Critical Path Method

Allen D. Holliday
March 18, 2009
Computer Science Department
California State University, Fullerton

Introduction

Planning a project usually involves dividing it into a number of small tasks that can be assigned to individuals or teams. The project's schedule depends on the duration of these tasks and the sequence in which they are arranged. This sequence can be driven by several factors: customer deadlines, availability of personnel or resources, and dependencies among tasks. The last factor is the subject of this paper—in particular, how this sequence can affect the project's duration and its finish date.

A schedule isn't an arbitrary sequence of tasks, and it isn't just a convenient arrangement that maximizes the use of resources for the shortest timeline. It must consider precedence—the relationship between the start and finish dates of interdependent tasks, where one task can't start until one or more other tasks are finished. These precedence relationships determine which tasks can be overlapped and which ones must be serial.

The purpose of any schedule is to define when tasks will begin and finish, but it's important to remember that the purpose of project is not merely to perform tasks but to deliver a product. The ultimate stakeholder question about the schedule is "When will this project finish?"

The project manager's simple answer could be "Look at the finish date of the last task." The stakeholder's response is often another question: "How do you know that these tasks will finish when the schedule says they will?" Of course, the project manager can't guarantee that they will—delays happen for many reasons. He must ask himself "Which tasks endanger the finish date? What needs the most attention so that the project won't be delayed? If I have to finish the project early, which tasks need to be shortened?"

Some of the project tasks directly affect the finish date—if they slip, the project slips by the same amount (the proverbial "day-for-day slip"). Other tasks may not have the same effect. There may be tasks of different durations that are scheduled on parallel paths. If a task on a shorter path slips, it will not affect the finish date because the longer path are the determining factor. The longest path from the project's first task to its last is called the *critical path*.

A diagram called a Precedence Network shows the tasks and their scheduling dependencies. The Critical Path Method (CPM) identifies the tasks on the critical path within this network of tasks. There can actually be more than one critical path and CPM will reveal every one of them. For simplicity, this paper will use the singular word "path."

Precedence networks are often called PERT charts. This isn't correct—PERT charts are a more comprehensive derivative of precedence networks that allow probabilistic analysis of a project schedule.

Precedence Network Example

Here's an example. The boxes represent tasks. The task ID is at the top, the d: number is its duration in whatever time scale is convenient (we'll use days), and the ES: and EF: numbers are the Early Start and Early Finish day numbers. The reason for adding the word Early will be explained in a moment.

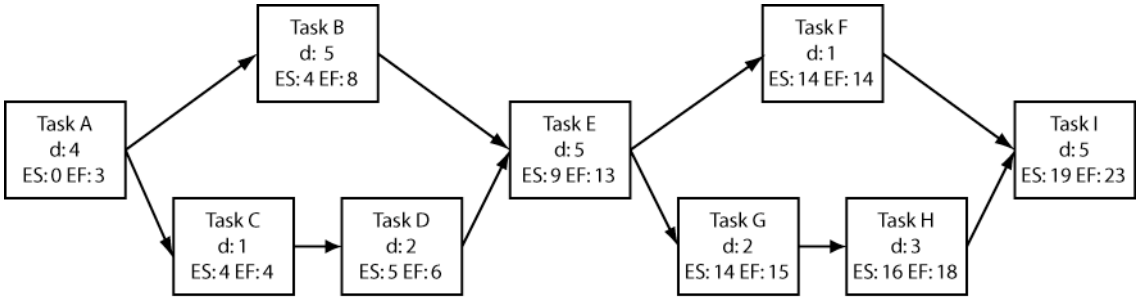

Figure 1. Precedence network with Early Starts and Early Finishes only.

A task's finish date is its start date plus its duration, minus one. Subtracting one day accounts for the convention that a task starts at the beginning of a day and finishes at the end of a day. Its successor starts on the next day. A 1-day task starts and finishes on the same day.

If we look at this diagram carefully, we can see that it will take 24 days to finish this project. Here's the step-by-step reasoning:

- Task A takes 4 days.
- There are two paths from A to E, but the one through B is longer, 5 days instead of 3 days through C and D.
- Task E takes another 5 days.
- There are two paths from E to I, but the one through G and H is longer, 5 days instead of 1 days through F.
- Task I takes another 5 days.
- 4 + 5 + 5 + 5 + 5 = 24 days

The 16-day path (A-B-E-G-H-I) is the critical path—every task on this path is critical to the project's schedule. If any of these tasks takes longer than its planned duration, the project will be delayed by the same amount. If we want to finish early, these are the tasks that must be considered for shortening. Reducing the duration of tasks C, D, or F will not change the project's finish date.

