8.1 An Introduction to Scheduling

We will now introduce the principal characters in any scheduling story.

- **The processors.** Every job requires workers. We will use the term **processors** to describe the "workers" who carry out the work. While the word *processor* may sound cold and impersonal, it does underscore an important point: processors need not be human beings. In scheduling, a processor could just as well be a robot, a computer, an automated teller machine, and so on. We will use N to represent the number of processors and $P_1, P_2, P_3, \ldots, P_N$ to denote the processors. We will assume throughout the chapter that $N \geq 2$ (scheduling a job with just one processor is trivial and not very interesting).
- **The tasks.** In every complex project there are individual pieces of work, often called "jobs" or "tasks." We will need to be a little more precise than that, however. We will define a **task** as an indivisible unit of work that (either by nature or by choice) cannot be broken up into smaller units. Thus, by definition a task cannot be shared—it is always *carried out by a single processor*. In general, we will use capital letters $A, B, C, \ldots,$ to represent the tasks, although in specific situations it is convenient to use abbreviations (such as WE for "wiring the electrical system" or PL for "plumbing").

  At a particular moment in time a task can be in one of four possible states: (1) *ineligible* (the task cannot be started because some of the prerequisites for the task have not yet been completed), (2) *ready* (the task has not been started but could be started at this time), (3) *in execution* (the task is being carried out by one of the processors), or (4) *completed*.
- **The processing times.** The **processing time** of a task is the amount of time, without interruption, required by *one processor* to execute that task. When dealing with human processors, there are many variables (ability, attitude, work ethic, etc.) that can affect the processing time of a task, and this adds another layer of complexity to an already complex situation. On the other hand, if we assume a "robotic" interpretation of the processors (either because they are indeed machines or because they are human beings trained to work in a very standardized and uniform way), then scheduling becomes somewhat more manageable.

  To keep things simple we will work under the following three assumptions:

  1. *Versatility:* Any processor can execute any task.
  2. *Uniformity:* The processing time for a task is the same regardless of which processor is executing the task.
  3. *Persistence:* Once a processor starts a task, it will complete it without interruption.

  Under the preceding assumptions, the concept of *processing time* for a task (we will sometimes call it the P-time) makes good sense—and we can conveniently incorporate this information by including it inside parentheses next to the name of the task. Thus, the notation $X(5)$ tells us that the task called X has a processing time of 5 units (be it minutes, hours, days, or any other unit of time) *regardless of which processor is assigned to execute the task.*
- **The precedence relations.** Precedence relations are formal restrictions on the order in which the tasks can be executed, much like those course prerequisites in the school catalog that tell you that you can't take course Y until you

70

have completed course X. In the case of tasks, these prerequisites are called **precedence relations.** A typical precedence relation is of the form *task X precedes task Y* (we also say X is *precedent* to Y), and it means that *task Y cannot be started until task X has been completed.* A precedence relation can be conveniently abbreviated by writing $X \rightarrow Y$, or described graphically as shown in Fig. 8-1(a). A single scheduling problem can have hundreds or even thousands of precedence relations, each adding another restriction on the scheduler's freedom.

At the same time, it also happens fairly often that there are no restrictions on the order of execution between two tasks in a project. When a pair of tasks X and Y have no precedence requirements between them (neither $X \rightarrow Y$ nor $Y \rightarrow X$), we say that the tasks are **independent.** When two tasks are independent, either one can be started before the other one, or they can both be started at the same time. Graphically, we can tell that two tasks are independent if there are no arrows connecting them [Fig. 8-1(b)].

Two final comments about precedence relations are in order. First, precedence relations are *transitive:* If $X \rightarrow Y$ and $Y \rightarrow Z$, then it must be true that $X \rightarrow Z$. In a sense, the last precedence relation is implied by the first two, and it is really unnecessary to mention it [Fig. 8-1(c)]. Thus, we will make a distinction between two types of precedence relations: *basic* and *implicit.* Basic precedence relations are the ones that come with the problem and that we must follow in the process of creating a schedule. If we do this, the implicit precedence relations will be taken care of automatically.

The second observation is that *precedence relations cannot form a cycle!* Imagine having to schedule the tasks shown in Fig. 8-1(d): X precedes Y, which precedes Z, which precedes W, which in turn precedes X. Clearly, this is logically impossible. From here on, we will always assume that there are no cycles of precedence relations among the tasks.

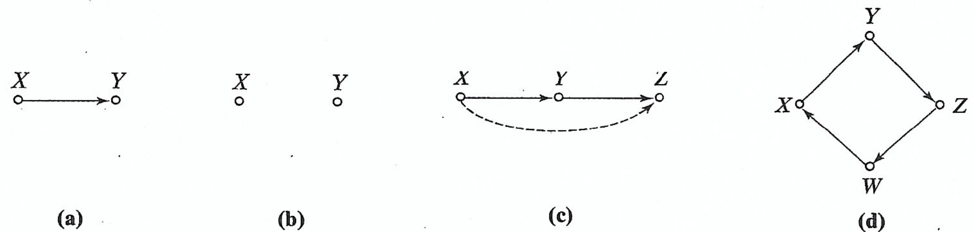
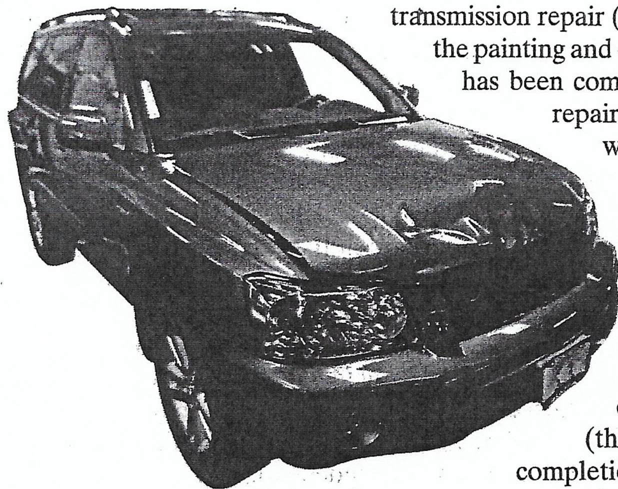


FIGURE 8-1 (a) X is precedent to Y. (b) X and Y are independent tasks. (c) When $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$ is implied. (d) These tasks cannot be scheduled because of the cyclical nature of the precedence relations.

Processors, tasks, processing times, and *precedence relations* are the basic ingredients that make up a scheduling problem. They constitute, in a manner of speaking, the hand that is dealt to us. But how do we play such a hand? To get a small inkling of what's to come, let's look at the following simple example.

EXAMPLE 8.1 REPAIRING A WRECK

Imagine that you just wrecked your car, but thank heavens you are OK, and the insurance company will pick up the tab. You take the car to the best garage in town, operated by the Tappet brothers Click and Clack (we'll just call them P_1 and P_2). The repairs on the car can be broken into four different tasks: (A) exterior body work (4 hours), (B) engine repairs (5 hours), (C) painting and exterior finish work (7 hours), and (D)



transmission repair (3 hours). The only precedence relation for this set of tasks is that the painting and exterior finish work cannot be started until the exterior body work has been completed ($A \rightarrow C$). The two brothers always work together on a repair project, but each takes on a different task (so they won't argue with each other). Under these assumptions, how should the different tasks be scheduled? Who should do what and when?

Even in this simple situation, there are many different ways to schedule the repair. Figure 8-2 shows several possible schedules, each one illustrated by means of a timeline. Figure 8-2(a) shows a schedule that is very inefficient. All the short tasks are assigned to one processor (P_1) and all the long tasks to the other processor (P_2)—obviously not a very clever strategy. Under this schedule, the project **finishing time** (the duration of the project from the start of the first task to the completion of the last task) is 12 hours. (We will use *Fin* to denote the project finishing time, so for this project we can write *Fin* = 12 hours.)

Figure 8-2(b) shows what looks like a much better schedule, but it violates the precedence relation $A \rightarrow C$ (as much as we would love to, we cannot start task C until task A is completed). On the other hand, if we force P_2 to be idle for one hour, waiting for the green light to start task C, we get a perfectly good schedule, shown in Fig. 8-2(c). Under this schedule the finishing time of the project is *Fin* = 11 hours.

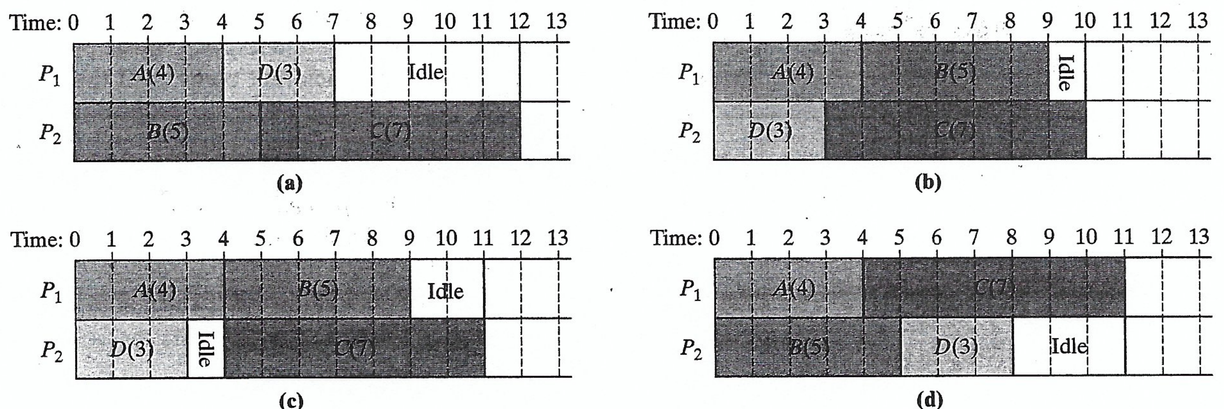


FIGURE 8-2 (a) A legal schedule with *Fin* = 12 hours. (b) An *illegal* schedule (the precedence relation $A \rightarrow C$ is violated). (c) An optimal schedule (*Opt* = 11 hours). (d) A different optimal schedule.

The schedule shown in Fig. 8-2(c) is an improvement over the first schedule. Can we do even better? No! No matter how clever we are and no matter how many processors we have at our disposal, the precedence relation $A(4) \rightarrow C(7)$ implies that 11 hours is a minimum barrier that we cannot break—it takes 4 hours to complete A, 7 hours to complete C, and *we cannot start C until A is completed!* Thus, the schedule shown in Fig. 8-2(c) is an **optimal schedule** and the finishing time of *Fin* = 11 hours is the **optimal finishing time.** (From now on we will use *Opt* instead of *Fin* when we are referring to the optimal finishing time.) Figure 8-2(d) shows a different optimal schedule with finishing time *Opt* = 11 hours. ■

As scheduling problems go, Example 8.1 was a fairly simple one. But even from this simple example, we can draw some useful lessons. First, notice that even though we had only four tasks and two processors, we were able to create several different schedules. The four we looked at were just a sampler—there are other possible schedules that we didn't bother to discuss. Imagine what would happen if we had hundreds of tasks and dozens of processors—the number of possible schedules to consider would be overwhelming. In looking for a good, or even an optimal, schedule, we are going to need a systematic way to sort through the many possibilities. In other words, we are going to need some good *scheduling algorithms.*

72

The second useful thing we learned in Example 8.1 is that when it comes to the finishing time of a project, there is an *absolute minimum* time that no schedule can break, no matter how good an algorithm we use or how many processors we put to work. In Example 8.1 this absolute minimum was 11 hours, and, as luck would have it, we easily found a schedule [actually two—Figs. 8-2(c) and (d)] with a finishing time to match it. Every project, no matter how simple or complicated, has such an absolute minimum (called the *critical time*) that depends on the processing times and precedence relations for the tasks and not on the number of processors used. We will return to the concept of critical time in Section 8.5.

