
a_9	7	10	3
a_{10}	16	16	0
a_{11}	14	14	0

Figure 6.37 Early, late, and criticality values

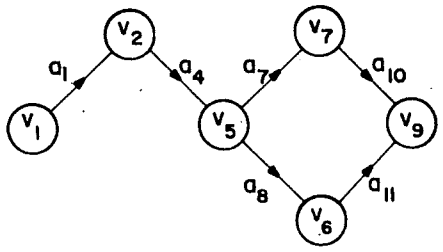


Figure 6.38 Graph obtained after deleting all noncritical activities

As a final remark on activity networks we note that the algorithm *Topological Order* detects only directed cycles in the network. There may be other flaws, such as vertices not reachable from the start vertex (Figure 6.39). When a critical path analysis is carried out on such networks, there will be several vertices with $ee[i] = 0$. Since all activity times are assumed >0, only the start vertex can have $ee[i] = 0$. Hence, critical path analysis can also be used to detect this kind of fault in project planning.

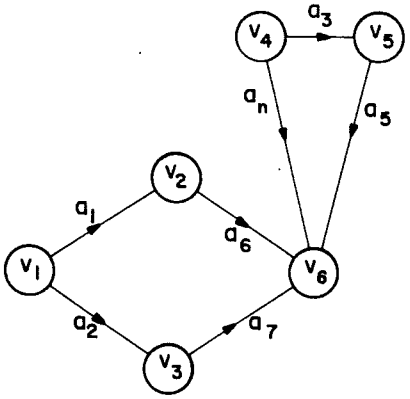


Figure 6.39 AOE network with some nonreachable activities

6.5 ENUMERATING ALL PATHS

In Section 6.3, we looked at the problem of finding a shortest path between two vertices. In this section we will be concerned with listing all possible simple paths between two vertices in order of nondecreasing path length. Such a listing could be of use, for example, in a situation where we are interested in obtaining the shortest path that satisfies some complex set of constraints. One possible solution to such a problem would be to generate in nondecreasing order of path length all paths between the two vertices. Each path generated could be tested against the other constraints and the first path satisfying all these constraints would be the path we were looking for.

Let $G=(V,E)$ be a digraph with n vertices. Let v_1 be the source vertex and v_n the destination vertex. We shall assume that every edge has a positive cost. Let $p_1 = r[0],r[1], \ldots, r[k]$ be a shortest path from v_1 to v_n; i.e., p_1 starts at $v_{r[0]} = v_1$, goes to $v_{r[1]}$ and then to $v_{r[2]}, \ldots, v_n = v_{r[k]}$. If P is the set of all simple v_1 to v_n paths in G, then it is easy to see that every path in $P - \{p_1\}$ differs from p_1 in exactly one of the following k ways:

(1) It contains the edges $(r[1],r[2]), \ldots, (r[k-1],r[k])$ but not $(r[0],r[1])$
(2) It contains the edges $(r[2],r[3]), \ldots, (r[k-1],r[k])$ but not $(r[1],r[2])$

.
.
.

(k) It does not contain the edge $(r[k-1],r[k])$