Figure 2 below shows the timeline for this network.

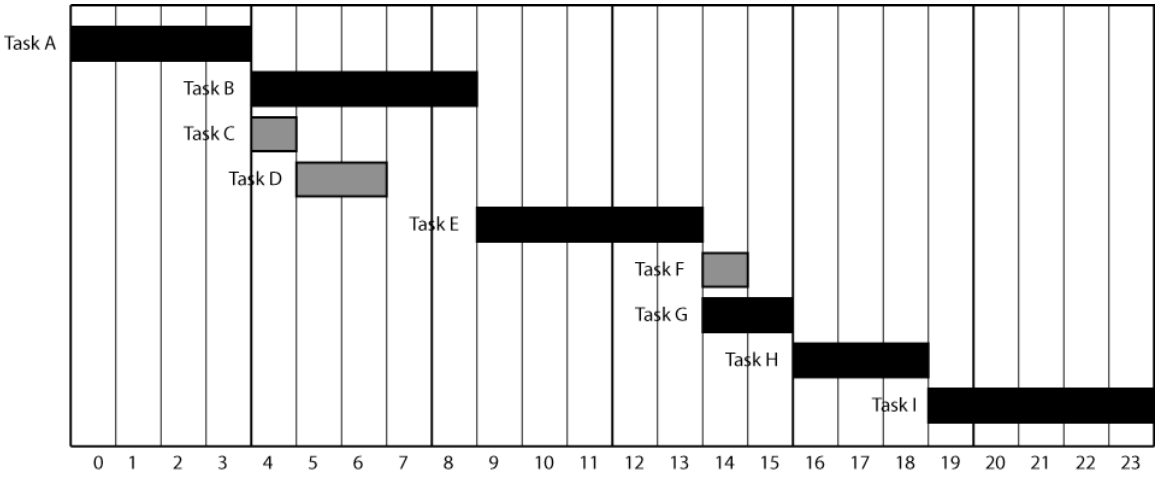

Figure 2. Timeline for Early Start, Early Finish network.

The darkest bars are the tasks that determine how long the project will take. They are on the critical path.

Critical Path Method

Determining the critical path by inspection takes a little thought, even for a small network. Inspection isn't practical for a large network which may have hundreds or even thousands of tasks. The Critical Path Method is an algorithmic approach to finding critical paths.

Let's look at the network again, especially at the differences between tasks that aren't on the critical path and those that are. In the segment between A and E, where C and D are, we notice that C and D require only 3 days. They could be delayed 1 day without delaying E, since task E's start depends on B's finish. Task C could start immediately after A finishes, or 1 day later; task D could also start 1 day later than its original start date, either immediately after C or with a 1-day gap.

This observation leads us to the idea that the network's tasks should have two start dates, an early start and a late start, with a corresponding pair of finish dates. The difference between the early and late dates, the amount that a task can be delayed, is called *float*. For tasks on the critical path, the float will be zero, since they can't be delayed. Note that the difference between the two start dates will be the same as the difference between the two finish dates. It doesn't matter which pair is used to calculate float.

The more accurate term is *total float*. There are actually three types of float, as explained in a later section of this paper, but *total float* is by far the most commonly used measure and is often shortened to *float*. Another term that's used in CPM descriptions is *slack*, which is synonymous with *float.* The two terms have the very same meaning. (Americans prefer *float*, the British prefer *slack*.)

There are five steps in CPM:

1. Creating a network showing the task names, durations, and precedence relationships, where the four dates are empty, except for the early start of the first task.
2. Making a Forward Pass through the network, determining the early start and early finish for every task.
3. Making a Backward Pass through the network, determining the late start and late finish for every task.
4. Calculating the float for every task.

5. Finding the paths that consist entirely of tasks with zero float. These paths will go from the very first task to the very last one.

In the remainder of this paper, the following abbreviations will be used for the four dates:

- ES for Early Start
- EF for Early Finish
- LS for Late Start
- LF for Late Finish

Although this paper uses day numbers for simplicity, the usual practice is to use calendar dates. They're more readable in real projects and more amenable to adjustments for situations such as weekends and holidays.