To set the stage for a more formal discussion of scheduling algorithms, we will introduce the most important example of this chapter. While couched in what seems like science fiction terms, the situation it describes is not totally farfetched—in fact, this example is a simplified version of the types of scheduling problems faced by home builders in general.

EXAMPLE 8.2 BUILDING THAT DREAM HOME (IN MARS)

It is the year 2050, and several human colonies have already been established on Mars. Imagine that you accept a job offer to work in one of these colonies. What will you do about housing?

Like everyone else on Mars, you will be provided with a living pod called a Martian Habitat Unit (MHU). MHUs are shipped to Mars in the form of prefabricated kits that have to be assembled on the spot—an elaborate and unpleasant job if you are going to do it yourself. A better option is to use specialized "construction" robots that can do all the assembly tasks much more efficiently than human beings can. These construction robots can be rented by the hour at the local Rent-a-Robot outlet.

The assembly of an MHU consists of 15 separate tasks, and there are 17 different precedence relations among these tasks that must be followed. The tasks, their respective processing times, and their precedent tasks are all shown in Table 8-1.

Task	Label (*P*-time)	Precedent tasks
Assemble pad	*AP*(7)	
Assemble flooring	*AF*(5)	
Assemble wall units	*AW*(6)	
Assemble dome frame	*AD*(8)	
Install floors	*IF*(5)	*AP, AF*
Install interior walls	*IW*(7)	*IF, AW*
Install dome frame	*ID*(5)	*AD, IW*
Install plumbing	*PL*(4)	*IF*
Install atomic power plant	*IP*(4)	*IW*
Install pressurization unit	*PU*(3)	*IP, ID*
Install heating units	*HU*(4)	*IP*
Install commode	*IC*(1)	*PL, HU*
Complete interior finish work	*FW*(6)	*IC*
Pressurize dome	*PD*(3)	*HU*
Install entertainment unit	*EU*(2)	*PU, HU*

TABLE 8-1 Tasks, *P*-times, and precedence relations for Example 8.2

73

Here are some of the questions we will address later in the chapter: How can you get your MHU built quickly? How many robots should you rent to do the job? How do you create a suitable work schedule that will get the job done? (A robot will do whatever it is told, but someone has to tell it what to do and when.)

8.2 Directed Graphs

A directed graph, or **digraph** for short, is a graph in which the edges have a direction associated with them, typically indicated by an arrowhead. Digraphs are particularly useful when we want to describe **asymmetric relationships** (X related to Y does not imply that Y must be related to X.

The classic example of an asymmetric relationship is romantic love: Just because X is in love with Y, there is no guarantee that Y reciprocates that love. Given two individuals X and Y and some asymmetric relationship (say love), we have four possible scenarios: Neither loves the other [Fig. 8-3(a)], X loves Y but Y does not love X [Fig. 8-3(b)], Y loves X but X does not love Y [Fig. 8-3(c)], and they love each other [Fig. 8-3(d)].

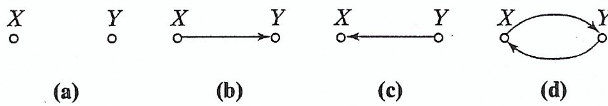


FIGURE 8-3

To distinguish digraphs from ordinary graphs, we use slightly different terminology.

- In a digraph, instead of talking about edges we talk about **arcs**. Every arc is defined by its *starting vertex* and its *ending vertex*, and we respect that order when we write the arc. Thus, if we write XY, we are describing the arc in Fig. 8-3(b) as opposed to the arc YX shown in Fig. 8-3(c).
- A list of all the arcs in a digraph is called the **arc-set** of the digraph. The digraph in Fig. 8-3(d) has arc-set $\mathcal{A} = \{XY, YX\}$.
- If XY is an arc in the digraph, we say that vertex X is **incident to** vertex Y, or, equivalently, that Y is **incident from** X.
- The arc YZ is said to be **adjacent** to the arc XY if the starting point of YZ is the ending point of XY. (Essentially, this means one can go from X to Z by way of Y.)
- In a digraph, a **path** from vertex X to vertex W ($W \neq X$) consists of a sequence of arcs $XY, YZ, ZU, \ldots, VW$ such that each arc is adjacent to the one before it and no arc appears more than once in the sequence—it is essentially a trip from X to W along the arcs in the digraph. The best way to describe the path is by listing the vertices in the order of travel: X, Y, Z, and so on.
- When the path starts and ends at the same vertex, we call it a **cycle** of the digraph. Just like circuits in a regular graph, cycles in digraphs can be written in more than one way—the cycle X, Y, Z, X is the same as the cycles Y, Z, X, Y and Z, X, Y, Z.
- In a digraph, the notion of the degree of a vertex is replaced by the concepts of *indegree* and *outdegree*. The **outdegree** of X is the number of arcs that have X as their *starting point* (outgoing arcs); the **indegree** of X is the number of arcs that have X as their *ending point* (incoming arcs).

The following example illustrates some of the above concepts.

EXAMPLE 8.3 DIGRAPH BASICS

The digraph in Fig. 8-4 has **vertex-set** $\mathcal{V} = \{A, B, C, D, E\}$ and arc-set $\mathcal{A} = \{AB, AC, BD, CA, CD, CE, EA, ED\}$. In this digraph, A is *incident to* B and C, but not to E. By the same token, A is *incident from* E as well as from C. The indegree of

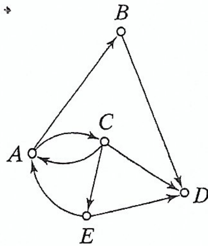


FIGURE 8-4

vertex A is 2, and so is the outdegree. The indegree of vertex D is 3, and the outdegree is 0. We leave it to the reader to find the indegrees and outdegrees of each of the other vertices of the graph.

In this digraph, there are several paths from A to D, such as A, C, D; A, C, E, D; A, B, D; and even A, C, A, B, D. On the other hand, A, E, D is not a path from A to D (because you can't travel directly from A to E). There are two cycles in this digraph: A, C, E, A (which can also be written as C, E, A, C and E, A, C, E) and A, C, A. Notice that there is no cycle passing through D because D has outdegree 0 and, thus, is a "dead-end," and there is no cycle passing through B because from B you can only go to D, and once there you are stuck.

While love is not to be minimized as a subject of study, there are many other equally important applications of digraphs:

- **The World Wide Web.** The Web is a giant digraph, where the vertices are Web pages and an arc from X to Y indicates that there is a *hyperlink* (informally called a *link*) on Web page X that allows you to jump directly to Web page Y. Web linkages are asymmetric—there may be a link on X that sends you to Y but no link in Y that sends you to X.
- **Traffic flow.** In most cities some streets are one-way streets and others are two-way streets. In this situation, digraphs allow us to visualize the flow of traffic through the city's streets. The vertices are intersections, and the *arcs* represent one-way streets. (To represent a two-way street, we use two arcs, one for each direction.)
- **Telephone traffic.** To track and analyze the traffic of telephone calls through their network, telephone companies use "call digraphs." In these digraphs the vertices are telephone numbers, and an arc from X to Y indicates that a call was initiated from telephone number X to telephone number Y.
- **Tournaments.** Digraphs are frequently used to describe certain types of tournaments, with the vertices representing the teams (or individual players) and the arcs representing the outcomes of the games played in the tournament (the arc XY indicates that X defeated Y). Tournament digraphs can be used in any sport in which the games cannot end in a tie (basketball, tennis, etc.).
- **Organization charts.** In any large organization (a corporation, the military, a university, etc.) it is important to have a well-defined chain of command. The best way to describe the chain of command is by means of a digraph often called an *organization chart*. In this digraph the *vertices* are the individuals in the organization, and an *arc* from X to Y indicates that X is Y's immediate boss (i.e., Y takes orders directly from X).

As you probably guessed by now, digraphs are also used in scheduling. There is no better way to visualize the tasks, processing times, and precedence relations in a project than by means of a digraph in which the vertices represent the tasks (with their processing times indicated in parentheses) and the arcs represent the precedence relations.

EXAMPLE 8.4 PROJECT DIGRAPH FOR THE MARTIAN HOME

Let's return to the scheduling problem first discussed in Example 8.2. We can take the tasks and precedence relations given in Table 8-1 and create a digraph like the one shown in Fig. 8-5(a). It is helpful to try to place the vertices of the

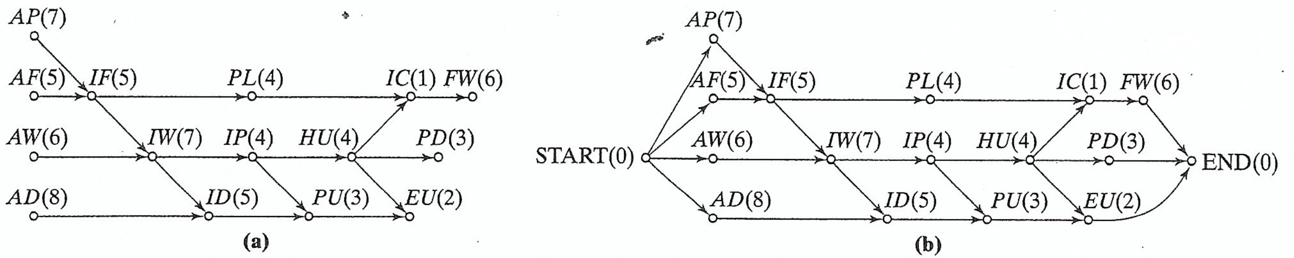


FIGURE 8-5

digraph so that the arcs point from left to right, and this can usually be done with a little trial-and-error. After this is done, it is customary to add two fictitious tasks called START and END, where START indicates the imaginary task of getting the project started (cutting the red ribbon, so to speak) and END indicates the imaginary task of declaring the project complete [Fig. 8-5(b)]. By giving these fictitious tasks zero processing time, we avoid affecting the time calculations for the project.

A digraph having the tasks (and completion times) as vertices, the precedence relations as arcs, and with the fictitious tasks START (0) and END (0) representing the beginning and end of the project, respectively, is called a **project digraph**. The project digraph allows us to better visualize the execution of the project as a flow, moving from left to right.

8.3 Priority-List Scheduling

The project digraph is the basic graph model used to package all the information in a scheduling problem, but there is nothing in the project digraph itself that specifically tells us how to create a schedule. We are going to need something else, some set of instructions that indicates the order in which tasks should be executed. We can accomplish this by the simple act of prioritizing the tasks in some specified order, called a *priority list.*

A **priority list** is a list of all the tasks prioritized in the order we prefer to execute them. If task X is ahead of task Y in the priority list, then X gets priority over Y. This means that when it comes to a choice between the two, *X is executed ahead of Y*. However, if X is not yet *ready* for execution, then *we skip over it and move on to the first ready task after X in the priority list.* If there are no ready tasks after X in the priority list, the free processors must sit idle and wait until a task becomes ready.

The process of scheduling tasks using a priority list and following these basic rules is known as the *priority-list model* for scheduling. The priority-list model is a completely general model for scheduling—every priority list produces a schedule, and any schedule can be created from some (usually more than one) "parent" priority list. The trick is going to be to figure out *which* priority lists give us good schedules and which don't. We will come back to this topic in Sections 8.4 and 8.5.

Since each time we change the order of the tasks we get a different priority list, there are as many priority lists as there are ways to order the tasks. For three tasks, there are six possible priority lists; for 4 tasks, there are 24 priority lists; for 10 tasks, there are more than 3 million priority lists; and for 100 tasks, there are more priority lists than there are molecules in the universe.

76

For a review of permutations and factorials, see Section 2.4.

Clearly, a shortage of priority lists is not going to be our problem. If this sounds familiar, it's because we have seen the idea before—priority lists are nothing more than *permutations of the tasks*. Like sequential coalitions (Chapter 2) and Hamilton circuits (Chapter 6), the number of priority lists is given by a factorial.

NUMBER OF PRIORITY LISTS

The number of possible priority lists in a project consisting of M tasks is $M! = M \times (M-1) \times \cdots \times 2 \times 1$

Before we proceed, we will illustrate how the priority-list model for scheduling works with a few small but important examples. Even with such small examples, there is a lot to keep track of, and you are well advised to have pencil and paper in front of you as you follow the details.

EXAMPLE 8.5 PREPARING FOR LAUNCH: PART 1

Immediately preceding the launch of a satellite into space, last-minute system checks need to be performed by the on-board computers, and it is important to complete these system checks as quickly as possible—for both cost and safety reasons. Suppose that there are five system checks required: $A(6), B(5), C(7), D(2)$, and $E(5)$, with the numbers in parentheses representing the hours it takes one computer to perform that system check. In addition, there are precedence relations: D cannot be started until both A and B have been finished, and E cannot be started until C has been finished. The project digraph is shown in Fig. 8-6.

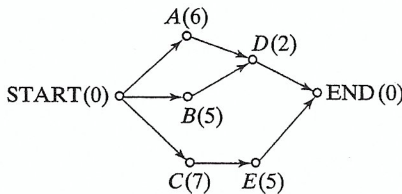


FIGURE 8-6

Let's assume that there are two identical computers on board (P_1 and P_2) that will carry out the individual system checks. How do we use the priority-list model to create a schedule for these two processors?

For starters, we will need a priority list. Suppose that the priority list is given by listing the system checks in alphabetical order, and let's follow the evolution of the project under the priority-list model. We will use T to indicate the elapsed time in hours.

Priority list: $A(6), B(5), C(7), D(2), E(5)$

- **$T = 0$ (START).** $A(6)$, $B(5)$, and $C(7)$ are the only *ready* tasks. Following the priority list, we assign $A(6)$ to P_1 and $B(5)$ to P_2.
- **$T = 5$.** P_1 is still *busy* with $A(6)$; P_2 has just *completed* $B(5)$. $C(7)$ is the only available *ready* task. We assign $C(7)$ to P_2.
- **$T = 6$.** P_1 has just *completed* $A(6)$; P_2 is *busy* with $C(7)$. $D(2)$ has just become a *ready* task (A and B have been completed). We assign $D(2)$ to P_1.
- **$T = 8$.** P_1 has just *completed* $D(2)$; P_2 is still *busy* with $C(7)$. There are no *ready* tasks at this time for P_1, so P_1 has to sit *idle*.

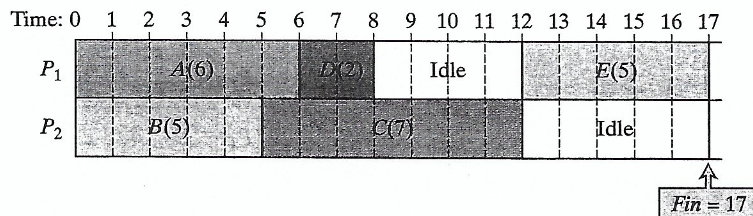


FIGURE 8-7

- **$T = 12$.** P_1 is *idle*; P_2 has just *completed* $C(7)$. Both processors are *ready* for work. $E(5)$ is the only ready task, so we assign $E(5)$ to P_1, P_2 sits *idle*. (Note that in this situation, we could have just as well assigned $E(5)$ to P_2 and let P_1 sit idle. The processors don't get tired and don't care if they are working or idle, so the choice is random.)
- **$T = 17$ (END).** P_1 has just *completed* $E(5)$, and, therefore, the project is completed.

The evolution of the entire project together with the project finishing time ($Fin = 17$) are captured in the timeline shown in Fig. 8-7. Is this a good schedule? Given the excessive amount of idle time (a total of 9 hours), one might suspect this is a rather bad schedule. How could we improve it? We might try changing the priority list.

EXAMPLE 8.6 **PREPARING FOR LAUNCH: PART 2**

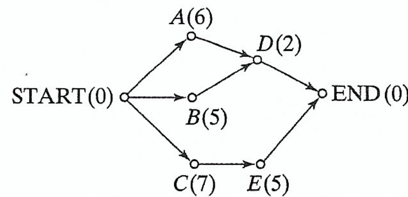


FIGURE 8-8

We are going to schedule the satellite launch system checks with the same two processors but with a different priority list. The project digraph is shown once again in Fig. 8-8. (When scheduling, it's really useful to have the project digraph right in front of you.)

This time let's try a reverse alphabetical order for the priority list. Why? Why not—at this point we are just shooting in the dark! (Don't worry—we will become a lot more enlightened later in the chapter.)

Priority list: $E(5), D(2), C(7), B(5), A(6)$

- **$T = 0$ (START).** $C(7)$, $B(5)$, and $A(6)$ are the only *ready* tasks. Following the priority list, we assign $C(7)$ to P_1 and $B(5)$ to P_2.
- **$T = 5$.** P_1 is still *busy* with $C(7)$; P_2 has just *completed* $B(5)$. $A(6)$ is the only available ready task. We assign $A(6)$ to P_2.
- **$T = 7$.** P_1 has just *completed* $C(7)$; P_2 is *busy* with $A(6)$. $E(5)$ has just become a *ready* task, and we assign it to P_1.
- **$T = 11$.** P_2 has just *completed* $A(6)$; P_1 is *busy* with $E(5)$. $D(2)$ has just become a *ready* task, and we assign it to P_2.
- **$T = 12$.** P_1 has just *completed* $E(5)$; P_2 is *busy* with $D(2)$. There are no tasks left, so P_1 sits idle.
- **$T = 13$. (END).** P_2 has just *completed* the last task, $D(2)$. Project is completed.

The timeline for this schedule is shown in Fig. 8-9. The project finishing time is $Fin = 13$ hours.

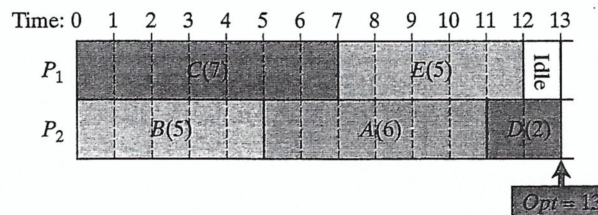


FIGURE 8-9

Clearly, this schedule is a lot better than the one obtained in Example 8.5. In fact, we were pretty lucky—this schedule turns out to be an optimal schedule for two processors. [Two processors cannot finish this project in less than 13 hours because there is a total of 25 hours worth of work (the sum of all processing times), which implies that in the best of cases it would take 12.5 hours to finish the project. But since the processing times are all whole numbers and tasks cannot be split, the finishing time cannot be less than 13 hours! Thus, $Opt = 13$ hours.]

Thirteen hours is still a long time for the computers to go over their system checks. Since that's the best we can do with two computers, the only way to speed things up is to add a third computer to the "workforce." Adding another computer to the satellite can be quite expensive, but perhaps it will speed things up enough to make it worth it. Let's see.

EXAMPLE 8.7 PREPARING FOR LAUNCH: PART 3

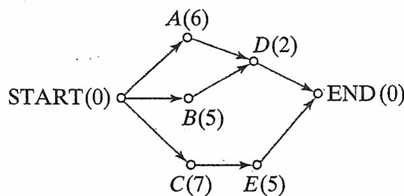


FIGURE 8-10

We will now schedule the system checks using $N = 3$ computers (P_1, P_2, P_3). For the reader's convenience the project digraph is shown again in Fig. 8-10. We will use the "good" priority list we found in Example 8.6.

Priority list: $E(5), D(2), C(7), B(5), A(6)$

- **$T = 0$ (START).** $C(7)$, $B(5)$, and $A(6)$ are the *ready* tasks. We assign $C(7)$ to P_1, $B(5)$ to P_2, and $A(6)$ to P_3.
- **$T = 5$.** P_1 is *busy* with $C(7)$; P_2 has just *completed* $B(5)$; and P_3 is *busy* with $A(6)$. There are no available ready tasks for P_2 [$E(5)$ can't be started until $C(7)$ is done, and $D(2)$ can't be started until $A(6)$ is done], so P_2 sits idle until further notice.
- **$T = 6$.** P_3 has just *completed* $A(6)$; P_2 is *idle*; and P_1 is still *busy* with $C(7)$. $D(2)$ has just become a *ready* task. We randomly assign $D(2)$ to P_2 and let P_3 be idle, since there are no other ready tasks. [Note that we could have just as well assigned $D(2)$ to P_3 and let P_2 be idle.]
- **$T = 7$.** P_1 has just *completed* $C(7)$ and $E(5)$ has just become a *ready* task, so we assign it to P_1. There are no other tasks to assign, so P_3 continues to sit idle.
- **$T = 8$.** P_2 has just *completed* $D(2)$. There are no other tasks to assign, so P_2 and P_3 both sit *idle*.
- **$T = 12$ (END).** P_1 has just *completed* the last task, $E(5)$, so the project is completed.

The timeline for this schedule is shown in Fig. 8-11. The project finishing time is $Fin = 12$ hours, a pathetically small improvement over the two-processor schedule found in Example 8.6. The cost of adding a third processor doesn't seem to justify the benefit.

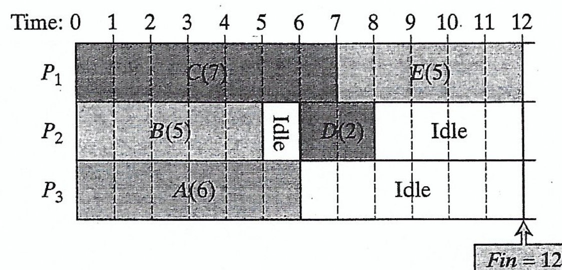


FIGURE 8-11

The Priority-List Model

The previous three examples give us a general sense of how to create a schedule from a project digraph *and* a priority list. We will now formalize the ground rules of the **priority-list model** for scheduling.

At any particular moment in time throughout a project, a processor can be either *busy* or *idle* and a task can be *ineligible, ready, in execution*, or *completed.* Depending on the various combinations of these, there are three different scenarios to consider:

- *All processors are busy.* In this case, there is nothing we can do but wait.
- *One processor is free.* In this case, we scan the priority list from left to right, looking for the first *ready* task in the priority list and assign it to the free processor. (Remember that for a task to be *ready*, all the tasks that are precedent to it must have been completed.) If there are no ready tasks at that moment, the processor stays idle until things change.
- *More than one processor is free.* In this case, the *first ready* task on the priority list is given to one free processor, the second ready task is given to another free processor, and so on. If there are more free processors than ready tasks, some of the processors will remain idle until one or more tasks become ready. Since the processors are identical and tireless, the choice of which free processor is assigned which ready task is completely arbitrary.

It's fair to say that the basic idea behind the priority-list model is not difficult, but there is a lot of bookkeeping involved, and that becomes critical when the number of tasks is large. At each stage of the schedule we need to keep track of the status of each task—which tasks are *ready* for processing, which tasks are *in execution*, which tasks have been *completed*, which tasks are still *ineligible*. One convenient recordkeeping strategy goes like this: On the priority list itself *ready* tasks are circled in red [Fig. 8-12(a)]. When a ready task is picked up by a processor and goes into *execution*, put a single red slash through the red circle [Fig. 8-12(b)]. When a task that has been in execution is completed, put a second red slash through the circle [Fig. 8-12(c)]. At this point, it is also important to check the project digraph to see if any new tasks have all of a sudden become eligible. Tasks that are *ineligible* remain unmarked [Fig. 8-12(d)].