Forward Pass

The forward pass goes from the initial task (the task with no predecessors) to the final task (the one with no successors), visiting every task in every path and setting the ES and EF dates on the tasks.

The algorithm is similar to graph theory's depth-first search, except that the forward pass follows every path from initial to final task, while depth-first search stops when it arrives at a task that it's already visited. When the forward pass arrives at a task, it may change that task's ES and EF dates, and that change must be carried forward. If the task has already been visited and this visit doesn't change its dates, there is no need to go forward from this task, since nothing beyond it will change either.

During the forward pass, a task may be visited several times as different paths through the network are followed. A task's ES is determined by the predecessor task with the latest EF, since a task can't start until all of its predecessors have finished.

The steps of this pass are:

1. Set the initial task's ES = 0.
2. Calculate the task's EF = ES + duration – 1.
3. Calculate an ES for the task's successors; $ES_{successor} = EF_{task} + 1$.
4. Visit all of the task's successors. If the successor's current ES is less than the ES from step 3:
   a. Replace the current ES with the step 3 value.
   b. Repeat steps 2–4 for the current task's successors, unless there are no more because the last task has been reached.
5. Go back up the path just completed to a task with unvisited successors.
6. Repeat steps 4–5 until all successors have been visited, which will cover all paths from those successors to the final task.

The example in Figure 1 is the result of a forward pass. Here is a step-by-step explanation of the forward pass, showing the tasks visited in order and actions taken at each:

1. Task A: $ES_A = 0$, $EF_A = ES_A + d_A - 1 = 3$
2. Task B: $ES_B = EF_A + 1 = 4$, $EF_B = ES_B + d_B - 1 = 8$
3. Task E: $ES_E = EF_B + 1 = 9$, $EF_E = ES_E + d_E - 1 = 13$
4. Task F: $ES_F = EF_E + 1 = 14$, $EF_F = ES_F + d_F - 1 = 14$

5. Task I: $ES_I = EF_F + 1 = 15$, $EF_I = ES_I + d_I - 1 = 19$

(Back up to E, next task is G)

6. Task G: $ES_G = EF_E + 1 = 14$, $EF_G = ES_G + d_G - 1 = 15$
7. Task H: $ES_H = EF_G + 1 = 16$, $EF_H = ES_H + d_H - 1 = 18$
8. Task I: $ES_I = EF_H + 1 = 19$, $EF_I = ES_I + d_I - 1 = 23$, replacing step 5's smaller ES and EF

(Back up to E, no next task; back up to A, next task is C)

9. Task C: $ES_C = EF_A + 1 = 4$, $EF_C = ES_C + d_C - 1 = 4$
10. Task D: $ES_D = EF_C + 1 = 5$, $EF_D = ES_D + d_D - 1 = 6$
11. Task E: possible new $ES_E = EF_D + 1 = 7$, $EF_E = ES_E + d_E - 1 = 11$, smaller than step 3's ES and EF, no replacement (no change to propagate forward)

(Back up to A, no next task)

12. The forward pass stops here, since there are no more paths to probe.

Notice the multiple visits to several tasks over different paths.

Backward Pass

The backward pass goes from the final task to the initial task, visiting every task in every path and setting the LS and LF dates on the tasks.

It's similar to the forward pass in that when it arrives at a task, it may change that task's LS and LF dates and that change must be carried back. The difference is:

- The forward pass sets the task's latest ES, as determined by the EFs of its predecessors
- The backward pass sets the task's earliest LF, as determined by the LSs of its successors.

The reason for the backward pass's rule for setting LF is not as obvious as the forward pass's rule. Any start date, ES or LS, must be after the corresponding finish dates of all of the task's predecessors. To maintain this consistency, the backward pass must set a task's LF to a value that's earlier than the LS of any of that task's successors.

If the task has already been visited and this visit doesn't change its dates, there is no need to go backward from this task, since nothing beyond it will change either.

The steps of this pass are:

1. Set the final task's LF = the task's EF. (The final task is always on the critical path and has no float. This means that its EF and LF are equal.)
2. Calculate the task's LS = LF - duration + 1
3. Calculate an LF for the task's predecessors; $LF_{predecessor} = LS_{task} - 1$
4. Visit all of the task's predecessors. If the predecessor's current LF is greater than the LF from step 3:
   a. Replace the current LF with the step 3 value.
   b. Repeat steps 2–4 for the current task's predecessors, unless there are no more because the first task has been reached.
5. Go back down the path just completed to a task with unvisited predecessors.