FIGURE 8-12 "Road" signs on a priority list. (a) Task X is *ready*. (b) Task X is *in execution*. (c) Task X is *completed*. (d) Task X is *ineligible*.

As they say, the devil is in the details, so a slightly more substantive example will help us put everything together—the project digraph, the priority list model, and the bookkeeping strategy.

EXAMPLE 8.8 ASSEMBLING A MARTIAN HOME WITH A PRIORITY LIST

This is the third act of the Martian Habitat Unit (MHU) building project. We are finally ready to start the project of assembling that MHU, and, like any good scheduler, we will first work the entire schedule out with pencil and paper. Let's start with the assumption that maybe we can get by with just two robots (P_1 and P_2). For the reader's convenience, the project digraph is shown again in Fig. 8-13.

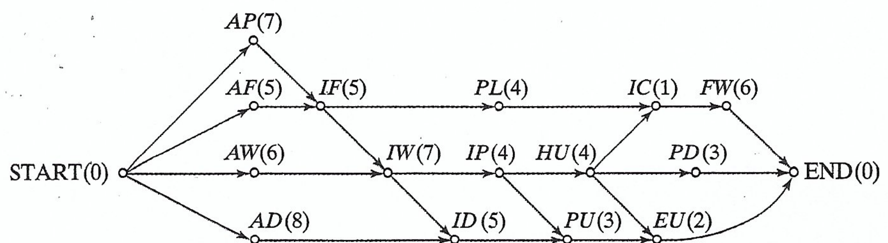


FIGURE 8-13

80

Let's start with a random priority list.

Priority list: $AD(8)$, $AW(6)$, $AF(5)$, $IF(5)$, $AP(7)$, $IW(7)$, $ID(5)$, $IP(4)$, $PL(4)$, $PU(3)$, $HU(4)$, $IC(1)$, $PD(3)$, $EU(2)$, $FW(6)$. (Ready tasks are circled in red.)

- **$T = 0$ (START).** Status of processors: P_1 starts AD; P_2 starts AW. We put a single red slash through AD and AW.

  Priority list: AD, AW, AF, IF, AP, IW, ID, IP, PL, PU, HU, IC, PD, EU, FW.
- **$T = 6$.** P_1 busy (executing AD); P_2 completed AW (put a second slash through AW) and starts AF (put a slash through AF).

  Priority list: AD, AW, AF, IF, AP, IW, ID, IP, PL, PU, HU, IC, PD, EU, FW.
- **$T = 8$.** P_1 completed AD and starts AP; P_2 is busy (executing AF).

  Priority list: AD, AW, AF, IF, AP, IW, ID, IP, PL, PU, HU, IC, PD, EU, FW.
- **$T = 11$.** P_1 busy (executing AP); P_2 completed AF, but since there are no ready tasks, it remains idle.

  Priority list: AD, AW, AF, IF, AP, IW, ID, IP, PL, PU, HU, IC, PD, EU, FW.
- **$T = 15$.** P_1 completed AP. IF becomes a ready task and goes to P_1; P_2 stays idle.

  Priority list: AD, AW, AF, IF, AP, IW, ID, IP, PL, PU, HU, IC, PD, EU, FW.

At this point, we will let you take over and finish the schedule. (Remember—the object is to learn how to keep track of the status of each task, and the only way to do this is with practice.)

After a fair amount of work, we obtain the final schedule shown in Fig. 8-14, with project finishing time $Fin = 44$ hours.

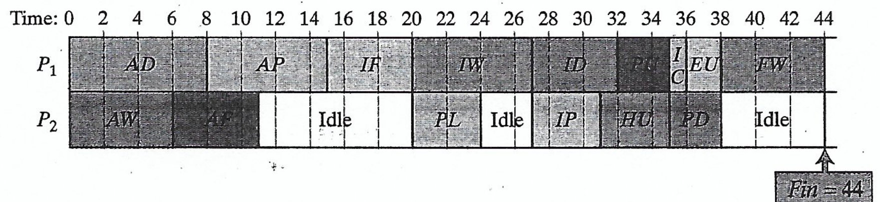


FIGURE 8-14

Scheduling with priority lists is a two-part process: (1) choose a priority list and (2) use the priority list and the ground rules of the priority-list model to come up with a schedule (Fig. 8-15). As we saw in the previous example, the second part is long and tedious, but purely mechanical—it can be done by anyone (or anything) that is able to follow a set of instructions, be it a meticulous student or a properly programmed computer. We will use the term *scheduler* to describe the entity (be it student or machine) that takes a priority list as input and produces the schedule as output.

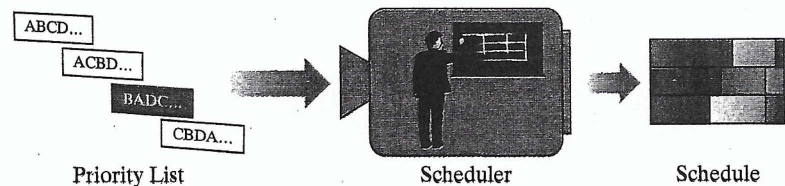


FIGURE 8-15

Ironically, it is the seemingly easiest part of this process—choosing a priority list—that is actually the most interesting. Among all the priority lists there is one (or more) that will give a schedule with the optimal finishing time. We will call these **optimal priority lists**. How do we find an optimal priority list? Short of that, how do we find "good" priority lists, that is, priority lists that give schedules with finishing times reasonably close to the optimal? These are both important questions, and some answers are coming up next.

81

8.4 The Decreasing-Time Algorithm

Our first attempt to find a good priority list is to formalize what is an intuitive and commonly used strategy: *Do the longer jobs first and leave the shorter jobs for last.* In terms of priority lists, this strategy is implemented by creating a priority list in which the tasks are listed in decreasing order of processing times—longest first, second longest next, and so on. (When there are two or more tasks with equal processing times, we order them randomly.)

A priority list in which the tasks are listed in decreasing order of processing times is called, not surprisingly, a **decreasing-time priority list**, and the process of creating a schedule using a decreasing-time priority list is called the **decreasing-time algorithm**.

EXAMPLE 8.9 THE DECREASING TIME ALGORITHM GOES TO MARS

Figure 8-16 shows, once again, the project digraph for the Martian Habitat Unit building project. To use the decreasing-time algorithm we first prioritize the 15 tasks in a decreasing-time priority list.

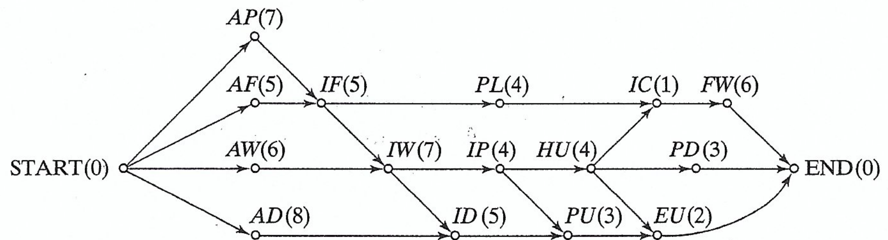


FIGURE 8-16

Decreasing-time priority list: $AD(8)$, $AP(7)$, $IW(7)$, $AW(6)$, $FW(6)$, $AF(5)$, $IF(5)$, $ID(5)$, $IP(4)$, $PL(4)$, $HU(4)$, $PU(3)$, $PD(3)$, $EU(2)$, $IC(1)$

Using the decreasing-time algorithm with $N = 2$ processors, we get the schedule shown in Fig. 8-17, with project finishing time $Fin = 42$ hours. (Table 8-2 shows the details of how this schedule came about, but you may want to try re-creating the schedule on your own.)

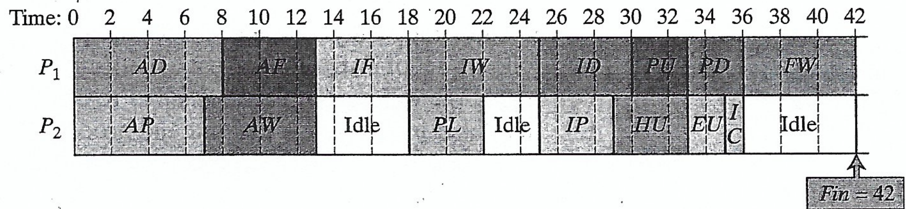


FIGURE 8-17

When looking at the finishing time under the decreasing-time algorithm, one can't help but feel disappointed. The sensible idea of prioritizing the longer jobs ahead of the shorter jobs turned out to be somewhat of a dud—at least in this example! What went wrong? If we work our way backward from the end, we can see that we made a bad choice at $T = 33$ hours. At this point there were three ready tasks [$PD(3)$, $EU(2)$, and $IC(1)$], and both processors were available. Based on the decreasing-time priority list, we chose the two "longer" tasks, $PD(3)$ and $EU(2)$, ahead of the short task, $IC(1)$. This was a bad move! $IC(1)$ is a much more *critical* task than the other two because we can't start task $FW(6)$ until we finish task $IC(1)$. Had we looked ahead at some of the tasks for which *PD*, *EU*, and *IC* are precedent tasks, we might have noticed this.

82

Step	Time	Priority-List Status	Schedule Status
1	$T = 0$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD P_2: AP
2	$T = 7$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD P_2: AP, AW
3	$T = 8$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF P_2: AP, AW
4	$T = 13$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF P_2: AP, AW
5	$T = 18$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW P_2: AP, AW, Idle, PL
6	$T = 22$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW P_2: AP, AW, Idle, PL
7	$T = 25$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW, ID P_2: AP, AW, Idle, PL, Idle, IP
8	$T = 29$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW, ID P_2: AP, AW, Idle, PL, Idle, IP, HU
9	$T = 30$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW, ID, PU P_2: AP, AW, Idle, PL, Idle, IP, HU
10	$T = 33$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW, ID, PU, PD P_2: AP, AW, Idle, PL, Idle, IP, HU, EU
11	$T = 35$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW, ID, PU, PD P_2: AP, AW, Idle, PL, Idle, IP, HU, EU, IC
12	$T = 36$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW, ID, PU, PD, FW P_2: AP, AW, Idle, PL, Idle, IP, HU, EU, IC
13	$T = 42$	AD(8) AP(7) IW(7) AW(6) FW(6) AF(5) IF(5) ID(5) IP(4) PL(4) HU(4) PU(3) PD(3) EU(2) IC(1)	Time: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 P_1: AD, AF, IF, IW, ID, PU, PD, FW P_2: AP, AW, Idle, PL, Idle, IP, HU, EU, IC, Idle

TABLE 8-2 Scheduling the assembly of the Martian Habitat Unit with the decreasing-time algorithm

83

An even more blatant example of the weakness of the decreasing-time algorithm occurs at the very start of this schedule, when the algorithm fails to take into account that task $AF(5)$ should have a very high priority. Why? $AF(5)$ is one of the two tasks that must be finished before $IF(5)$ can be started, and $IF(5)$ must be finished before $IW(7)$ can be started, which must be finished before $IP(4)$ and $ID(5)$ can be started, and so on down the line.

8.5 Critical Paths and the Critical-Path Algorithm

When there is a long path of tasks in the project digraph, it seems clear that the first task along that path should be started as early as possible. This idea leads to the following informal rule: *The greater the total amount of work that lies ahead of a task, the sooner that task should be started.*

To formalize these ideas, we will introduce the concepts of *critical paths* and *critical times*.