6. Repeat steps 4–5 until all predecessors have been visited, which will cover all paths from those predecessors to the initial task.

When the backward pass is finished, the initial task's EF and LF should be equal. It is on the critical path, just like the final task, and it has no float. If EF and LF are different, an error was made in one of the passes.

Figure 3 below shows the result of the backward pass on the example shown earlier.

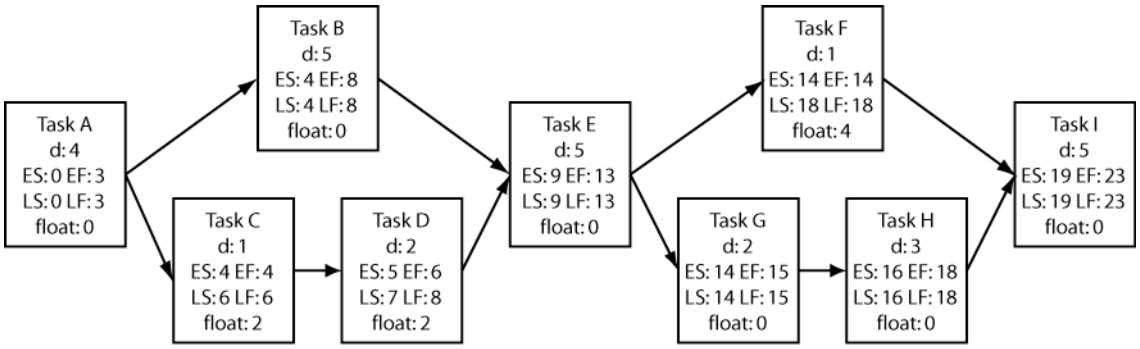

Figure 3. Precedence network with total float.

Here is a step-by-step explanation of the backward pass, showing the tasks visited in order and actions taken at each:

1. Task I: $LF_I = EF_I$, = 23 $LS_I = LF_I - d_I + 1 = 19$
2. Task F: $LF_F = LS_I - 1 = 18$, $LS_F = LF_F - d_F + 1 = 18$
3. Task E: $LF_E = LS_F - 1 = 17$, $LS_E = LF_E - d_E + 1 = 13$
4. Task B: $LF_B = LS_E - 1 = 12$, $LS_B = LF_B - d_B + 1 = 8$
5. Task A: $LF_A = LS_B - 1 = 7$, $LS_A = LF_A - d_A + 1 = 4$

(Back up to E, next task is D)

6. Task D: $LF_D = LS_E - 1 = 12$, $LS_D = LF_D - d_D + 1 = 11$
7. Task C: $LF_C = LS_D - 1 = 10$, $LS_C = LF_C - d_C + 1 = 10$
8. Task A: possible new $LF_A = LS_C - 1 = 9$, step 5's value is smaller, no replacement.

(Back up to E, no next task; back up to I, next task is H)

9. Task H: $LF_H = LS_I - 1 = 18$, $LS_H = LF_H - d_H + 1 = 16$
10. Task G: $LF_G = LS_H - 1 = 15$, $LS_G = LF_G - d_G + 1 = 14$
11. Task E: $LF_E = LS_G - 1 = 13$, $LS_E = LF_E - d_E + 1 = 9$, replacing step 3's larger LF and LS
12. Task B: $LF_B = LS_E - 1 = 8$, $LS_B = LF_B - d_B + 1 = 4$, replacing step 4's larger LF and LS
13. Task A: $LF_A = LS_B - 1 = 3$, $LS_D = LF_D - d_D + 1 = 0$, replacing step 5's larger LF and LS

(Back up to E, next task is D)

14. Task D: $LF_D = LS_E - 1 = 8$, $LS_C = LF_C - d_C + 1 = 7$, replacing step 6's larger LF and LS
15. Task C: $LF_C = LS_D - 1 = 6$, $LS_C = LF_C - d_C + 1 = 6$, replacing step 7's larger LF and LS

16. Task A: possible new $LF_A = LS_C - 1 = 5$, current value of 1 is smaller, no replacement.

(Back up to E, no next task; back up to I, no next task)

17. The backward pass stops here, since there are no more paths to probe.

There were multiple visits to several tasks over different paths, just as in the forward pass.