- **Critical path (time) for a vertex.** For a given vertex X of a project digraph, the *critical path for X* is the path *from X to END* with *longest* processing time. (The *processing time* of a path is defined to be the sum of the processing times of all the tasks along the path.) When we add the processing times of all the tasks along the critical path for a vertex X, we get the *critical time for X*. (By definition, the critical time of END is 0.)
- **Critical path (time) for a project.** The path with longest processing time from START to END is called the *critical path for the project*, and the total processing time for this critical path is called the *critical time for the project*.

EXAMPLE 8.10 CRITICAL PATHS IN THE MARTIAN DIGRAPH

Figure 8-18 shows the project digraph for assembling the Martian Habitat Unit. We will find critical paths and critical times for several vertices of the project digraph.

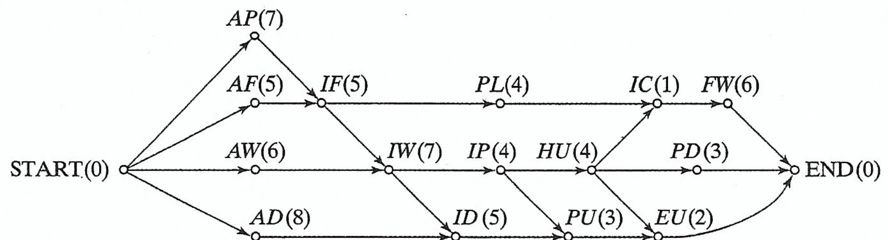


FIGURE 8-18

- Let's start with a relatively easy case—the vertex HU. A quick look at Fig. 8-18 should convince you that there are only three paths from HU to END, namely,

  1. HU, IC, FW, END, with processing time $4 + 1 + 6 = 11$ hours,
  2. HU, PD, END, with processing time $4 + 3 = 7$ hours, and
  3. HU, EU, END, with processing time $4 + 2 = 6$ hours.

  Of the three paths, (1) has the longest processing time, so HU, IC, FW, END is the *critical path* for vertex HU. The *critical time* for HU is 11 hours.

84

- Next, let's find the critical path for vertex AD. There is only one path from AD to END, namely AD, ID, PU, EU, END, which makes the decision especially easy. Since this is the only path, it is automatically the longest path and, therefore, the *critical path* for AD. The *critical time* for AD is $8 + 5 + 3 + 2 = 18$ hours.
- To find the critical path for the project, we need to find the path from START to END with longest processing time. Since there are dozens of paths from START to END, let's just eyeball the project digraph for a few seconds and take our best guess. . . .
- OK, if you guessed START, AP, IF, IW, IP, HU, IC, FW, END, you have good eyes. This is indeed the *critical path*. It follows that the *critical time* for the Martian Habitat Unit project is 34 hours.

We will soon discuss the special role that the critical time and the critical path play in scheduling, but before we do so, let's address the issue of how to find critical paths. In a large project digraph there may be thousands of paths from START to END, and the "eyeballing" approach we used in Example 8.10 is not likely to work. What we need here is an efficient algorithm, and fortunately there is one—it is called the **backflow algorithm**.

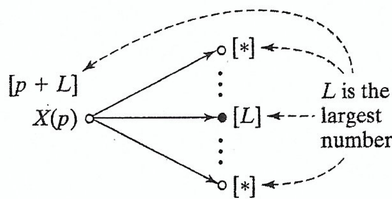


FIGURE 8-19

THE BACKFLOW ALGORITHM

- **Step 1.** Find the critical time for every vertex of the project digraph. This is done by starting at END and working backward toward START according to the following rule: *The critical time for a task X equals the processing time of X plus the largest critical time among the vertices incident from X.* The general idea is illustrated in Fig. 8-19. [To help with the recordkeeping, it is suggested that you write the critical time of the vertex in square brackets [ ] to distinguish it from the processing time in parentheses ( ).]
- **Step 2.** Once we have the critical time for every vertex in the project digraph, critical paths are found by just following the *path along largest critical times*. In other words, the critical path for any vertex X (and that includes START) is obtained by starting at X and moving to the adjacent vertex with largest critical time, and from there to the adjacent vertex with largest critical time, and so on.

While the backflow algorithm sounds a little complicated when described in words, it is actually pretty easy to implement in practice, as we will show in the next example.

EXAMPLE 8.11 THE BACKFLOW ALGORITHM AND THE MARTIAN DIGRAPH

We are now going to use the backflow algorithm to find the critical time for each of the vertices of the Martian Habitat Unit project digraph (Fig. 8-20).

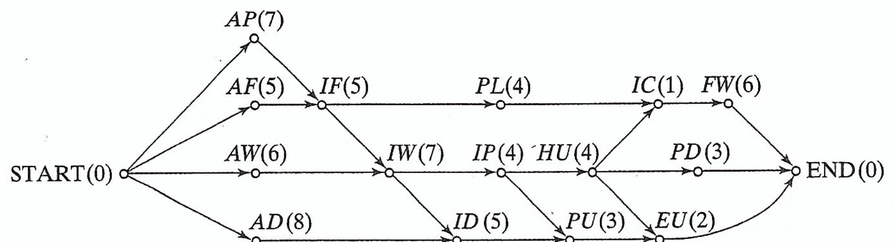


FIGURE 8-20

- **Step 1.**
  - Start at END. The critical time of END is 0, so we add a $[0]$ next to END(0).
  - The backflow now moves to the three vertices that are incident to END, namely, $FW(6)$, $PD(3)$, and $EU(2)$. In each case the critical time is the processing time plus 0, so the critical times are $FW[6]$, $PD[3]$, and $EU[2]$. We add this information to the project digraph.
  - From $FW[6]$, the backflow moves to $IC(1)$. The vertex $IC(1)$ is incident only to $FW[6]$, so the critical time for IC is $1+6=7$. We add a $[7]$ next to IC in the project digraph.
  - The backflow now moves to $HU(4)$, $PL(4)$, and $PU(3)$. There are three vertices $HU(4)$ is incident to ($IC[7]$, $PD[3]$, and $EU[2]$). Of the three, the one with the largest critical time is $IC[7]$. This means that the critical time for HU is $4+7=11$. $PL(4)$ is only incident to $IC[7]$, so its critical time is $4+7=11$. $PU(3)$ is only incident to $EU[2]$, so its critical time is $3+2=5$. Add $[11]$, $[11]$, and $[5]$ next to HU, PL, and PU, respectively.
  - The backflow now moves to $IP(4)$ and $ID(5)$. $IP(4)$ is incident to $HU[11]$ and $PU[5]$, so the critical time for IP is $4+11=15$. $ID(5)$ is only incident to $PU(5)$, so its critical time is $5+5=10$. We add $[15]$ next to IP, and $[10]$ next to ID.
  - The backflow now moves to $IW(7)$. The critical time for IW is $7+15=22$. (Please verify that this is correct!)
  - The backflow now moves to $IF(5)$. The critical time for IF is $5+22=27$. (Ditto.)
  - The backflow now moves to $AP(7)$, $AF(5)$, $AW(6)$, and $AD(8)$. Their respective critical times are $7+27=34$, $5+27=32$, $6+22=28$, and $8+10=18$.
  - Finally, the backflow reaches START(0). We still follow the same rule—the critical time is $0+34=34$. This is the critical time for the project!
- **Step 2.** The critical time for every vertex of the project digraph is shown in red numbers in Fig. 8-21. We can now find the critical path by following the trail of largest critical times: START, *AP*, *IF*, *IW*, *IP*, *HU*, *IC*, *FW*, END.

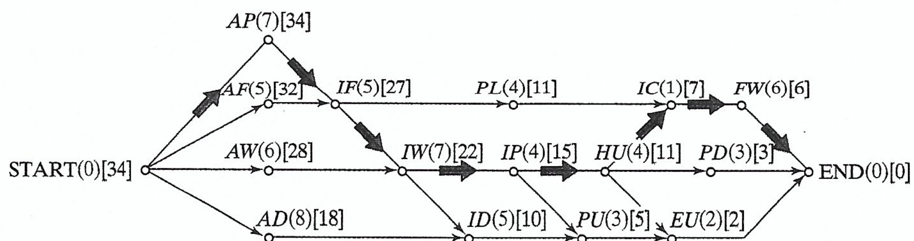


FIGURE 8-21

Why are the critical path and critical time of a project of special significance? We saw earlier in the chapter that for every project there is a theoretical time barrier below which a project cannot be completed, regardless of how clever the scheduler is or how many processors are used. Well, guess what? This theoretical barrier *is the project's critical time.*

If a project is to be completed in the optimal completion time, it is absolutely essential that all the tasks in the critical path be done at the earliest possible time. Any delay in starting up one of the tasks in the critical path will necessarily delay the finishing time of the entire project. (By the way, this is why this path is called *critical.*)

Unfortunately, it is not always possible to schedule the tasks on the critical path one after the other, bang, bang, bang without delay. For one thing, processors

are not always free when we need them. (Remember that a processor cannot stop in the middle of one task to start a new task.) Another reason is the problem of uncompleted precedent tasks. We cannot concern ourselves only with tasks along the critical path and disregard other tasks that might affect them through precedence relations. There is a whole web of interrelationships that we need to worry about. Optimal scheduling is extremely complex.

The Critical-Path Algorithm

The concept of critical paths can be used to create very good (although not necessarily optimal) schedules. The idea is to use *critical times* rather than processing times to prioritize the tasks. The priority list we obtain when we write the tasks in decreasing order of *critical times* (with ties broken randomly) is called the **critical-time priority list**, and the process of creating a schedule using the critical-time priority list is called the **critical-path algorithm**.

CRITICAL-PATH ALGORITHM

- **Step 1: Find critical times.** Using the backflow algorithm, find the *critical time* for every task in the project.
- **Step 2: Create priority list.** Using the critical times obtained in Step 1, create a *priority list* with the tasks listed in decreasing order of critical times (i.e., a critical-time priority list).
- **Step 3: Create schedule.** Using the critical-time priority list obtained in Step 2, create the *schedule*.

There are, of course, plenty of small details that need to be attended to when carrying out the critical-path algorithm, especially in Steps 1 and 3. Fortunately, everything that needs to be done we now know how to do.

EXAMPLE 8.12 THE CRITICAL-PATH ALGORITHM GOES TO MARS

We will now describe the process for scheduling the assembly of the Martian Habitat Unit using the critical-path algorithm and $N = 2$ processors.

We took care of Step 1 in Example 8.11. The critical times for each task are shown in red in Fig. 8-22.

Step 2 follows directly from Step 1. The critical-time priority list for the project is $AP[34]$, $AF[32]$, $AW[28]$, $IF[27]$, $IW[22]$, $AD[18]$, $IP[15]$, $PL[11]$, $HU[11]$, $ID[10]$, $IC[7]$, $FW[6]$, $PU[5]$, $PD[3]$, $EU[2]$.

Step 3 is a lot of busywork—the details are left to the reader.

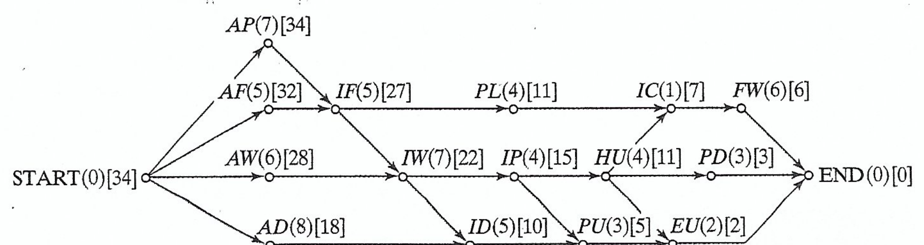


FIGURE 8-22

87

The timeline for the resulting schedule is given in Fig. 8-23. The project finishing time is *Fin* = 36 hours. This is a very good schedule, but it is not an optimal schedule. (Figure 8-24 shows the timeline for an *optimal schedule* with finishing time *Opt* = 35 hours.)

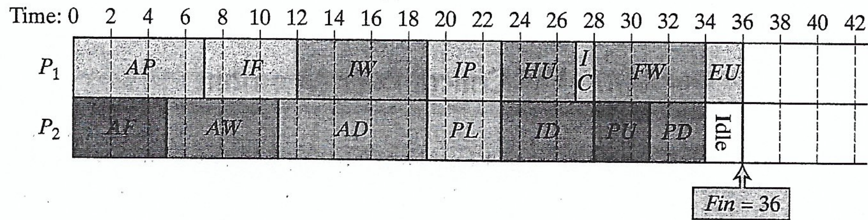


FIGURE 8-23 Timeline for the MHU building project under the critical-path algorithm ($N = 2$).

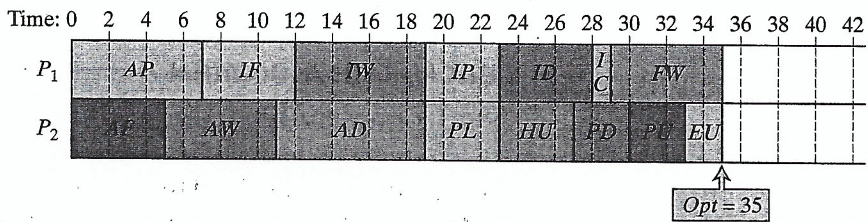


FIGURE 8-24 Timeline for an *optimal schedule* for the MHU building project ($N = 2$).

The critical-path algorithm is an excellent *approximate* algorithm for scheduling a project, but as Example 8.12 shows, it does not always give an optimal schedule. In this regard, scheduling problems are like TSPs (Chapter 6)—there are *efficient approximate* algorithms for scheduling, but no *efficient optimal* algorithm is currently known. Of the standard scheduling algorithms, the critical-path algorithm is by far the most commonly used. Other, more sophisticated algorithms have been developed in the last 40 years, and under specialized circumstances they can outperform the critical-path algorithm, but as an all-purpose algorithm for scheduling, the critical-path algorithm is hard to beat.

Conclusion

In one form or another, the scheduling of human (and nonhuman) activity is a pervasive and fundamental problem of modern life. At its most informal, it is part and parcel of the way we organize our everyday living (so much so that we are often scheduling things without realizing we are doing so). In its more formal incarnation, the systematic scheduling of a set of activities for the purposes of saving either time or money is a critical issue in management science. Business, industry, government, education—wherever there is a big project, there is a schedule behind it.

By now, it should not surprise us that at their very core, scheduling problems are mathematical in nature and that the mathematics of scheduling can range from the simple to the extremely complex. In this chapter we focused on a very specific type of scheduling problem in which we are given a set of *tasks*, a set of *precedence relations* among the tasks, and a set of identical *processors*. The objective is to schedule the tasks by properly assigning tasks to processors so that the *project finishing time* is as small as possible.

88

2. For the digraph shown in Fig. 8-26, give

(a) the arc-set.

(b) the sum of the indegrees of all the vertices.

(c) the sum of the outdegrees of all the vertices.

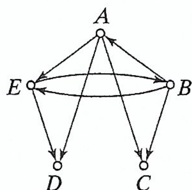


FIGURE 8-26

3. For the digraph in Fig. 8-25 (page 250), find

(a) all the possible paths from A to D.

(b) all the cycles in the digraph.

4. For the digraph in Fig. 8-26, find

(a) all the possible paths from B to E.

(b) all the cycles in the digraph.

5. For the digraph in Fig. 8-25 (page 250), find

(a) all vertices that are incident *to* A.

(b) all vertices that are incident *from* A.

(c) all vertices that are incident *to* D.

(d) all vertices that are incident *from* D.

(e) all the arcs adjacent to AC.

(f) all the arcs adjacent to CD.

6. For the digraph in Fig. 8-26, find

(a) all vertices that are incident *to* A.

(b) all vertices that are incident *from* A.

(c) all vertices that are incident *to* D.

(d) all vertices that are incident *from* D.

(e) all the arcs adjacent to BA.

(f) all the arcs adjacent to BC.

7. (a) Draw a digraph with vertex-set $\mathcal{V} = \{A, B, C, D\}$ and arc-set $\mathcal{A} = \{AB, AC, AD, BD, DB\}$.

(b) Draw a digraph with vertex-set $\mathcal{V} = \{A, B, C, D, E\}$ and arc-set $\mathcal{A} = \{AC, AE, BD, BE, CD, DC, ED\}$.

(c) Draw a digraph with vertex-set $\mathcal{V} = \{W, X, Y, Z\}$ and such that W is incident to X and Y, X is incident to Y and Z, Y is incident to Z and W, and Z is incident to W and X.

8. (a) Draw a digraph with vertex-set $\mathcal{V} = \{A, B, C, D\}$ and arc-set $\mathcal{A} = \{AB, AC, AD, BC, BD, DB, DC\}$.

(b) Draw a digraph with vertex-set $\mathcal{V} = \{V, W, X, Y, Z\}$ and arc-set $\mathcal{A} = \{VW, VZ, WZ, XV, XY, XZ, YW, ZY, ZW\}$.

(c) Draw a digraph with vertex-set $\mathcal{V} = \{W, X, Y, Z\}$ and such that every vertex is incident to every other vertex.

9. Consider the digraph with vertex-set $\mathcal{V} = \{A, B, C, D, E\}$ and arc-set $\mathcal{A} = \{AB, AE, CB, CE, DB, EA, EB, EC\}$. Without drawing the digraph, determine

(a) the outdegree of A.

(b) the indegree of A.

(c) the outdegree of D.

(d) the indegree of D.

10. Consider the digraph with vertex-set $\mathcal{V} = \{V, W, X, Y, Z\}$ and arc-set $\mathcal{A} = \{VW, VZ, WZ, XY, XZ, YW, ZY, ZW\}$. Without drawing the digraph, determine

(a) the outdegree of V.

(b) the indegree of V.

(c) the outdegree of Z.

(d) the indegree of Z.

11. Consider the digraph with vertex-set $\mathcal{V} = \{A, B, C, D, E, F\}$ and arc-set $\mathcal{A} = \{AB, BD, CF, DE, EB, EC, EF\}$.

(a) Find a path from vertex A to vertex F.

(b) Find a Hamilton path from vertex A to vertex F. (*Note*: A Hamilton path is a path that passes through every vertex of the graph once.)

(c) Find a cycle in the digraph.

(d) Explain why vertex F cannot be part of any cycle.

(e) Explain why vertex A cannot be part of any cycle.

(f) Find all the cycles in this digraph.

12. Consider the digraph with vertex-set $\mathcal{V} = \{A, B, C, D, E\}$ and arc-set $\mathcal{A} = \{AB, AE, CB, CD, DB, DE, EB, EC\}$.

(a) Find a path from vertex A to vertex D.

(b) Explain why the path you found in (a) is the only possible path from vertex A to vertex D.

(c) Find a cycle in the digraph.

(d) Explain why vertex A cannot be part of a cycle.

(e) Explain why vertex B cannot be part of a cycle.

(f) Find all the cycles in this digraph.

13. The White Pine subdivision is a rectangular area six blocks long and two blocks wide. Streets alternate between one way and two way as shown in Fig. 8-27. Draw a digraph that represents the traffic flow in this neighborhood.

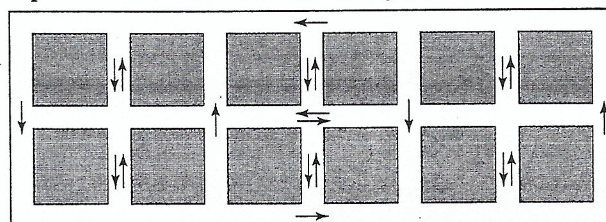


FIGURE 8-27

89

14. A mathematics textbook consists of 10 chapters. Although many of the chapters are independent of the others, some chapters require that previous chapters be covered first. The following list describes all the chapter dependences: Chapter 1 is a prerequisite to Chapters 3 and 5; Chapters 2 and 9 are both prerequisites to Chapter 10; Chapter 3 is a prerequisite to Chapter 6; and Chapter 4 is a prerequisite to Chapter 7, which in turn is a prerequisite to Chapter 8. Draw a digraph that describes the dependences among the chapters in the book.

15. The digraph in Fig. 8-28 is a *respect* digraph. That is, the vertices of the digraph represent members of a group and an arc XY represents that X respects Y.

(a) If you had to choose one person to be the leader of the group, whom would you pick? Explain.

(b) Who would be the worst choice to be the leader of the group? Explain.

(c) Assume that you know the respect digraph of a group of individuals, and that is the only information available to you. Which individual would be the most reasonable choice for leader of the group?

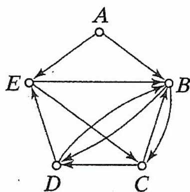


FIGURE 8-28

16. The digraph in Fig. 8-29 is an example of a *tournament* digraph. In this example the vertices of the digraph represent five volleyball teams in a round-robin tournament (i.e., every team plays every other team). An arc XY represents that X defeated Y in the tournament. (*Note*: There are no ties in volleyball.)

(a) Which team won the tournament? Explain.

(b) Which team came in last in the tournament? Explain.

(c) Suppose that you are given the tournament digraph of some tournament. What does the *indegree* of a vertex represent? What does the *outdegree* of a vertex represent?

(d) If T denotes the tournament digraph for a round-robin tournament with N teams, then for any vertex X in T, the indegree of X plus the outdegree of $X = N - 1$. Explain why.

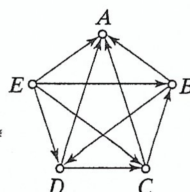


FIGURE 8-29

17. As part of its extended concert tour around the United States, the famous rock band Angelface will be giving a concert at the Smallville Bowl next month. This is a big event for Smallville, and the whole town is buzzing. The digraph in Fig. 8-30 shows the hyperlinks connecting the following six Web sites: (1) www.angelface.com (the rock band's Web site), (2) www.ticketmonster.com (the Web site for TicketMonster, the only ticket agency licensed to sell tickets to the concert), (3) www.knxrock.com (the Web site of Smallville rock station KNXR), (4) www.joetheblogger.com (the personal blog of Joe Fan, a local rock and roll aficionado), and (5) and (6) www.SmallvilleInn.com and www.SmallvilleSuites.com (the Web sites of two local sister hotels owned by the same company and both offering a special rate for out-of-town fans coming to the concert).

(a) Make an educated guess as to which vertex is the most likely to represent (1), the rock band's Web site, and explain the reasoning behind your answer.

(b) Make an educated guess as to which vertex is the most likely to represent (2), the TicketMonster Web site, and explain the reasoning behind your answer.

(c) Make an educated guess as to which vertex is the most likely to represent (3), the Web site of rock station KNXR, and explain the reasoning behind your answer.

(d) Make an educated guess as to which vertex is the most likely to represent (4), Joe Fan's blog, and explain the reasoning behind your answer.

(e) Make an educated guess as to which two vertices are the most likely to represent (5) and (6), the two Smallville sister hotel Web sites.