Figure 4 below shows the timeline for this network.

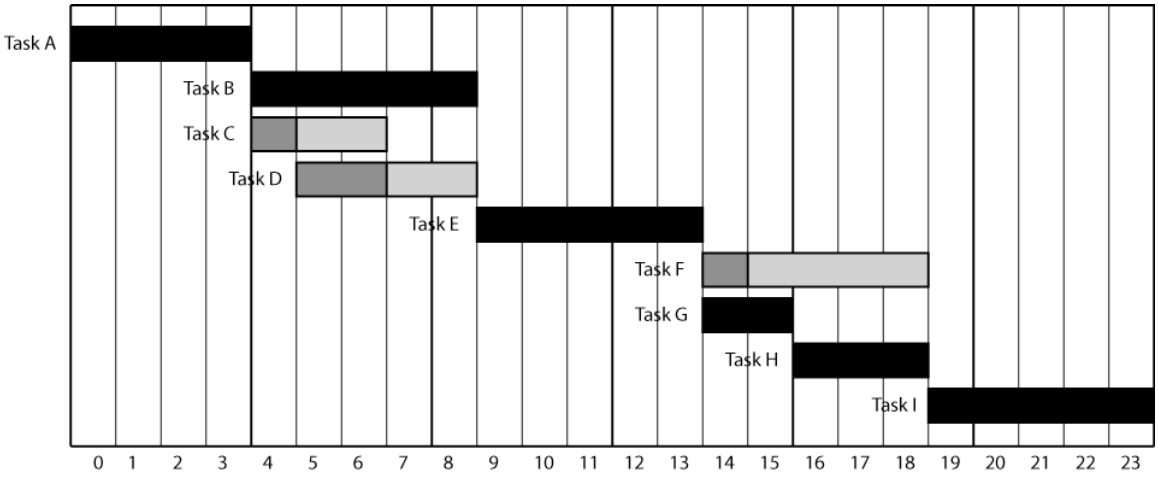

Figure 4. Timeline for precedence network with total float.

This timeline is similar to Figure 2, except that the float for tasks B, C, and F is now shown with the additional light gray bars. Notice that their lengths match the float values in Figure 3.

Finding The Critical Path

A critical path is one where all of its tasks have zero float. There is always at least one, and there can be more. All critical paths begin at the initial task. They can be found by visiting successor tasks in a manner similar to the forward pass, except that only successors with zero float are considered and a probe along a path stops when a task with nonzero float is encountered.

Figure 3 shows the example network's critical path to be A-B-E-G-H-I. If any of these tasks is delayed, the project will finish late. Tasks C and D can be delayed one day without changing task I's (the project's) finish date. Task F can be delayed four days.

Float—Three Types

This paper pointed out earlier that there are three types of float. One of them, total float, is the most commonly used, because it's the one used in the determination of critical paths. The other two, free float and independent float, can be different from total float for tasks that aren't on a critical path.

- Total float is the amount that a task can be delayed without delaying the completion of the project.
- Free and independent floats are the amounts that a task can be delayed without delaying the starts of any of its successors.

The latter two differ in their assumption regarding when the task's predecessors finish. Free float is calculated as if the task's predecessors with the latest EF have finished at that EF, allowing it to start at its

ES. Independent float is calculated as if the task's predecessors with the latest LF have finished at that LF, causing its start to slip past its ES.

Free and independent float can be used to deal with two situations involving the successor of a non-critical task:

- The successor may be on a critical path, and delaying such a successor will delay the project.
- The successor may not be on a critical path, but it may have been assigned a scarce resource. Delaying it may have a cascading effect on the project by upsetting the resource leveling (the process of adjusting the project schedule to smooth out peak demands and eliminate overcommitment of people and equipment)

The formulas for calculating free and independent float are:

$$\text{Free Float} = EES_{successor} - EF - 1$$

$$\text{Independent Float} = EES_{successor} - LLF_{predecessor} - \text{duration} - 1$$

where:

$EES_{successor}$ is the Earliest Early Start of the task's successors.

$LLF_{predecessor}$ is the Latest Late Finish of the task's predecessors.