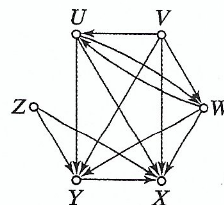


FIGURE 8-30

18. Wobble, a start-up company, is developing a search engine for the Web. Given a particular search word, say Angelface (the name of a rock band), Wobble's strategy is to find all the Web sites containing the term *Angelface,* and then look at Web sites to which those Web sites link, to obtain a ranked list of search results. In other words, if there are lots of links pointing to Web site X from Web sites that mention Angelface, then it is likely that X contains lots of useful information about Angelface. In the digraph shown in Fig. 8-31, the vertices represent Web pages that contain the word *Angelface* and the arcs represent hyperlinks from one Web site to another.

(a) Which Web sites would show up first, second, and third in the search results? Explain your answer.

90

(b) One of the vertices of the digraph is the official Angelface Web site. Which do you think it is? Explain your answer.

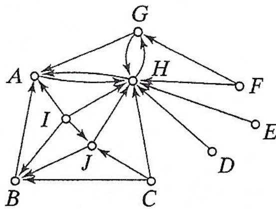


FIGURE 8-31

19. A project consists of eight tasks labeled A through H. The processing time (P-time) and precedence relations for each task are given in Table 8-3. Draw the project digraph.

Task	P-time	Precedent tasks
A	3	
B	10	C, F, G
C	2	A
D	4	G
E	5	C
F	8	A, H
G	7	H
H	5	

■ TABLE 8-3

20. A project consists of eight tasks labeled A through H. The processing time (P-time) and precedence relations for each task are given in Table 8-4. Draw the project digraph.

Task	P-time	Precedent tasks
A	5	C
B	5	C, D
C	5	
D	2	G
E	15	A, B
F	6	D
G	2	
H	2	G

■ TABLE 8-4

21. Eight experiments need to be carried out. One of the experiments requires 10 hours to complete, two of the experiments require 7 hours each to complete, two more require 12 hours each to complete, and three of the experiments require 20 hours each to complete. In addition, none of the 20-hour experiments can be started until both of the 7-hour experiments have been completed, and the 10-hour experiment cannot be started until both of the 12-hour experiments have been completed. Draw a project digraph for this scheduling problem.

22. Every fifty thousand miles an aircraft undergoes a complete inspection that involves 10 different diagnostic "checks." Three of the checks require 4 hours each to complete, three more require 7 hours each to complete, and four of the checks require 15 hours each to complete. Moreover, none of the 15-hour checks can be started until all the 4-hour checks have been completed. Draw a project digraph for this scheduling problem.

23. Apartments Unlimited is an apartment maintenance company that refurbishes apartments before new tenants move in. Table 8-5 shows the tasks for refurbishing a one-bedroom apartment, their processing times (in hours), and their precedent tasks. Draw a project digraph for the project.

Tasks	Label (P-time)	Precedent tasks
Bathrooms (clean)	$B(2)$	P
Carpets (shampoo)	$C(1)$	S, W
Filters (replace)	$F(0.5)$	
General cleaning	$G(2)$	B, F, K
Kitchen (clean)	$K(3)$	P
Lights (replace bulbs)	$L(0.5)$	
Paint	$P(6)$	L
Smoke detectors (battery)	$S(0.5)$	G
Windows (wash)	$W(1)$	G

■ TABLE 8-5

24. A ballroom is to be set up for a large wedding reception. Table 8-6 shows the tasks to be carried out, their processing times (in hours), and their precedent tasks. Draw a project digraph for the project of setting up for the wedding reception.

Tasks	Label (P-time)	Precedent tasks
Set up tables and chairs	$TC(1.5)$	
Set tablecloths and napkins	$TN(0.5)$	TC
Make flower arrangements	$FA(2.2)$	
Unpack crystal, china, and flatware	$CF(1.2)$	
Put place settings on table	$PT(1.8)$	TN, CF
Arrange table decorations	$TD(0.7)$	FA, PT
Set up the sound system	$SS(1.4)$	
Set up the bar	$SB(0.8)$	TC

TABLE 8-6

25. Preparing a banquet for a large number of people requires careful planning and the execution of many individual tasks. Imagine preparing a four-course dinner for a party of 20 friends. Suppose that the project is broken down into 10 individual tasks. The tasks and their processing times (in hours) are as follows: $A(1.5)$, $B(1)$, $C(0.5)$, $D(1.25)$, $E(1.5)$, $F(1)$, $G(3)$, $H(2.5)$, $I(2)$, and $J(1.25)$. The precedence relations are as follows: B cannot be started until A and D have been completed, C cannot be started until B and J have been completed, E cannot be started until D and G have been completed, F cannot be started until E and J have been completed, H cannot be started until F and C have been completed, and J cannot be started until I has been completed. Draw a project digraph for this culinary project.

26. Speedy Landscape Service has a project to landscape the garden of a new model home. The tasks that must be performed and their processing times (in hours) are: collect and deliver rocks, $R(4)$; collect and deliver soil (two loads), $S_1(3)$ and $S_2(3)$; move and position rocks, $RM(4)$; grade soil, $SG(5)$; seed lawn, $SL(3)$; collect and deliver bushes and trees, $B(2)$; plant bushes and trees, $BP(1)$; water the new plantings, $W(1)$; and lay mulch, $M(2)$. The precedence relations between tasks are $R \rightarrow RM$, $RM \rightarrow SG$, $S_1 \rightarrow SG$, $S_2 \rightarrow SG$, $SG \rightarrow SL$, $SG \rightarrow B$, $B \rightarrow BP$, $SL \rightarrow W$, $BP \rightarrow W$, and $BP \rightarrow M$. Draw a project digraph for this landscaping project.

8.3 Priority-List Scheduling

Exercises 27 through 30 refer to a project consisting of 11 tasks (A through K) with the following processing times (in hours): $A(10)$, $B(7)$, $C(11)$, $D(8)$, $E(9)$, $F(5)$, $G(3)$, $H(6)$, $I(4)$, $J(7)$, $K(5)$.

27. (a) A schedule with $N = 3$ processors produces finishing time $Fin = 31$ hours. What is the total idle time for all the processors?

(b) Explain why a schedule with $N = 3$ processors must have finishing time $Fin \geq 25$ hours.

28. (a) A schedule with $N = 5$ processors has finishing time $Fin = 19$ hours. What is the total idle time for all the processors?

(b) Explain why a schedule with $N = 5$ processors must have finishing time $Fin \geq 15$ hours.

29. Explain why a schedule with $N = 6$ processors must have finishing time $Fin \geq 13$ hours.

30. (a) Explain why a schedule with $N = 10$ processors must have finishing time $Fin \geq 11$ hours.

(b) Explain why it doesn't make sense to put more than 10 processors on this project.

Exercises 31 and 32 refer to the Martian Habitat Unit scheduling project with $N = 2$ processors discussed in Example 8.8. The purpose of these exercises is to fill in some of the details left out in Example 8.8.

31. For the priority list in Example 8.8, show the priority-list status at time $T = 26$.

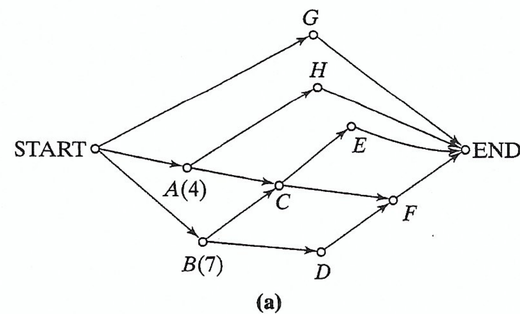
32. For the priority list in Example 8.8, show the priority-list status at time $T = 32$.

33. Consider the project digraph shown in Fig. 8-32(a). (Processing times not relevant to this question have been omitted.) Suppose the project is to be scheduled with two processors, P_1 and P_2, and the priority list is A, B, C, D, E, F, G, H. The schedule starts as shown in Fig. 8-32(b).

(a) Can C be the next task assigned to P_1 at $T = 4$? If not, explain why not.

(b) Can D be the next task assigned to P_1 at $T = 4$? If not, explain why not.

(c) Can H be the next task assigned to P_1 at $T = 4$? If not, explain why not.



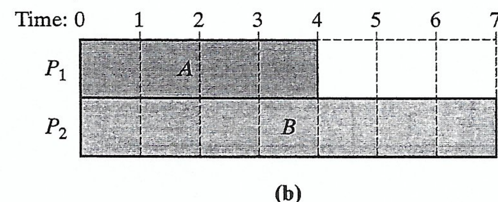


FIGURE 8-32

34. Consider the project digraph shown in Fig. 8-33(a). (Processing times not relevant to this question have been omitted.) Suppose the project is to be scheduled with two

processors, P_1 and P_2, and the priority list is $A, B, C, D, E, F, G, H, I$. The schedule starts as shown in Fig. 8-33(b).

(a) Can D be the next task assigned to P_1 at $T = 6$? If not, explain why not.

(b) Can E be the next task assigned to P_1 at $T = 6$? If not, explain why not.

(c) Can G be the next task assigned to P_1 at $T = 6$? If not, explain why not.

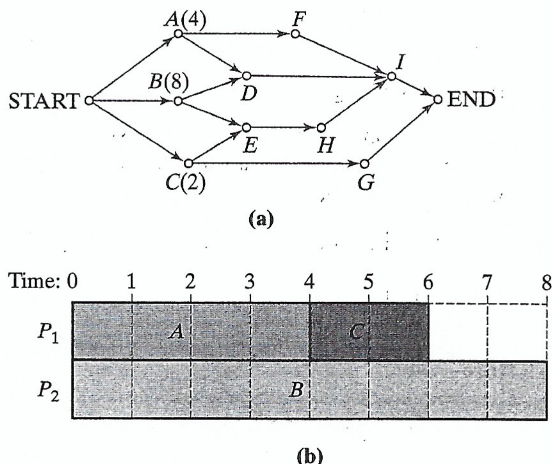


FIGURE 8-33

35. Using the priority list D, C, A, E, B, G, F, schedule the project described by the project digraph shown in Fig. 8-34 using $N = 2$ processors. Show the project timeline and give its finishing time.

36. Using the priority list G, F, E, D, C, B, A, schedule the project described by the project digraph shown in Fig. 8-34 using $N = 2$ processors. Show the project timeline and give its finishing time.

37. Using the priority list D, C, A, E, B, G, F, schedule the project described by the project digraph shown in Fig. 8-34 using $N = 3$ processors. Show the project timeline and give its finishing time.

38. Using the priority list G, F, E, D, C, B, A, schedule the project described by the project digraph shown in Fig. 8-34 using $N = 3$ processors. Show the project timeline and give its finishing time.

39. Explain why the priority lists A, B, D, F, C, E, G and F, D, A, B, G, E, C produce the same schedule for the project shown in Fig. 8-34.

40. Explain why the priority lists E, G, C, B, A, D, F and G, C, E, F, D, B, A produce the same schedule for the project shown in Fig. 8-34.

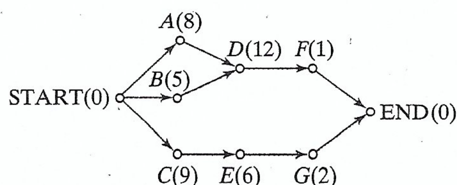


FIGURE 8-34

8.4 The Decreasing-Time Algorithm

41. Use the decreasing-time algorithm to schedule the project described by the project digraph shown in Fig. 8-34 using $N = 2$ processors. Show the timeline for the project, and give the project finishing time.

42. Use the decreasing-time algorithm to schedule the project described by the project digraph shown in Fig. 8-34 using $N = 3$ processors. Show the timeline for the project, and give the project finishing time.

43. Use the decreasing-time algorithm to schedule the project described by the project digraph shown in Fig. 8-35 using $N = 3$ processors. Show the timeline for the project, and give the project finishing time.