Free and independent float are significant only on non-critical path segments, where two or more segments join at a task that depends on multiple predecessors. On critical paths, all three float values are zero. Most tasks on non-critical path segments have zero free float, only the last task in the segment before the join will have a nonzero free float. All of the tasks except the last on the segment are starting and finishing as early as they can, there's no time gap between them. Only the last task can slip and only if its successor must start later than its finish because the successor is controlled by a task on another, longer segment.

Independent float can be thought of as placing two constraints on a task, one forward and one backward. It shows how much this task can move (float) without delaying any of its successors, while allowing all of its predecessors to finish as late as possible (at their LFs).

Free Float Example

Figure 5 shows a precedence network where a task has free float.

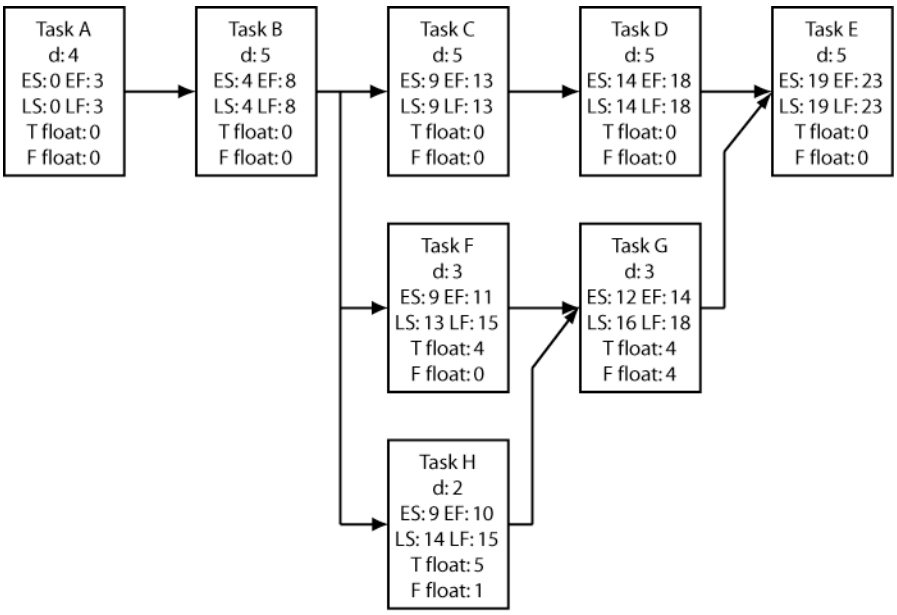

Figure 5. Precedence network with total and free float.

Actually, two tasks, G and H, have free float. Task G's free and total float are equal; it can be delayed by 4 days before it causes its successor, task E to start late. Since task E is on the critical path, this would also cause the project to finish late.

Task H on the other hand can be delayed by 5 days before causing a late project finish, but only by 1 day before delaying the start of task G. Although this doesn't appear to affect the project's finish date, it might have an impact on the resources (people and equipment) allocated to task G.

Figure 6 below shows the timeline corresponding to Figure 5's network.

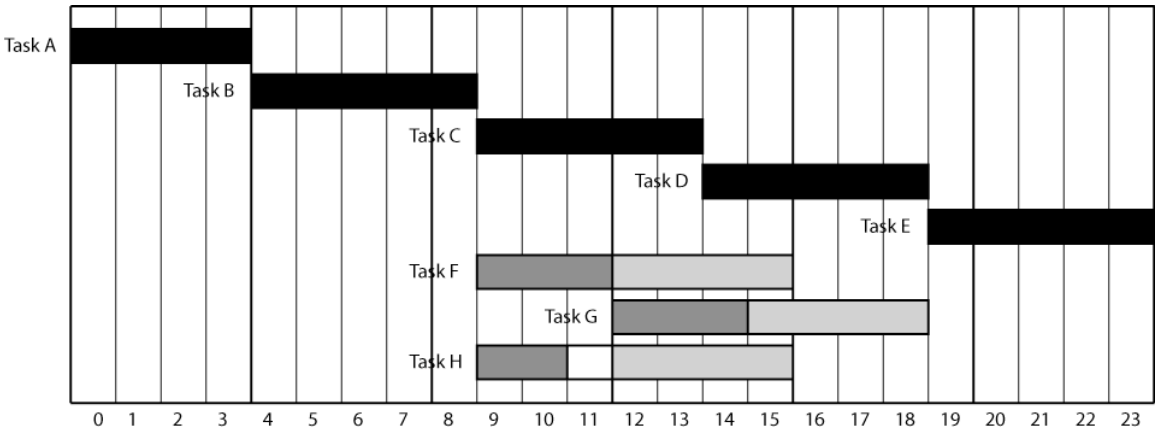

Figure 6. Timeline for precedence network with total and free float.

This timeline shows the floats for non-critical tasks F, G, and H. For task H, the white bar represents its free float; the white bar and the light gray bar combined represent its total float.

Independent Float Example

Figure 7 shows a precedence network where a task has independent float.

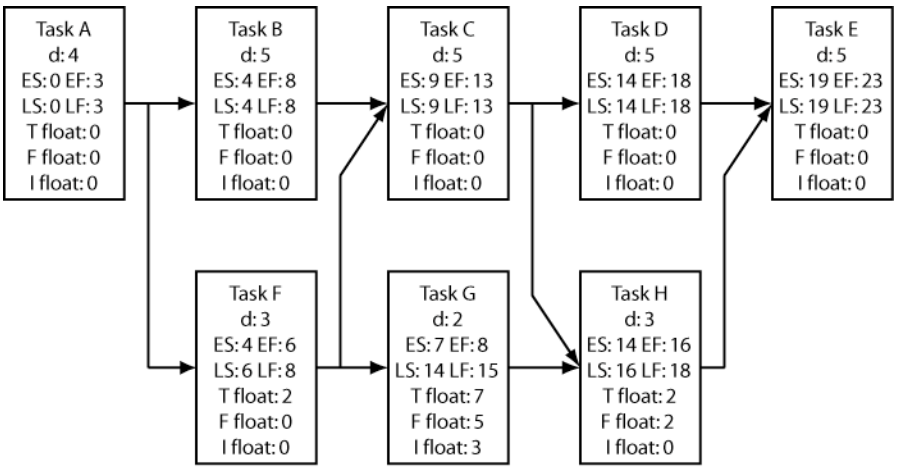

Figure 7. Precedence network with total, free, and independent float.

Task G is the one of interest. Its free float is determined as described previously by examining its effect on its successor, task H. Its independent float is different from its free float because the assumptions about task F are different:

Free float: assume that task F finishes as early as possible.

Independent float: assume that task F finishes as late as possible.

Figure 8 below shows the timeline corresponding to Figure 7's network.

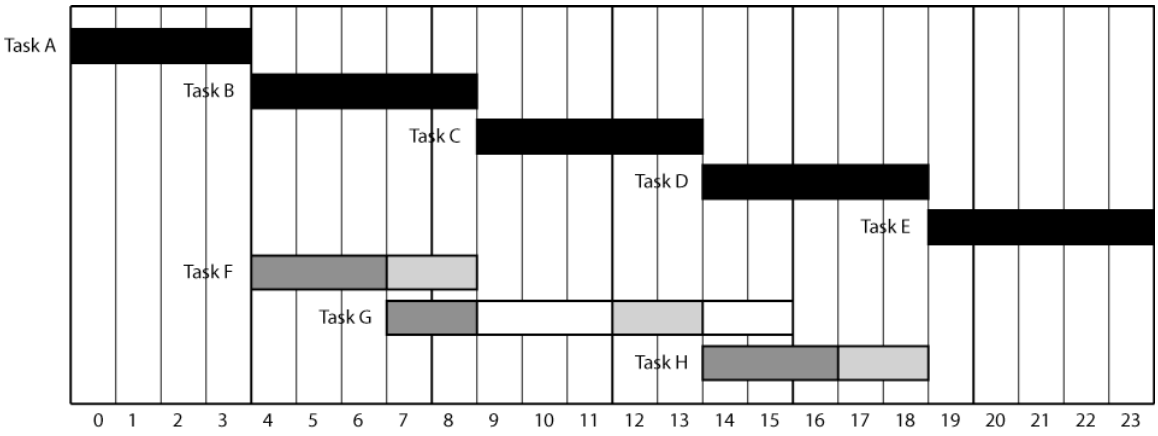

Figure 8. Timeline for precedence network with total, free, and independent float.

It's hard to show all three floats and task G's bars may be slightly confusing. They can be interpreted as follows:

- The left-hand white bar represents independent float.
- The two white bars combined represent free float.
- The white bars and the light gray bar combined represent total float.