44. Use the decreasing-time algorithm to schedule the project described by the project digraph shown in Fig. 8-35 using $N = 2$ processors. Show the timeline for the project, and give the project finishing time.

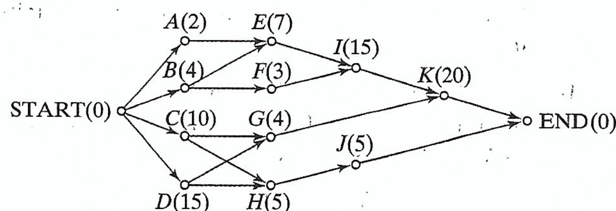


FIGURE 8-35

45. Consider the project described by the project digraph shown in Fig. 8-36, and assume that you are to schedule this project using $N = 2$ processors.

(a) Use the decreasing-time algorithm to schedule the project. Show the timeline for the project and the finishing time *Fin*.

(b) Find an optimal schedule and the optimal finishing time *Opt*.

(c) Use the relative error formula $\varepsilon = \frac{Fin - Opt}{Opt}$ to find the relative error of the schedule found in (a), and express your answer as a percent.

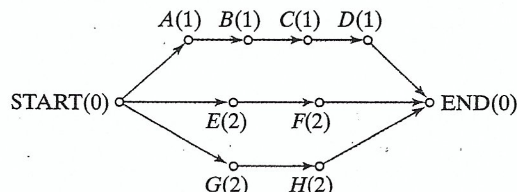


FIGURE 8-36

46. Consider the project described by the project digraph shown in Fig. 8-37, and assume that you are to schedule this project using $N = 2$ processors.

(a) Use the decreasing-time algorithm to schedule the project. Show the timeline for the project and the finishing time *Fin*.

(b) Find an optimal schedule and the optimal finishing time *Opt*.

93

(c) Use the relative error formula $\varepsilon = \frac{Fin - Opt}{Opt}$ to find the relative error of the schedule found in (a), and express your answer as a percent.

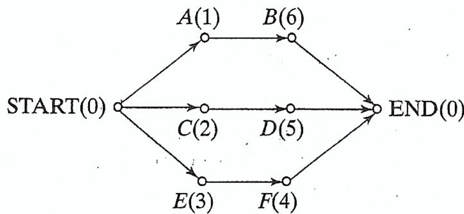


FIGURE 8-37

47. Consider the project described by the project digraph shown in Fig. 8-38, and assume that you are to schedule this project using $N = 3$ processors.

(a) Use the decreasing-time algorithm to schedule the project. Show the timeline for the project and the finishing time *Fin*.

(b) Find an optimal schedule and the optimal finishing time *Opt*.

(c) Use the relative error formula $\varepsilon = \frac{Fin - Opt}{Opt}$ to find the relative error of the schedule found in (a), and express your answer as a percent.

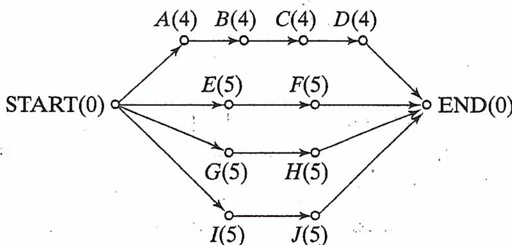


FIGURE 8-38

48. Consider the project described by the project digraph shown in Fig. 8-39, and assume that you are to schedule this project using $N = 4$ processors.

(a) Use the decreasing-time algorithm to schedule the project. Show the timeline for the project and the finishing time *Fin*.

(b) Find an optimal schedule. (*Hint*: $Opt = 17$.)

(c) Use the relative error formula $\varepsilon = \frac{Fin - Opt}{Opt}$ to find the relative error of the schedule found in (a), and express your answer as a percent.

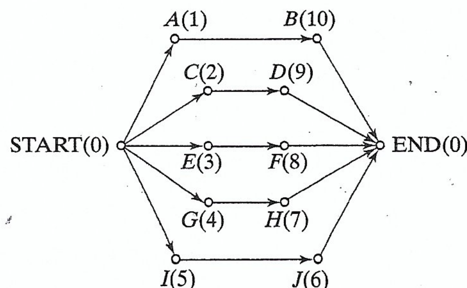


FIGURE 8-39

8.5 Critical Paths and the Critical-Path Algorithm

49. Consider the project digraph shown in Fig. 8-40.

(a) Use the backflow algorithm to find the critical time for each vertex.

(b) Find the critical path for the project.

(c) Schedule the project with $N = 2$ processors using the critical-path algorithm. Show the timeline, and give the project finishing time.

(d) Explain why the schedule obtained in (c) is optimal.

50. Consider the project digraph shown in Fig. 8-40.

(a) Schedule the project with $N = 3$ processors using the critical-path algorithm. Show the timeline, and give the project finishing time.

(b) Explain why the schedule obtained in (a) is optimal.

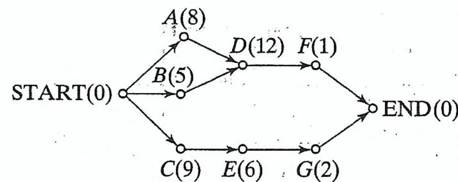


FIGURE 8-40

51. Consider the project digraph shown in Fig. 8-41.

(a) Find the critical path for the project.

(b) Schedule the project with $N = 3$ processors using the critical-path algorithm. Show the timeline, and give the project finishing time.

52. Consider the project digraph shown in Fig. 8-41.

(a) Use the backflow algorithm to find the critical time for each vertex.

(b) Schedule the project with $N = 2$ processors using the critical-path algorithm. Show the timeline, and give the project finishing time.

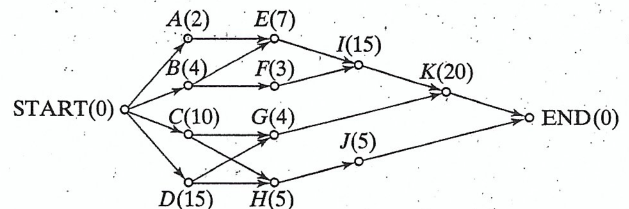


FIGURE 8-41

53. Schedule the Apartments Unlimited project given in Exercise 23 (Table 8-5) with $N = 2$ processors using the critical-path algorithm. Show the timeline, and give the project finishing time. (Note that the project digraph was done in Exercise 23.)

54. Schedule the project given in Exercise 24, Table 8-6 (setting up a ballroom for a wedding) with $N = 3$ processors using the critical-path algorithm. Show the timeline, and give the project finishing time. (Note that the project digraph was done in Exercise 24.)

94

55. Consider the project described by the project digraph shown in Fig. 8-42.

(a) Find the critical path and critical time for the project.

(b) Find the critical-time priority list.

(c) Schedule the project with $N = 2$ processors using the critical-path algorithm. Show the timeline and the project finishing time.

(d) Find an optimal schedule for $N = 2$ processors.

(e) Use the relative error formula $\varepsilon = \frac{Fin - Opt}{Opt}$ to find the relative error of the schedule found in (c).

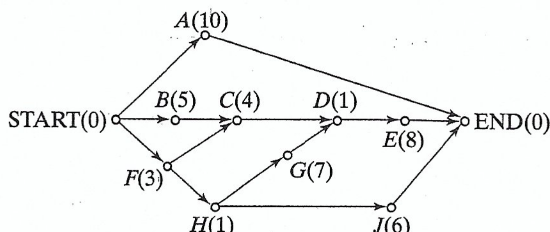


FIGURE 8-42

56. Consider the project digraph shown in Fig. 8-43, with the processing times given in hours.

(a) Find the critical path and critical time for the project.

(b) Find the critical-time priority list.

(c) Schedule the project with $N = 2$ processors using the critical-path algorithm. Show the timeline and the project finishing time.

(d) Explain why any schedule for $N = 2$ processors with finishing time $Fin = 22$ must be an optimal schedule.

(e) Schedule the project with $N = 3$ processors using the critical-path algorithm. Show the timeline and the project finishing time.

(f) Explain why the schedule found in (e) is an optimal schedule for three processors.

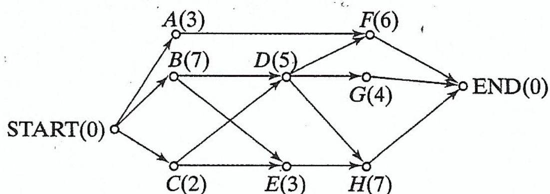


FIGURE 8-43

JOGGING

57. Explain why, in any digraph, the sum of all the indegrees must equal the sum of all the outdegrees.

58. Symmetric and totally asymmetric digraphs. A digraph is called *symmetric* if, whenever there is an arc from vertex X to vertex Y, there is *also* an arc from vertex Y to vertex X. A digraph is called *totally asymmetric* if, whenever there is an arc from vertex X to vertex Y, there *is not* an arc from vertex Y to vertex X. For each of the following, state whether the digraph is symmetric, totally asymmetric, or neither.

(a) A digraph representing the streets of a town in which all streets are one-way streets.

(b) A digraph representing the streets of a town in which all streets are two-way streets.

(c) A digraph representing the streets of a town in which there are both one-way and two-way streets.

(d) A digraph in which the vertices represent a group of men, and there is an arc from vertex X to vertex Y if X is a brother of Y.

(e) A digraph in which the vertices represent a group of men, and there is an arc from vertex X to vertex Y if X is the father of Y.

59. For the schedule shown in Fig. 8-44, find a priority list that generates the schedule. (*Note*: There is more than one possible answer.)

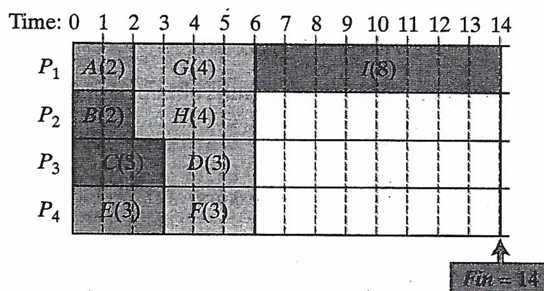


FIGURE 8-44

Exercises 60 through 62 introduce three paradoxes that can occur in scheduling. In all three exercises we will use the project described by the project digraph shown in Fig. 8-45. Assume that processing times are given in hours. [Source: Garey, M.R. et al. "Performance Guarantees for Scheduling Algorithms." Operations Research, 26 (1978)]

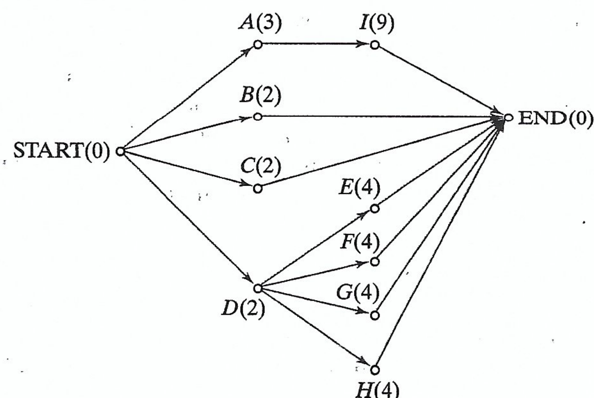


FIGURE 8-45

60. (a) Using the priority list $A, B, C, D, E, F, G, H, I$, schedule the project shown in Fig. 8-45 using $N = 3$ processors.

(b) Using the same priority list as in (a), schedule the project using $N = 4$ processors.

(c) There is a paradox in the results obtained in (a) and (b). Describe the paradox.

61. (a) Using the priority list $A, B, C, D, E, F, G, H, I$, schedule the project shown in Fig. 8-45 using $N = 3$ processors.